Universitatea “Politehnica” din București

Facultatea de Electronică, Telecomunicații și Tehnologia Informației

**Studiul bazelor de date distribuite in sistemul de gestiune al bazelor de date MariaDB**

**Proiect de diplomă**

prezentat ca cerință parțială pentru obținerea titlului de

Inginer în domeniul Calculatoare și Tehnologia

Informației

programul de studii de licență Ingineria

Informației (CTI – INF)

Conducător științific Absolvent

S.l. Ing. Valentin PUPEZESCU Bogdan - Petru MATRAGOCIU

Anul

2014

Cuprins

1. Introducere
2. Baze de date
   1. Noțiuni generale
   2. SQL vs NoSQL
3. MariaDB
   1. Introducere
   2. Replicarea
      1. Introducere
      2. Setarea replicarii
      3. Implementarea replicarii
      4. Topologii de replicare
      5. Administrare si mentenanta
4. Tehnologii utilizate
   1. HTML
   2. CSS
   3. JavaScript
   4. jQuery
   5. Java
      1. Introducere
      2. Sintaxa Java
      3. OOP
      4. Integrarea cu baza de date
5. Proiectarea aplicației
   1. Prezentare generala a interfeței
      1. Sectiunea de autentificare
      2. Secțiunea de stiri si articole
      3. Secțiunea evenimente
      4. Sectinea de intrebari si raspunsuri
      5. Sectiunea de administrator
   2. Structura bazei de date
   3. Testarea replicării
6. Concluzii
7. Bibliografie
8. Anexe

**Lista acronimelor**

SDK - Software Development Kit (Platformă de dezvoltare software)

SQL - Structured Query Language (Limbaj de interogare structurat)

PHP - PHP: Hypertext Preprocessor (Limbaj preprocesare hypertext)

HTML - HyperText Markup

Language (Limbaj de marcaje hypertext)

CSS - Cascading Style SheetLAMP - Linux Apache MySQL PHP

SGBD - Sistem de Gestiune a Bazelor de Date

VARCHAR - Variable Length Character String (Șir de caractere cu lungime variabilă)

BLOB - Binary Large OBject

XML - eXtensible Markup Language

JSON - JavaScript Object Notation

RDBMS - Relational DataBase Management System (Sistem de management al bazelor de date)

BSON - Binary Script Object Notation

UTF-8 - Universal Character Set Transformation Format - 8 bit

DB - DataBase

PNG - Portable Network Graphics

POO/OOP - Object Oriented Programming (Programare Obiect Orientată)

DML - Data Manipulation Language

DDL - Data Definition Language

JSP - Java Server Pages

URL - Uniform Resource Locator

DOM - Document Object Model

AJAX - Asynchronous JavaScript And XML

UI - User Interface (Interfața cu utilizatorul)

JPA - Java Persistence API

**1. Introducere**

In ultimul deceniu internetul si aplicatiile web au cunoscut o crestere exponentiala. Aplicatiile clasice ce necesita descarcarea si instalarea pe calculatorul propriu au scazut ca numar, implementarile lor trecand in cloud. Tendinta este clara de a oferi servicii utilizatorului in interiorul unui browser sau ca aplicatie mobila.

Aproape orice aplicatie foloseste o baza de date. Desi orice programator poate fi capabil sa isi defineasca propriul mod de organizare a datelor, bazele de date ii usureaza munca oferind un mod structurat, testat si tolerant la defecte pentru a stoca datele necesare aplicatiei. Tipurile de baze de date sunt numeroase si prin urmare folosirea celei potrivite pentru aplicatia ce se vrea dezvoltata este o decizie ce trebuie luata cu grija si simt de raspundere. In momentul de fata exista doua mari tipuri de baze de date ce pot fi folosite, cele clasice SQL si sistemul mai nou aparut, NoSQL.

Lucrarea de fata isi propune prezentarea unei stive de tehnologii folosite pentru dezvoltarea unei aplicatii web pentru gestionarea de articole, stiri, evenimente si intrebari legate de domeniul medical si stomatologic. Tehnologiile folosite sunt:

* MariaDB - v. 10.0
* Java - v. 1.7
* HTML - v. 4.0/v. 5.0
* CSS - v. 2.0/v.3.0
* JavaScript/jQuery - v. 1.9

Instrumente folosite:

* Maven 3.0.5
* Git 1.8.4.mysysgit.0
* BitBucket
* IntelliJ IDEA 135.908 EAP
* MySQL Workbench 6.0 CE

Accentul va fi pus pe nivelul bazei de date si posibilitatiele de evitare pierderii datelor si a distribuirii incarcarii. Se va insista pe modalitatile de replicare, avantajele si dezavantajele acestora in sistemul de gestiune al bazelor de date MariaDB, acest sistem fiind la granita dintre SQL si NoSQL.

**2. Baze de date**

Bazele de date uneori numite si banci de date sunt modalitati de stocare a informatiei si a datelor pe un suport extern (dispozitiv de stocare) . Ele ofera posibilitatea exitnderii usoare si a regasirii rapide a datelor. Desi aceste functii pot parea banale, atunci cand se lucreaza cu milioane de elemente ce trebuie accesate simultan prin internet de catre mii de utilizatori raspanduiti pe tot globul si disponibilitatea aplicatiei trebuie sa fie permanenta solutiile cele mai bune nu sunt deloc simple.

De obicei o baza de date este memorata in unul sau mai multe fisiere si sunt manipulate cu ajutorul sistemelor de gestiune a bazelor de date. Cel mai raspandid este modelul relational in care datele sunt memorate in tabele. In afara de tabele o baza de date relationala mai poate contine indecsi, proceduri stocate, declansatori, utilizatori si grupuri de utilizatori, tipuri de date, mecanisme de securitate si gestiune a tranzactiilor.

Alte tipuri de baze de date sunt sunt cele NoSQL i n care datele sunt salvate in format obiectual BSON (Binary JSON – JavaScript Object Notation) sau uneori XML.

**2.1 Notiuni generale**

Un server de baze de date este un program ce poate stoca o cantitate mare de date si este disponibil , accesibil prin ,lombaje de programare cum ar fi Java sau PHP. Serverul, in cazul nostru MariaDB, poate fi folosit atat pe Windows cat si platforme Unix. Nici un sistem de baze de date nu este perfect dar trebuie sa fie ales portrivit nevoilor aplicatiei.

**2.2 SQL vs. NoSQL**

**Modelul realtional**

Modelul relational (SQL) introdus in anii 70 ofera un model matematic usor adaptabil pentru a structura, salva si utiliza date. Datele sunt salvate in tabele si sunt definite realtii intre ele. Deoarece exita de foarte mult timp pe piata programatorii si administratorii de baze de date sunt foarte familiaru cu ele. Chiar daca impun destule conditii si constrangeri asupra datelor ele pot deveni flexibile usor cu un pic de efort din partea utilizatorului.

Cele mai populare sisteme sunt :

* SQLite
* MySQL
* PostgreSQL

**Model-less (Modelul NoSQL)**

Acest model presupune eliminarea constrangerilor impuse de primul model pastrand capabilitatile de stocare, acces si utilizare a informatiei. Spre deosebire de sistemul relational datele sunt pastrate ca documente, obiecte. Un dezavantaj al lor este faptul ca fiecare tip de sistem foloseste propriul mod de interogare spre deosebire de bazele relationale care au un limbaj comun de extragere a datelor.

Cele mai poulare sunt:

* MongoDB
* Apache Cassandra
* Couchbase
* Redis
* Neo4j

**Diferente notabile**

Structura in care sunt tinute datele – bazele de date relationale necesita o structura cu atribute bine definite pentru pastrarea datelor spre deosebire de NoSQL care ofera o strucura libera in functie de necesitati.

Interogarea – Indiferent de ce sistem de baze de date relationale este folosit el implementeaza pana la un anumit nivel standardul SQL. Bazele de date NoSQL implementeza un mod unic de interogare in functie de tipul lor.

Scalabilitatea – Ambele sisteme sunt usor scalabile prin cresterea performantelor sistemului (scalare pe directie verticala). Insa scalare ape directie orizonala (creearea unui cluster ce contine mai multe masini) este mqai usor de implementat in sistemel NoSQL.

Siguranta – din acest punct de vedere bazele de date SQL ofera siguranta mai are a tranzactiilor efectuate si intgritate a datelor.

Suport si mentenanta – bazele de date relationale avand o istorie mai mare in spate si fiind ofarte populare pot oferi de multe ori un suport mai bun. Daca apar probleme ele pot fi usor rezolvate spre deosebire de probleme care apar in NoSQL ce pot genera solutionari complexe.

Interogari complexe – bazele de date relationale ofera disponibilitate mai amre in ceea ce priveste interogaruile complexe.

**3. MariaDB**

**3.1 Introducere**

**3.2 Replicarea**

**3.2.1 Introducere**

**Replicarea**

Replicarea in MariaDB se realizează la fel a in MySQL. Replicarea este fundația pentru a construi aplicații mari de performanta înalta folosind MariaDB. Replicarea permite configurarea unui server sau a mai multor servere, replici a altui server. Acest lucru nu este valabil doar pentru aplicațiile de înalta performanta dar si pentru rezolvarea altor problem cu mar fi, sharing-ul de date cu mașini aflate la distanta, păstrarea unei copii de rezerva sau păstrarea unei copii pentru testare sau învățare (training).

In continuare for fi analizate aspectele toate aspectele replicării. Pentru început va fi prezentat modul de funcționare, apoi configurarea unui server, proiectarea unor configurații mai avansate de replicare, management-ul si optimizarea serverelor replicate.

**Problema replicării**

Problema replicării se rezolva prin punerea la dispoziție a unui server care sa fie sincronizat cu un altul. Mai multe salve-uri se pot conecta la un singur master, iar un salve se poate comporta la rândul sau ca un master. Master-ii si salve-urile pot fi aranjați in diferite topologii. Se poate replica întregul server, doar anumite baze de date sau se pot alege doar anumite tabele pentru replicare.

MariaDB suporta 2 tipuri de replicare : replicare „statement-based” sau replicare „row-based. „Statement-based” sau replicare logica este disponibila de la MySQL 3.23 si este cea mai folosita in acest moment. Replicare „row-based” este valabila de la MySQL 5.1. Ambele tipuri de replicare salvează schimbările efectuate pe baza de date in așa numitul „binary log” iar acest log este rulat pe slave. Ambele tipuri de replicare sunt asincrone, asta însemnând ca datele copiate pe slave nu au certitudinea ca sunt actualizate la un anumit moment de timp. Nu exista nicio garanție asupra timpului de actualizare a datelor pe slave. Slave-ul poate poate garanta actualizarea datelor după secunde, minute sau chiar ore daca query-urile au dimensiuni mari.

Replicarea MySQL este ca mai compatibila cu versiunile anterioare. Asta înseamnă ca un server cu o versiune mai noua poate fi slave pentru un server cu o versiune mai veche fără probleme. Totuși versiunile mai vechi in principiu nu pot fi slave pentru un server cu o versiune mai noua (nu pot înțelege funcționalități noi apărute si pot fi diferențe intre formatul de fișier folosit pentru replicare). Replicare nu presupune un overhead mare pe master, presupune doar activarea funcției de „binary logging”. Fiecare slave poate adăuga o încărcare suplimentara dar infima datorita operațiilor I/O de rețea.

Replicarea este in general folositoare pentru a scala citiri care pot fi direcționate către un slave, dar nu este o modalitate buna de a scala scrieri, doar in cazul in care este făcută așa cum trebuie. Atașarea multor slave-uri la un master presupune scrierea datelor pe fiecare din mașini. Fiecare sistem este limitata la viteza de scriere pe care o are cea mai slaba mașina.

Replicare este ineficienta când există mai mult de câteva slave-uri deoarece se duplica date fără rost. De exemplu un singur master cu 10 slave-uri va stoca 11 copii a acelorași date si va duplica mare parte din date in 11 cache-uri diferite. Acesta este analog cu „11 way RAID to 1” la nivel de server care specifica faptul ca acest timp de arhitectura nu este un mod de a utiliza eficient aparatura hardware. Totuși acest tip de arhitectura poate fi întâlnit destul de des. In continuare vor fi discutate moduri de a evita aceasta problema.

**Probleme rezolvate de replicare**

Cele mai întâlnite utilizări ale replicării:

* Distribuția datelor (Data distribution)

Replicare in MariaDB si MySQl nu folosește de obicei foarte mult din lungimea de banda a rețelei si poate fi oprita sau pornita la cerere. Prin urmare este recomandată păstrarea unei copii a datelor într-un loc distant din punct de vedere geografic cum ar fi alt centru de date. Slave-ul de la distanta poate lucra chiar cu o conexiune intermitenta (intenționată sau nu). Totuși daca este necesara o replicare fără timp de răspuns mare va fi nevoie de o conexiune stabila.

* Distribuirea încărcării (Load balancing)

Replicare poate ajuta la distribuția query-urilor de citire pe ,ai multe servere, lucru ce funcționează foarte bine pe aplicații cu operații intensive de citire. Se poate efectua „load balancing” doar cu câteva linii de cod. Pe scala mica se poate utiliza abordarea simplista cum ar fi nume de host „hardcoded” sau DNS round-robin (care face ca mai multe IP uri sa pointeze către aceeași gazdă). Exista si abordări mai sofisticate. Soluții standard mai costisitoare sunt produsele pentru distribuție a încărcării pe rețea care funcționează bine împreună cu serverele MySQL. Proiectul server virtual Linux (LVS) se pliază pe aceasta situație.

* Rezerva (Backup)

Replicarea este o tehnica buna pentru a păstra o copie de rezerva a datelor. Însă un slave nu este prin definiție un backup.

* Disponibilitate mare si failover

Replicarea ajuta la evitarea problemei ce presupune prăbușirea aplicației daca exista un singur punct care daca pica poate face aplicația inutilizabila (defectare hardware a mașinii ce tine serverul de baze de date). Un sistem bun de failover este deținerea de slave-uri replicate ce pot lua locul master-ului in caz de defectare.

* Testarea actualizărilor (updates) aduse sistemului de gestiune de baze de date

Este o practica des întâlnită setarea unui slave care are o versiune actualizata de MariaDB/MySQL si folosirea ei pentru a testa buna funcționalitate înainte de a actualiza toate sistemele.

**Cum funcționează replicarea**

Modul in care sunt replicate datele este prezentat in continuare. Văzută de la un nivel înalt replicarea presupune 3 pași:

1. Masterul înregistrează schimbările in fișierul de „binary log”. (Aceste modificări se numesc evenimente ale „binary log”)
2. Slave-ul copiază fișierul „binary log” de la master in log-ul sau „relay log”.
3. Slave-ul reia evenimentele din „relay log”, aplicând modificările pe datele sale.

Aceasta este vederea de ansamblu. Fiecare pas este destul de complex. In continuare va fi prezentata replicarea mai in detaliu.

Prima parte a procesului este cea de „binary logging” care are log pe master. Înainte ca orice tranzacție sa înceapă sa modifice date pe master, master-ul înregistrează schimbările in „binary log”. Tranzacțiile sunt scrise serial chiar daca ele au fost intercalate in momentul execuției lor. După această scriere in log, masterul indică motorului de stocare (storage engine) să salveze (to commit) tranzacțiile.

Cum funcționează replicarea:



Fig. 1

Următorul pas este ca slave-ul sa copieze „binary log”-ul de pe master pe hard-disk-ul sau în așa numitul „relay log”. Pentru început pornește un fir de execuție „worker” numit fitul de execuție I/O al slave-ului. Firul de execute I/O deschide o conexiune obișnuită de la client către master, apoi începe o operație speciala numita „binlog dump” (nu exista o comanda SQL corespondenta). Acest proces citește evenimentele din „binary log”-ul masterului. El nu interoghează master-ul in vederea schimbărilor ci după ce este adus la curent cu masterul intra in „sleep” si așteaptă ca masterul sa semnaleze noi evenimente. Acest fir de execuție scrie evenimentele in „relay log”.

Fitul de execuție slave SQL se ocupa de ultima parte a întregului proces. Acest fir citește si derulează evenimentele din „relay log” astfel încât datele de pe slave dor fi la curent cu cele de pe master. Atât timp cat acest fir de execuție tine pasul cu firul I/O, „relay log”-ul sta de obicei in cache-ul sistemului de operare, prin urmare „relay log”-urile au un overhead foarte mic. Evenimentele pe care firul de execuție SQL le executa pot ajunge opțional in „binary log”-ul slave-ului, lucru folositor pentru scenariile de replicare ce vor fi prezentate ulterior.

In figura sunt prezente 2 fire de execuție pe slave dar conexiunea pe care slave-ul o deschide pe master pornește un nou fir de execuție si pe master.

Acest proces de replicare decuplează procesele de detectare si derulare a evenimentelor pe slave ceea ce oferă posibilitatea ca ele sa fie asincrone. Prin urmare, firul de execuție I/O poate lucra independent de firul de execuție SQL. De asemenea sunt plasate constrângeri pe procesul de replicare, cea mai importanta fiind ca replicarea este serializată pe slave. Asta înseamnă ca actualizări ce ar fi putut rula in paralel pe master nu for fi paralelizate pe slave. Dat fiind acestui fapt pot exista bottle-neck-uri când exista multe date de prelucrat.

**3.2.2 Setarea replicării**

Setarea replicării este un proces relativ simplu, dar exista multe variațiuni ale procesului în funcție de scenariu. Cel mai simplu scenariu este atunci când avem un master si un slave. Procesul presupune următoare etape in mare:

1. Configurarea conturilor de replicare pe fiecare server;
2. Configurarea masterului si a slave-ului;
3. Instruirea serverului sa se conecteze si sa replice din master.

In principiu multe setări standard (default) nu vor trebui modificate, atât timp cat cele doua instanțe sunt proaspăt instalate si conțin aceleași date. In continuare vor presupune ca cele doua instanțe se numesc server1 (cu adresa IP 192.168.0.1) si server2 (cu adresa IP 192.168.0.2). Apoi va fi explicata inițializarea slave-ului .

Crearea conturilor pentru replicare

Exista câteva privilegii ce trebuie setate pentru ca replicarea sa funcționeze. Firul de execuție I/O de pe slave se conectează la master prin TCP/IP. Asta însemnând ca va trebui creat un cont pe master si setarea privilegiilor corespunzătoare pentru ca firul I/O sa se conecteze ca si acel user sa citească „binary log”-ul masterului cum se crea contul pentru user-ul *repl:*

**mysql> GRANT REPLICATION SLAVE, REPLICATION CLIENT ON \*.\***

**-> TO repl@'192.168.0.%' IDENTIFIED BY 'p4ssword';**

Vom crea acest cont user si pe master si pe slave. Replicarea este constrânsă la rețeaua locala curenta deoarece contul de replicare nu este sigur.

Replicarea presupune doar setarea contului doar pe master dar s-a făcut setarea lui si pe client deoarece este mai ușoară monitorizarea ulterioara a schimbărilor de pe client si eventuala comportarea a slave-ului ca master pentru un nou slave.

**Configurarea masterului si slave-ului**

Următorul pas consta in activarea unor opțiuni pe master, pe care îl numim server1. Trebuie activat „binary logging” si specificat un ID pentru server. In fișierul my.cnf trebuie adăugate sau modificate următoarele linii :

**log\_bin = mysql-bin**

**server\_id = 10**

Valorile sunt la alegerea utilizatorului. Ele au fost alese pentru simplitate.

ID-ul de server trebuie sa fie unic. De aceea a fost ales 10 in loc de 1. 1 este valoarea standard (default). Prin urmarea utilizarea numărului 1 poate crea confuzii sau conflicte intre servere care au același ID sau nu îl au setat. Un obicei comun este utilizarea ultimului octet din adresa IP a serverului presupunând ca nu se va schimba si ca este unica.

Daca opțiunea „binary logging” nu era specificata in configurare este necesara repornirea server-ului. Pentru a verifica faptul ca fișierul a fost creat pe master se poate rula comanda **SHOW MASTER STATUS** si se va obține un output similar cu următorul:

**mysql> SHOW MASTER STATUS;**

**+------------------+----------+--------------+------------------+**

**| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB |**

**+------------------+----------+--------------+------------------+**

**| mysql-bin.000001 | 98 | | |**

**+------------------+----------+--------------+------------------+**

**1 row in set (0.00 sec)**

Slave-ul necesita si el o configurare similara cu masterul in fișierul my.cnf si o repornire.

**log\_bin = mysql-bin**

**server\_id = 2**

**relay\_log = mysql-relay-bin**

**log\_slave\_updates = 1**

**read\_only = 1**

Câteva din aceste opțiuni nu sunt strict necesare, sunt doar enunțate cele standard. Pe salve doar server ID-ul este necesar dar a fost activat si „binary logging” cu un nume explicit. In principiu vrem aceeași configurare pe toate mașinile astfel încât promovarea de al slave la master sa fie facila. La fel cum au fost create aceleași conturi si pe master si pe slave, vrem si aceleași setări.

Exista 2 parametri opționali: relay\_log – care specifica locația si numele „relay log”-ului si log\_slave\_updates – care specifica slave-ului sa adauge evenimentele primite si in „binary log”ul propriu. Ultima opțiune cauzează lucru in plus pentru slave dar in continuare va fi explicata de ce adăugarea ei este un practice bun.

**Pornirea slave-ului**Următorul pas spune slave-ului cum sa se conecteze la master si sa înceapă derularea „binary logg”-urilor primite. Pentru acest scop nu trebuie folosit fișierul my.cnf ci comanda CHANGE MASTER TO. Aceasta comanda înlocuiește setările corespondente din my.cnf in totalitate. I n viitor permite si schimbarea masterului fără oprirea serverului. Următorul cod trebuie rulat pe slave:

**mysql> CHANGE MASTER TO MASTER\_HOST='server1',**

**-> MASTER\_USER='repl',**

**-> MASTER\_PASSWORD='p4ssword',**

**-> MASTER\_LOG\_FILE='mysql-bin.000001',**

**-> MASTER\_LOG\_POS=0;**

Câmpul MASTER\_LOG\_POS este setat cu valoarea 0 deoarece indica începutul log-ului. După ce se rulează aceasta comanda se poate verifica setarea cu comanda SHOW SLAVE STATUS ce produce următorul output:

**mysql> SHOW SLAVE STATUS\G**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Slave\_IO\_State:**

**Master\_Host: server1**

**Master\_User: repl**

**Master\_Port: 3306**

**Connect\_Retry: 60**

**Master\_Log\_File: mysql-bin.000001**

**Read\_Master\_Log\_Pos: 4**

**Relay\_Log\_File: mysql-relay-bin.000001**

**Relay\_Log\_Pos: 4**

**Relay\_Master\_Log\_File: mysql-bin.000001**

**Slave\_IO\_Running: No**

**Slave\_SQL\_Running: No**

**...omitted...**

**Seconds\_Behind\_Master: NULL**

Coloanele Slave\_IO\_State, Slave\_IO\_Running, si Slave\_SQL\_Running arata ca procesele de pe slave nu rulează. Se observa ca poziția log-ului este 4. Prin urmare 0 setata anterior nu este o poziție ci indica doar începutul fișierului de log. Primul eveniment este de fapt la poziția 4.

Pentru a rula aplicația se executa comanda :

**mysql> START SLAVE;**

Aceasta comanda nu ar trebui sa producă erori sau output. Se poate verifica cu comanda SHOW SLAVE STATUS:

**mysql> SHOW SLAVE STATUS\G**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Slave\_IO\_State: Waiting for master to send event**

**Master\_Host: server1**

**Master\_User: repl**

**Master\_Port: 3306**

**Connect\_Retry: 60**

**Master\_Log\_File: mysql-bin.000001**

**Read\_Master\_Log\_Pos: 164**

**Relay\_Log\_File: mysql-relay-bin.000001**

**Relay\_Log\_Pos: 164**

**Relay\_Master\_Log\_File: mysql-bin.000001**

**Slave\_IO\_Running: Yes**

**Slave\_SQL\_Running: Yes**

**...omitted...**

**Seconds\_Behind\_Master: 0**

Se observa ca ambele fire de execuție I/O si SQL rulează si câmpul Seconds\_Behind\_

Master nu mai este NULL. Firul de execuție I/O așteaptă evenimente de la master ceea ce înseamnă ca a preluat toate fișierele „binary log” de la master. Pozițiile de log s-au incrementat ceea ce înseamnă ca exista evenimente ce au fost preluate si executate. Modificare masterului va face schimbări pe slave.

Se pot vizualiza si firele de execuție pentru replicare in lista de procese atât pe master cat si pe slave. Pe master va fi o conexiune creata de firul I/O al slave-ului:

**mysql> SHOW PROCESSLIST\G**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Id: 55**

**User: repl**

**Host: slave1.webcluster\_1:54813**

**db: NULL**

**Command: Binlog Dump**

**Time: 610237**

**State: Has sent all binlog to slave; waiting for binlog to be updated**

**Info: NULL**

Pe slave se pot observa 2 fire de execuție, cel I/O si cel SQL:

**mysql> SHOW PROCESSLIST\G**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Id: 1**

**User: system user**

**Host:**

**db: NULL**

**Command: Connect**

**Time: 611116**

**State: Waiting for master to send event**

**Info: NULL**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Id: 2**

**User: system user**

**Host:**

**db: NULL**

**Command: Connect**

**Time: 33**

**State: Has read all relay log; waiting for the slave I/O thread to update it**

**Info: NULL**

**Inițializarea unui alt slave de pe alt server**

In configuratia anterioara am presupus ca si masterul si slave-ul aveau aceleași date inițial fiind nou instalate. Acesta este un caz ideal, de obicei un master a rulat mult timp si se dorește sincronizarea unui slave care nu are toate datele masterului.

Exista mai multe moduri de a inițializa, clona un server pe slave. Printre acestea se numără copierea datelor de pe master, clonarea unui slave cu un alt slave, pornirea unui slave dintr-un backup. Pentru a sincroniza un slave cu masterul sunt necesare:

* O clona, „snapshot” al masterului la un anumit moment;
* Fișierul de log curent al masterului si offset-ul din log la care a fost făcută acea clona. Aceste valori se numesc coordonate ale fișierului de log, deoarece identifica poziția in „binary log”. Coordonatele pot fi găsite cu comanda SHOW MASTER STATUS.
* „binary log”-ul masterului din acel moment pana in prezent.

Câteva moduri de a clona un slave dintr-un alt slave:

* Copie „la rece”

Aceasta presupune oprirea masterului si copierea fișierelor pe slave. Apoi masterul poate fi pornit din nou ceea ce începe un nou „binary log”. Folosind comanda CHANGE MASTER TO se va porni un slave de la începutul acelui „binary log”. Dezavantajul evident este necesitatea de a opri masterul pana când se va termina copierea.

* Copie „la cald”

Daca se folosesc numai tabele MyISAM, se poate folosi mysqlhotcopy pentru a copia fișiere in timp ce serverul rulează.

* Folosind mysqldump

Daca se folosesc numai tabele InnoDB se poate folosi următoarea comanda pentru a face „dump” tuturor datelor de pe master, încărcarea lor pe slave si schimbarea coordonatelor slave-ului la poziția corespunzătoare din „binary log”.

**$ mysqldump --single-transaction --all-databases --master-data=1**

**--host=server1 | mysql --host=server2**

Opțiunea –single-transaction impune operației sa citească datele așa cum erau la începutul tranzacției. Se poate folosi si –lock-all-tables pentru avea un „dump” consistent al tuturor tabelelor.

* Folosind LVM snapshot sau backup

Atât timp cat se știu coordonatele corespunzătoare din „binary log” , se poate folosi un LVM snapshot sau backup al masterului pentru a inițializa slave-ul (ex: InnoDB Hot Backup, backup-ul presupune existenta tuturor „binary log”-urilor de la data de când a fost făcut backup-ul).

Backup-ul va fi rulat pe slave apoi cu comanda CHANEG MASTER TO se pornește slave-ul.

* Folosind un alt slave

Orice metoda menționată mai sus poate fi folosita pentru a clona un slave dintr-un alt slave. Daca se folosește comanda mysqldump, opțiunea –master-data nu va funcționa.

De asemenea, in locul comenzii SHOW MASTER STATUS, se va folosi SHOW SLAVE STATUS.

Cel mai mare dezavantaj când se clonează un slave dintr-un alt slave este acela ca daca slave-ul nu este sincronizat cu masterul, se vor clona date invalide.

**3.2.3 Implementarea replicarii**

In continuare va fi explicat modul in care funcționează replicarea mai în amănunt. Vor fi discutate punctele forte si slăbiciunile si ulterior vor fi analizate configurații mai complexe de replicare.

**Replicarea bazata pe statement (statement-based)**

Numita si replicare logica, ea este primul mod de replicare folosit in MySQL. Acest mod nu este unul des întâlnit in lumea bazelor de date. Replicarea „statement based” funcționează înregistrând query-ul care a modificat datele de pe master. Când slave-ul citește evenimentele din „relay log” si le executa va face același lucru pe care l-a făcut masterul executând query-uirle SQL. Acest tip de replicare are atât avantaje cat si dezavantaje.

Avantajul imediat observabil este acela ca este ușor de implementat. Simpla înregistrare si reexecutare a query-urilor ce au modificat datele in teorie va tine slave-ul sincronizat cu masterul. Un al beneficiul al replicării „statement based” este acela ca evenimentele in „binary log” sunt compacte. Deci acest tip de replicare teoretic nu va face un trafic mare pe rețea – un query care actualizează giga octeți de date poate avea doar câțiva octeți in „binary log”.

Totuși acest tip de replicare nu este la fel de simplu in practica pentru ca multe schimbări de pe master pot depinde de alți factori diferiți de query-ul efectiv. De exemplu instrucțiunile se vor executa la o diferență de timp cel puțin mica sau posibil mai mare intre slave si master. Ca rezultat in „binary log” sunt ținute si informații (meta data) cum ar fi un timestamp. Chiar si așa unele, instrucțiuni nu se pot replica corect cum ar fi funcția CURRENT\_USER(). Procedurile stocate si trigger-ele sunt si ele problematice in replicarea „statement-based”.

O alta problema la acest tip de replicare este acela ca modificările trebuie sa fie sterilizabile. Asta presupune un mare volum de lucru la configurare, funcții extra ale server-ului. InnoDB folosește replicarea „statement based”. Nu toate motoarele de stocare folosesc acest tip de replicare deși pana la MySQL 5.1 acest lucru era posibil.

**Replicarea pe baza de rând (row-based)**

Replicarea „row-based” înregistrează in „binary log” exact ce modificări au fost aduse datelor, similar cum ce oferă alte produse de tip baze de date de pe piață. Acest tip de replicare are si ea avantajele si dezavantajele sale. Cel mai mare avantaj este acela ca se va replica fiecare schimbare corect si unele instrucțiuni pot fi replicate mult mai eficient. Dezavantajul primar este acela ca „binary log”-ul va deveni considerabil mai mare si exista o vizibilitate mai proasta asupra căror instrucțiuni au modificat datele. Prin urmare nu se va mai putea folosi *mysqlbinlog* pentru audit.

Replicarea „row-based” din păcate nu este compatibila backward (compatibila cu versiunile mai vechi). *mysqlbinlog,* care poate citi fișierele „binary log” aducându-le într-o formă descifrabilă pentru oameni, va iesi din program afișând eroare daca se încearcă citirea unui „binary log” dintr-o distribuție mai veche a software-ului.

Unele schimbări pot fi replicate mai eficient deoarece salve-ul nu va trebui sa reexecute query-urile ce au făcut schimbări pe master. Reexecutarea de query-uri poate fi costisitoare. Ca exemplu poate fi luat acest query ce sumarizează date dintr-un tabel mare in unul mai mic:

**mysql> INSERT INTO summary\_table(col1, col2, sum\_col3)**

**-> SELECT col1, col2, sum(col3)**

**-> FROM enormous\_table**

**-> GROUP BY col1, col2;**

Sa presupunem ca exista trei combinații unice dintre coloanele col1 si col2 din tabelul enormous\_table. Acest query va scana multe rânduri din tabelul sursa si vor rezulta doar 3 rânduri in tabelul destinație. Replicarea acestui eveniment ca „statement based” va obliga slave-ul sa fac din nou toata procesarea pe care a făcut-o master-ul doar pentru a genera câteva rânduri însă replicarea „row based” va simplifica semnificativ procesarea slave-ului. In acest caz replicarea „row based” este mult mai eficienta.

Pe de alta parte următorul eveniment este mult mai simplu de replicat prin „statement based”:

**mysql> UPDATE enormous\_table SET col1 = 0;**

Replicarea row-based va fi foarte costisitoare in acest caz deoarece fiecare schimbare va fi scrisa in „binary log”, acesta devenind semnificativ mai mare. Prin urmare încărcarea master-ului va fi mai mare si la înregistrare si la replicare.

Deoarece nici unul dintre moduri nu este perfect pentru fiecare situație, ambele moduri sunt folosite alternativ dinamic. Ca si default este folosita replicarea „statement based” dar când sunt detectate evenimente ce nu pot fi replicate corect se trece la replicare „row based”. Modul poate fi controlat prin schimbarea variabilei binlog\_format.

In teorie replicare row-based rezolva câteva din problemele menționate anterior, dar in producție încă se folosește considerabil replicare „statement-based”. Prin urmare este prea devreme sa concluzionăm asupra replicării „row-based”.

**Fișiere folosite la replicare**

In afara fișierelor de „binary log” si „relay log” exista si câteva alte fișiere folosite de replicare. Unde sunt plasate ele depinde de configurarea instanței. Versiuni diferite pot plasa aceste fișiere in directoare diferite. Pot fi găsite in directorul destinat datelor sau in cel care conține fișierul .pid al serverului (var/run/mysqld – sistemele Unix). Aceste fișiere sunt:

* mysql-bin.index

Un server care are opțiunea de „binary logging” activata va avea de asemenea un fișier numit la fel ca acela numai ca extensia va fi .index. Acest fișier menține evidenta fișierelor de „binary logging” care exista pe disc. Nu este un index in sensul index-ului din tabele; fiecare linie din acest fișier conține numele unui fișier de „binary log”.

Poate părea ca acest fișier este redundant si poate fi șters, însă serverul nu se va uita pe disc ci in acest fișier pentru a cunoaște existenta fișierelor de „binary log”.

* mysql-relay-bin.index

Acest fișier are același rol pe care îl are cel de mai sus numai ca pentru „relay log”.

* master.info

Acest fișier conține informație folosita de către slave pentru a se conecta la master. Formatul este text simplu (o valoare pe linie) si variază in funcție de versiunea serverului. Nu trebuie șters deoarece un slave nu va mai ști cum sa se conecteze la master după restart. Fișierul conține parola user-ului de replicare ca si text simplu deci accesul la el ar trebui sa fie restricționat.

* relay-log.info

Acest fișier conține coordonatele slave-ului pentru „binary log” si „relay log” (poziția pe master a slave-ului). Nu trebuie șters deoarece slave-ul nu va mai ști de unde sa replice după restart si este foarte posibil sa încerce replicarea unor instrucțiuni ce deja au fost executate.

**Trimiterea evenimentelor de replicare către slave-uri**

Opțiunea log\_slave\_updates permite folosirea unui slave ca master pentru alt slave. Se impune server-ului sa scrie evenimentele pe care le firul de execuție SQL le executa in propriul „binary log” care va fi preluat si executat de serverele slave ale sale. Următoarea figura ilustrează acest mod de lucru.

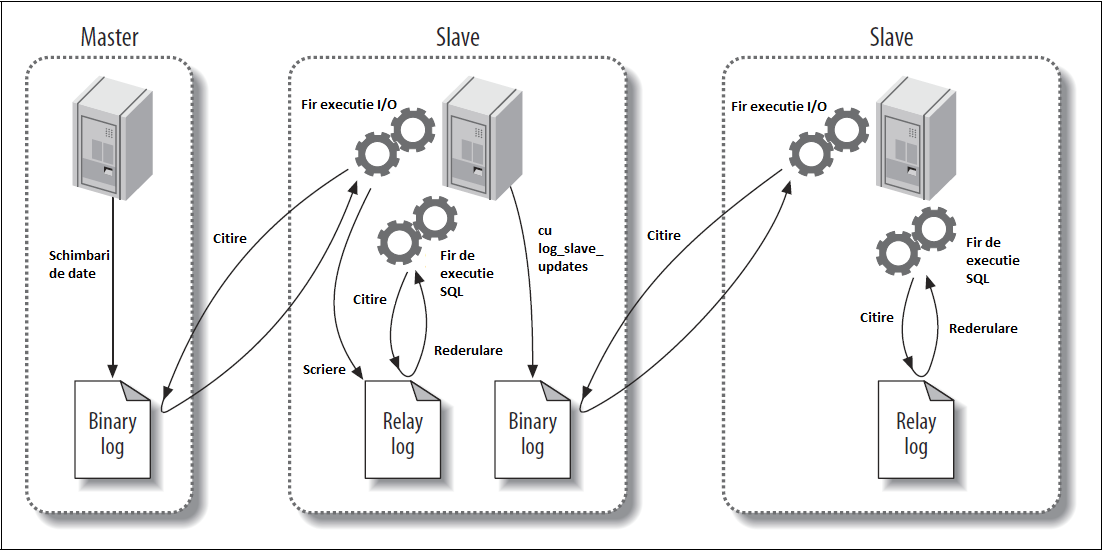


Fig. 2

In acest scenariu o schimbare pe master cauzează un eveniment scris in „binary log”. Primul slave preia evenimentul si îl executa. In acest punct, de obicei ciclul de viată al evenimentului ar fi luat sfârșit însă deoarece este activata opțiunea log\_slave\_updates slave-ul va scrie modificările in „binary log”. Acum cel de-al doilea slave va putea prelua modificările in propriul sau „relay log” si sa le execute. Acest lucru presupune ca schimbările de pe serverul master se pot propaga către mașini ce nu sunt direct conectate la master. Opțiunea log\_slave\_updates de preferabil trebuie activata deoarece permite conectarea unui slave fără repornirea serverului.

Când primul salve scrie in „binary log” un eveniment apărut pe master, acel eveniment cu siguranță va avea o poziție diferita fata de cel de pe master – un alt fișier de log sau altă poziție numerica. Asta înseamnă ca nu se poate presupune ca toate serverele care sunt in același punct al replicării vor avea aceleași coordonate de log.

Daca din greșeală nu a fost asignat un ID de server diferit pentru fiecare mașină, acest tip de configurație poate cauza erori subtile, poate chiar cauza oprirea replicării. Una din cele mai întâlnite întrebări legate de replicate este de ce trebuie specificat acest ID. Când firul de execuție SQL citește din „relay log” va ignora evenimentele care au un server ID identic cu al sau. Acest lucru împiedica apariția buclelor infinite in replicare. Acestea pot apărea la replicarea master-master.

**Filtre de replicare**

Filtrele de replicare pot constrânge replicarea doar a anumitor părți ale datelor de pe master. Există doua tipuri de filtre de replicare: acelea care filtrează evenimentele din „binary log”-ul masterului si acelea care filtrează evenimentele din „relay log”-ul slave-ului. Figura următoare ilustrează aceste doua tipuri:

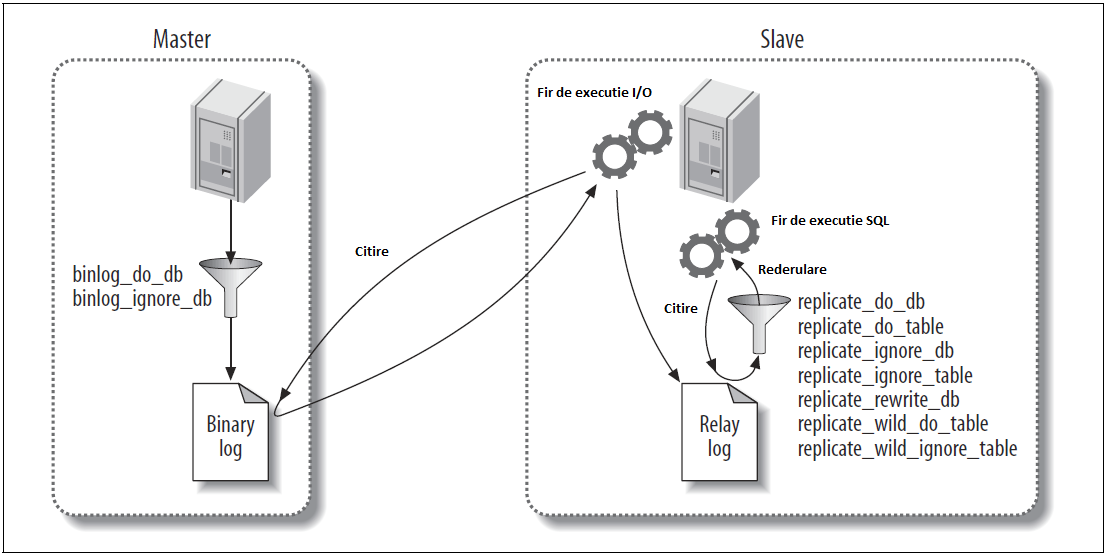


Fig. 3

Opțiunile care controlează filtrarea „bianry log”-ului sunt *binlog\_do\_db* si *binary\_ignore\_db*. De obicei nu trebuie activate după cum va fi explicat in continuare.

Pe slave opțiunile *replicate\_\** filtrează evenimente pe care firul de execuție SQL le citește din „relay log”. Se pot replica sau ignora una sau mai multe baze de date, rescrierea unei baze de date in alta baza de date si replicarea sau ignorarea unor tabele pe identificarea lor cu instrucțiunea LIKE.

Cel mai important lucru de reținut in legătură cu aceste opțiuni este faptul ca *\*\_do\_db* si *\*\_ignore\_db*, atât pe master sau slave pot sa nu se manifeste așa cum ar fi de așteptat. Ar putea filtra după numele bazei de date dar de fapt filtrează după baza de date setata curent „default”. Daca se execută următoarele instrucțiuni:

**mysql> USE test;**

**mysql> DELETE FROM sakila.film;**

parametrii *\*\_do\_db* si *\*\_ignore\_db* for filtra instrucțiunile DELETE pe baza de date *test* nu pe *sakila.* De obicei nu se vrea acest lucru deoarece instrucțiuni nedorite pot fi replicate sau ignorate.

Parametrii *\*\_do\_db* si *\*\_ignore\_db* pot avea utilizări dar acestea sunt limitate si rare, folosirea lor trebuie să fie făcută cu grija. Folosirea lor poate duce foarte ușor la desincronizare.

Oprirea instrucțiunilor GRANT sau REVOKE de la replicare către salve-uri este o practica utilizata frecvent in filtre. Problema frecventa este ca administratorul folosește GRANT pentru a da privilegii unui user pe master si apoi observa ca acestea au fost propagate si pe slave, unde acel user nu ar trebui sa aibă acces. Următoarele opțiuni pot preveni acest lucru:

**replicate\_ignore\_table1`=mysql.columns\_priv**

**replicate\_ignore\_table=mysql.db**

**replicate\_ignore\_table=mysql.host**

**replicate\_ignore\_table=mysql.procs\_priv**

**replicate\_ignore\_table=mysql.tables\_priv**

**replicate\_ignore\_table=mysql.user**

Se poate aplica mai simplu un filtru pentru toate bazele de date care încep cu *mysql* cu o regula de tipul :

**replicate\_wild\_ignore\_table=mysql.%**

In acest fel cu siguranță instrucțiunile de tip GRANT nu vor fi replicate, dar anumite evenimente sau rutine pot fi excluse din replicare de asemenea. Prin urmare acesta este motivul pentru care utilizarea filtrelor trebuie să fie făcută cu grija. O ideea mai buna poate fi prevenirea anumitor instrucțiuni specifice de a fi executate setând SET SQL\_LOG\_BIN=0, dar si acest mot are deficientele sale.

Ca si concluzie folosirea filtrelor trebuie făcută cu atenție deoarece se replicarea de tipul „statement based” poate fi întreruptă foarte ușor. Replicare „row based” poate rezolva anumite probleme dar nu este încă dovedit 100% acest lucru.

**3.2.4 Topologii de replicare**

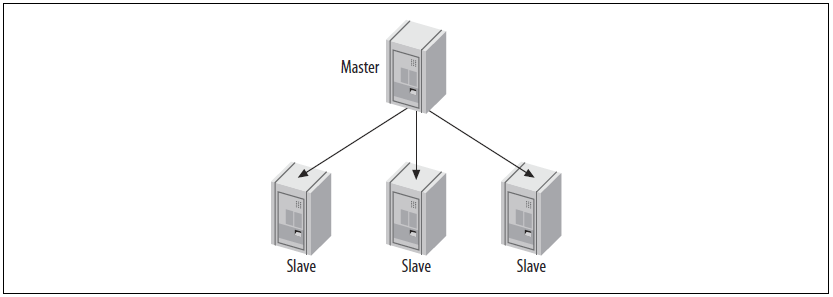
Replicare poate fi setata pentru aproape orice tip posibil de configurare master-slave cu limitarea ca o instanță slave poate avea un singur master. Prin urmare diferite topologii complexe pot fi dezvoltate, iar cele simple sunt destul de flexibile. O anumita topologie poate avea diferite utilizări.

După cum a fost prezentat anterior setarea unu8i singur master cu in singur slave a fost acoperita. In continuare vor fi discutate diferite alte topologii cu avantajele si dezavantajele lor. De reținut sunt următoarele reguli:

* O instanță salve poate avea un singur master
* Fiecare server trebuie sa aibă un server ID unic
* Un master poate avea mai mule salve-uri (un salve poate avea mai mulți copii la rândul sau)
* Un slave poate propaga schimbările de pe master si poate fi la rândul sau master pentru alte slave-uri daca este activata opțiunea log\_slave\_updates.

**Un master cu mai multe slave-uri**

Aceasta topologie este foarte simpla deoarece implica atașarea uni nou slave la topologia clasica master-slave. Cele 2 sau mai multe salve-uri nu vor comunica intre ele ci doar cu masterul ca in următoarea figura:



Aceasta topologie este folositoare si avantajoasa atunci când se executa multe citiri. Aceste citiri pot fi distribuite pe mai multe servere atunci când încărcarea pe master este prea mare sau încărcarea rețelei de la master către salve-uri este foarte mare. Pot fi setate mai multe salve-uri o data sau adăugarea lor pe parcurs in funcție de necesar, folosind aceeași pași prezentați anterior la setarea unui slave.

Deși este o topologie foarte simpla este destul de flexibila pentru a satisface diferite probleme cum ar fi :

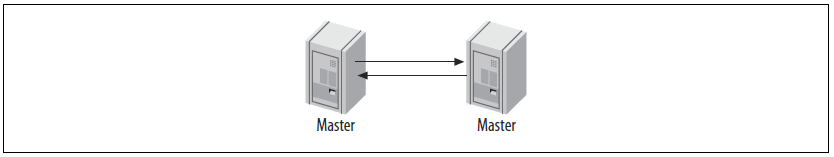
* Folosirea a diferite salve-uri pentru diferite roluri (de exemplu folosirea a diferite motoare de stocare pe slave-uri)
* Setarea uni server de replicare fără alt tip de trafic in afara de replicare
* Punerea uni slave într-un centru de date pentru recuperarea in caz de dezastru
* Întârzierea replicării pe anumite slave-uri pentru recuperare in caz de dezastru
* Folosirea unuia dintre salve-uri pentru backup, pregătire personal sau ca server de dezvoltare (development).

Unul dintre motivele pentru care acest tip de replicare este foarte răspândit este acela ca evita multe complexități ce apar in celelalte tipuri de replicare. De exemplu este ușor de comparat un salve cu altul ca si poziție a „binary log”-ului deoarece ea trebuie sa fie aceeași mereu după replicare. Daca toate slave-urile vor fi oprite la același moment de tip logic in replicare ele vor citii din nou de la același punct. Acest lucru rezolva probleme administrative cum ar fi promovarea uni slave ca master.

Acest lucru este posibil intre slave-uri de același nivel (frați). Este mai complicata compararea intre slave-uri care nu sunt la același nivel fata de master. Multe din modurile următoare de replicare cum ar fi arborele sau distribuția de master fac mult mai grea depistarea secvenței de unde se face replicarea.

**Master–master in modul activ-activ**

Replicarea master-master (numita si dual master sau replicare bidirecțională) implica doua servere ambele master si slave unul pentru celălalt. In alta ordine de idei 2 perechi master-slave ca in figura următoarea:



Replicare master-master in modul activ-activ are utilizări speciale. Una din utilizări poate fi pentru birouri separate geografic unde fiecare birou are nevoie de copia sa de date locala.

Cea mai mare problema cu acest tip de configurarea este rezolvarea de conflicte. Lista de probleme poate fi foarte lunga. Prolemele apar de obicei când doua query (unul pe prima mașină, celălalt pe a doua mașină) schimba același rând simultan pe ambele servere sau inserează un rând cu AUTO\_INCREMENT in același timp.

**Replicare multimaster nu este suportata**

Replicare multimaster reprezintă posibilitate ca un salve sa replice de pe mai multe mastere. Deși alte tipuri de sisteme de baze de date suportă acest lucru topologia din figura următoare nu este posibila:

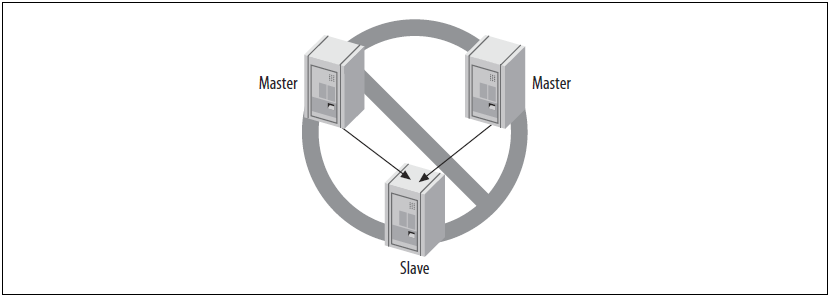


Fig. 5

Totuși acest tip de replicare poate fi emulat după cum va fi explicat mai departe in acest capitol.

Din păcate exista o confuzie in ceea ce privește terminologia. Multe persoane folosesc termenul mulitmaster pentru a descrie alte tipuri de topologie care conțin mai mult de doua mastere.

Funcționalități ce rezolva problemele acestui tip de replicare au fost adăugate in MySQL 5.0 cum ar fi opțiunile *auto\_increment\_increment* sau *auto\_increment\_offset*. Aceste opțiuni pot autogeneză valori care sa nu aibă conflicte la INSERT. Totuși operația de scriere pe ambele mastere rămâne periculoasa. Actualizări ce au loc in ordine diferita pe cele 2 mașini pot duce la corupere de date fără nici un fel de avertizare. Ca exemplu putem lua un singur tabel cu o singura coloana cu valoarea 1. Aceste două query-uri se vor executa simultan:

* Pe primul master

**mysql> UPDATE tbl SET col=col + 1;**

* Pe al doilea master

**mysql> UPDATE tbl SET col=col \* 2;**

Rezultat va fi acela ca pe un server vom avea valoarea 4 pe altul 3 si nicio eroare data de replicare.

Desincronizarea datelor este doar începutul. Replicarea normala se poate opri cu eroarea dar aplicațiile nu se opresc din a scrie pe ambele servere. Rezolvarea acestei probleme devine foarte complicata deoarece ambele servere au modificări ce ar trebui propagate pe celălalt server.

Setarea master-master in modul active-active devine foarte dificila dar nu imposibila folosind date partiționate corect si privilegii setate corect. De obicei exista o modalitate mai buna decât aceasta pentru a răspunde nevoilor.

In general a lăsa libertatea de a scrie pe ambele servere poate aduce mai multe probleme decât beneficii. Prin urmare configurarea active-pasive ce va fi prezentata ulterior este mai indicata.

**Master-master in modul activ-pasiv**

Acest mod evita problemele ce au apărut la modul discutat anterior, este chiar un mod foarte indicat si puternic de a proiecta un sistem tolerant la defecte si cu disponibilitate mare. Diferența este aceea ca unul din servere este pasiv in modul „read-only” ca in figura următoare:

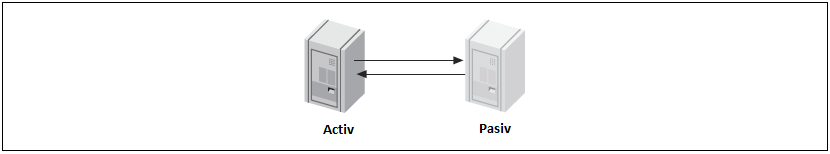


Fig.6

Acest mod permite interschimbarea intre activ si pasiv foarte ușor deoarece configurațiile serverelor sunt simetrice. Asta face sistemul tolerant la defecte. De asemenea mentenanța, optimizarea tabelelor si actualizarea sistemului de operare sau a aplicațiilor, a hardware-ului devine ușor de realizat.

De exemplu folosirea instrucțiunii ALTER TABLE presupune închiderea tabelei (lock), blocarea scrierii si citirii către tabelă. Acest lucru poate fi de durata si poate întrerupe serviciul. Prin urmare configurație master-master permite oprirea firelor de execuție slave pe serverul activ si nu va procesa schimbări de pe serverul pasiv. Se vor altera datele de pe serverul pasiv apoi se vor schimba rolurile si se va reporni procesul slave pe serverul anterior activ. Acest server va citi „relay log”-ul si va executa aceeași instrucțiune de ALTER TABLE. Acest lucru poate fi de durata dar nu contează pentru ca serverul nu oferă disponibilitate pentru query-uri.

Topologia activ-pasiv master-master poate fi ușor de folosit instalând utilitarul MySQL Master-Master Replication manager Tool (<http://code.google.com/p/mysql-master-master/>). Automatizează diferite operațiuni cum ar fi recuperarea si resincronizarea, setarea unor slave-uri noi si așa mai departe.

Pași pentru a configura o topologie master-master simetrica. Setul de pași trebuie realizat pe ambele mașini pentru a deveni simetrice.

1. Activarea „binary logging”, alegerea unui server ID unic, adăugarea conturilor de replicare.
2. Activarea *log\_slave\_updates*. Acest lucru este vital pentru toleranta la defecte.
3. Opțional serverul pasiv poate fi configurat ca „read-only” pentru a preveni schimbări ce ar putea genera conflicte pe serverul activ.
4. Ambele mașini trebuie sa conțină exact aceleași date.
5. Pornirea fiecărei instanțe de MariaDB.
6. Configurarea fiecărei mașini ca slave pentru cealaltă, începând de la „binary log”-ul recent creat.

In continuare vom urmări ce se întâmplă când o modificare este făcută pe serverul activ. Schimbarea este scrisa in „binary log” si ajunge prin replicare in „relay log”-ul slave-ului. Acest server pasiv executa instrucțiunile si scrie modificările in propriul „binary log” deoarece am activat *log\_slave\_updates*. Serverul activ primește aceleași schimbări prin replicare in propriul „relay log” dar le ignora deoarece server ID-ul lor este propriul ID. Interschimbarea activ-pasive va fi prezentata mai târziu.

Setarea unei topologii activ-pasiv master-master presupune crearea unei copii de rezerva dar aceasta poate fi folosita pentru a spori performanța. Poate fi folosita pentru a citi din baza de date, backup, mentenanță si așa mai departe. Ea nu poate fi folosita pentru a obține performanta mai mare la scriere cum se obține cu un singur server. In următoarele topologii vom reveni la acest tip deoarece este o modalitate foarte comuna in replicare.

**Master-master cu slave-uri**

O configurație asemănătoare poate presupune adăugarea unui slave la fiecare master ca in figura următoare:

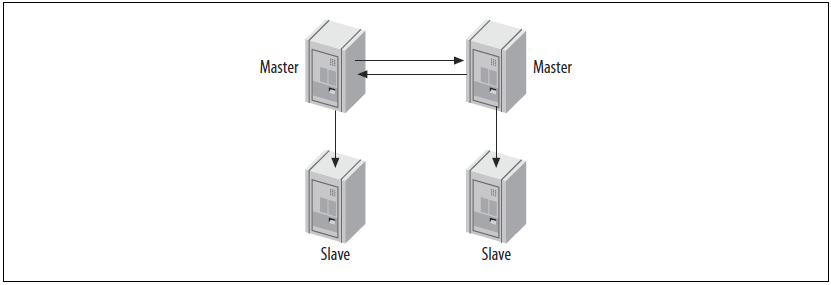


Fig. 7

Avantajul este extra redundanta. Într-o rețea distribuita geografic elimina posibilitatea de defectare într-un anumit punct. De asemenea se poate distribui citirea intensa pe slave-uri.

Daca aceasta topologie master-master este folosita local, aceasta configurație este încă folositoare. Promovarea unui slave ca master este posibila in cazul defectării, deși este o operație mai complexa. Această operație este la fel de costisitoare ca schimbarea masterului unu slave. Aceasta complexitate este de luat in considerare.

**Topologia inel**

Topologia dual-master este de fapt un caz special al topologiei de replicare tip inel. Un inel are 3 sau mai multe mastere. Fiecare server este slave pentru cel de dinaintea sa in inel si master pentru serverul de după el. Aceasta topologie se mai numește si replicare circulara.

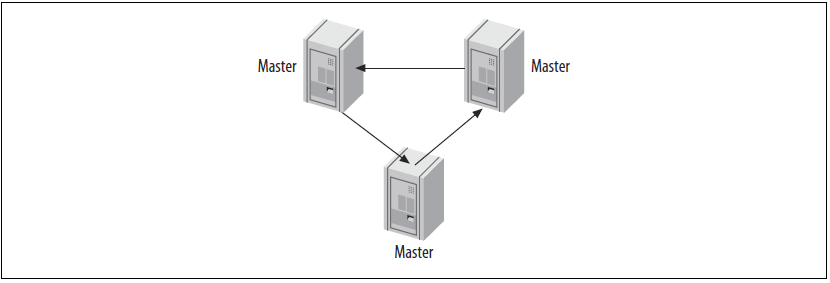
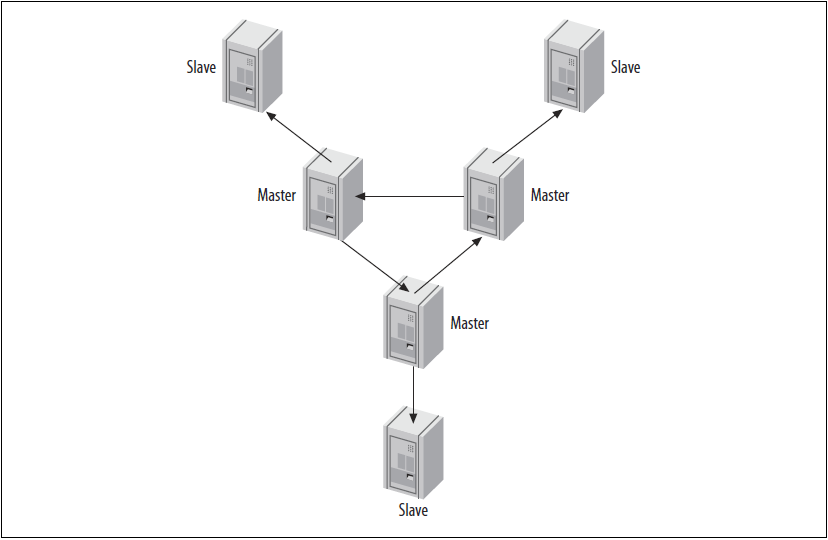


Fig. 8

Topologia inel nu are câteva beneficii ale topologiei master-master cum ar fi simetria configurației si toleranta la defectare. Rețeaua depinde complet de fiecare nod din inel ceea ce creste șansa de defectare a întregului sistem. Daca se scoate un nod din inel , evenimentele replicate din el pot intra într-o bucla infinita. Vor circula la infinit prin topologie pentru ca singurul server care poate filtra acele evenimente este cel care le-a inițiat. In general inelele sunt de evitat.

Se pot evita anumite probleme ale acestei topologii adăugând salve-uri la fiecare nod oferind redundanta ca in figura următoare:

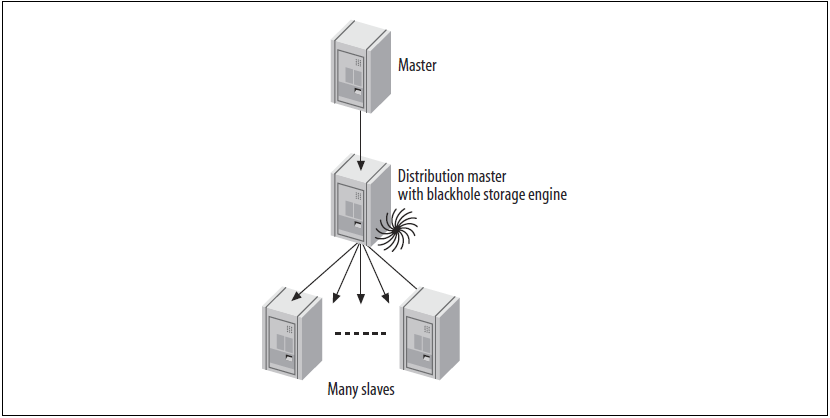


**Master, distribuția masterului si slave-uri**

Am menționat anterior ca salve-urile pot produce o încărcare pe master daca sunt destul de multe. Fiecare slave creează un nou fir de execuție pe master, care executa comanda speciala *binlog dump.* Aceasta comanda citește datele din „binary log” si le trimite către slave. Acest lucru se repeta pentru fiecare slave.

Dacă exista multe salve-uri configurate si un „binary log” cu o dimensiune considerabila cum ar fi un fișier imens LOAD DATA INFILE, încărcarea pe master poate creste semnificativ. Masterul poate chiar sa rămână fără memorie si sa se defecteze pentru ca toate slave-urile cer același eveniment cu o dimensiune foarte mare in același timp. Pe de alta parte daca fiecare slave cere un alt eveniment care nu exista in memoria cache a sistemului de fișiere vor apărea căutări pe disc care pot de asemenea sa intervină in performanta masterului.

Din acest motiv daca sunt necesare multe slave-uri este o practica buna înlăturarea încărcării de pe master si folosirea unui master de distribuție. Un master de distribuție este un master al cărui singur scop este sa citească si sa distribuie „binary log”-urile de pe master. Multe slave-uri se pot conecta la masterul de distribuție care preia încărcarea de pe masterul original. Pentru a bloca executarea evenimentelor pe acest server va trebui modificat motorul de stocare la Blackhole storage engine ca in figura următoare:



Este destul de greu de spus cate slave-uri poate susține un master pan atunci când va fi nevoie de un master de distribuție. Ca si regula generala daca masterul rulează la o capacitate aproape de maxim, nu este indicata conectarea a mai mult de 10 slave-uri la el. Daca scrierile sunt puține sau se replica doar o parte din tabele masterul foarte probabil poate susține mai mult de 10 slave-uri. In plus poate fi folosit mai mult de un master de distribuție. Pot fi folosite mai multe daca exista un număr mare de slave-uri sau se poate folosi distribuția piramida a masterelor.

Masterul de distribuție poate fi folosit si in alte scopuri cum ar fi aplicarea filtrelor si rescrierea regulilor evenimentelor in „binary log”. Acest lucru este mai eficient decât repetarea filtrării si a celorlalte operații pe fiecare slave.

Daca se folosesc tabele Blackhole pe masterul de distribuție el poate susține mai multe slave-uri decât de obicei. Masterul de distribuție va executa query-urile însă ele vor fi deloc costisitoare pentru ca motorul de stocare Blackhole nu salvează date in tabele.

O problema des întâlnită este cum se asigura ca masterul de distribuție folosește motorul de stocare Blackhole. Soluția este setarea lui pe întreg mediu:

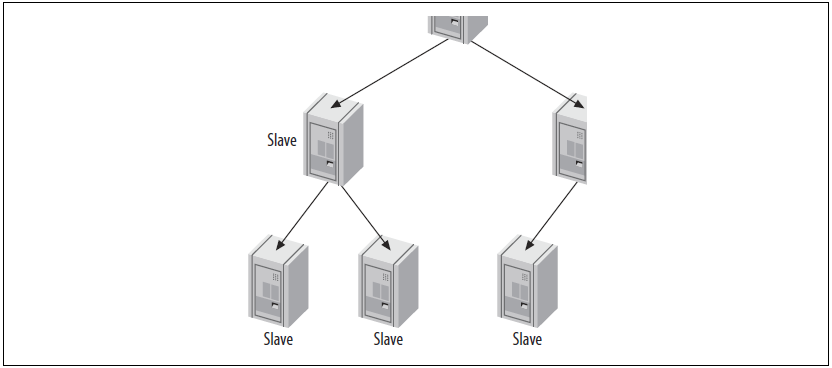
**Storage\_engine = blackhole**

Acest lucru va afecta doar instrucțiunile CREATE TABLE care nu specifica direct un motor de stocare. Daca acesta problema nu poate fi controlata mediul poate deveni fragil. Motorul InnoDB poate fi dezactivat însă tabelele for folosi MyISAM cu opțiunea *skip\_innodb* , MyISAM nu poate fi dezactivat.

O altă dificultate ar fi aceea ca nu poate fi înlocuit masterul cu unul din slave-urile terminale. Este dificila promovarea uni slave in locul sau pentru că masterul intermediar de distribuție va avea aproape sigur coordonate diferite ale „binary log”-ului decât masterul original.

**Arbore sau piramida**

Daca un master trebuie replicat către un număr mare de slave-uri, chiar daca rețeaua trebuie distribuita geografica sau se încearcă o creștere a capacitații de citire se poate folosi cu succes o proiectare de tip piramida ca in figura:



Avantajul acestui tip de proiectare este scăderea încărcării pe master la fel cum făcea masterul de distribuție din arhitectura anterioara. Dezavantajul este ca orice defectare in nivelurile intermediare vor afecta multiple servere mai jos in ierarhie, lucru care nu s-ar fi întâmplat daca ele erau conectate la master direct. De asemena cu cat sunt mai multe niveluri cu atât este mai complicata repararea defectărilor.

**3.2.5 Administrare si mentenanță**

Setarea replicării probabil nu este un lucru care se va face constat, doar daca exista multe servere. O data ce a fost setata, monitorizarea si administrarea replicării topologiei implementate va fi on proces repetitiv, oricât de multe servere ar fi.

Aceste lucruri trebuie automatizate cat mai mult. Se pot folosi programe care includ Nagios, MySQL Enterprise Monitor sau MonYOG.

**Monitorizarea replicării**

Replicare creste nivelul de complexitate al monitorizării. Deși replicarea se realizează atât pe master cat si pe slave, lucrul intens este desfășurat pe slave si aici apar cel mai des problemele. Toate salve-urile trebuie sa replice corect fără erori si cu întârzieri cat mai mici. Informații privind aceste aspecte sunt generate automat si creșterea performantei este la mana administratorului.

Pe master se poate folosi comanda SHOW MASTER STATUS care arata poziția curenta in „binary log” si configurarea. Se poate interoga masterul in legătură cu ce „binary log”-uri exista pe disc.

**mysql> SHOW MASTER LOGS;**

**+------------------+-----------+**

**| Log\_name | File\_size |**

**+------------------+-----------+**

**| mysql-bin.000220 | 425605 |**

**| mysql-bin.000221 | 1134128 |**

**| mysql-bin.000222 | 13653 |**

**| mysql-bin.000223 | 13634 |**

**+------------------+-----------+**

Aceasta informație este folositoare in determinarea parametrilor dați comenzii PURGE MASTER LOGS. Se pot vizualiza evenimente de replicare din „binary log” cu comanda BINLOG EVENTS. Ca exemplu va fi creat un tabel pe un server nefolosit. Pentru ca aceasta este singura comanda se va putea vizualiza:

**mysql> SHOW BINLOG EVENTS IN 'mysql-bin.000223' FROM 13634\G**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Log\_name: mysql-bin.000223**

**Pos: 13634**

**Event\_type: Query**

**Server\_id: 1**

**End\_log\_pos: 13723**

**Info: use `test`; CREATE TABLE test.t(a int)**

**Măsurarea întârzierii pe slave**

Una din cele mai comune lucru care trebuie monitorizate este cat de întârziat este slave-ul fata de master. Chiar daca coloana *Seconds\_behind\_master* din SHOW SLAVE STATUS teoretic arata întârzierea slave-ului, nu este întotdeauna corecta din mai multe motive:

* Slave-ul calculează *Seconds\_behind\_master* comparând timpul de pe server cu cel din „binary log” deci slave-ul nu poate calcula întârzierea daca procesează un eveniment.
* Slave-ul va raporta NULL daca procesele de pe slave nu rulează.
* Unele erori sau rețeaua instabila intre master si slave pot împiedică replicarea si pot opri firele de execuție de pe slave si *Seconds\_behind\_master*  va indica 0 si nu o eroare.
* O tranzacție foarte mare poate cauza fluctuație a întârzierii. De exemplu daca o tranzacție care actualizează date este deschisa timp de o ora si apoi se realizează, întârzierea va fi înregistrată după ce ea a avut loc. Când slave-ul procesează instrucțiuni va raporta ca este cu o ora in urma apoi va ajunge din nou la 0.
* Daca un master de distribuție are slave-uri întârzierea raportata va fi 0 chiar daca există întârziere relativa la primul master in ierarhie.

Soluția pentru aceasta problema este ignorarea câmpului *Seconds\_behind\_master* si măsurarea întârzierii in alt mod prin care se poate observa direct. O soluție este *heartbeat record* ceea ce este instanță de timp ce va fi actualizata la fiecare secunda pe master. Pentru a calcula întârzierea se va scade acest timp din timpul curent de pe slave. Aceasta metoda este toleranta la toate problemele enumerate mai sus si arata la ce moment de timp datele vor fi sincronizate pe slave. O implementare a replicării *heartbeat* este *pt-heartbeat*, inclusa in Percona.

Niciuna din aceste metrici nu pot spune cu exactitate cat ii ia slave-ului sa ajungă la curent cu masterul. Acest lucru depinde de o mulțime de factori cum ar fi puterea slave-ului sau cate evenimente de scriere se transmit.

**Determinarea consistentei slave-ului cu masterul**

Într-o lume perfecta, un slave trebuie sa fie o copie exacta a masterului. Dar in realitate, erori in replicare por cauza ca datele sa fie corupte, nesincronizate cu masterul. Chiar daca aparent nu sunt erori, slave-urile pot deveni nesincronizate pentru ca anumite funcționalități nu sunt replicate corect, bug-uri, pierderi pe rețea, încheieri neașteptate ale programului, întreruperea alimentarii sau alte defecte.

Aceasta este o regula si nu o excepție ceea ce înseamnă ca verificarea slave-urilor ca si consistenta ar trebui sa fie un proces de rutina. Acest lucru este in special important daca replicarea se folosește pentru backup pentru ca nu se dorește coruperea datelor de pe un backup.

Nu exista o funcționalitate care sa verifice acest lucru. Exista câteva sume de control cum ar fi CHECKSUM TABLE dar nu este recomandata verificare slave-ului cu masterul in timp ce serverele rulează.

Percona oferă o metoda care rezolva aceasta problemă si alte câteva : *pt-table-checksum.* Acest instrument are cateva funcții de luat in seama, comparație in paralel a mai multor servere in același timp, dar cea mai de folos este aceea de a verifica daca un slave este sincronizat cu masterul. Funcționează folosind instrucțiuni INSERT … SELECT pe master.

Aceste instrucțiuni fac o suma de control a datelor si inserează rezultatul într-o tabela. Acestea sunt replicate si se executa pe slave. Se pot compara rezultatele de pe master si de pe server si se poate observa o diferență. Deoarece acest proces funcționează prin replicare, nu este nevoie ca tabelele sa fie închise (locked) pe ambele servere.

Un mod de folosire al acestui instrument este:

**$ pt-table-checksum --replicate=test.checksum --chunksize 100000 --sleep-coef=2**

**<master\_host>**

Aceasta comanda face suma de control a tuturor tabelelor si încearcă procesarea pe bucăți de aproximativ 100000 de rânduri si inserează valorile in *test.checksum.* Se oprește după fiecare bucata, si așteaptă de doua ori mai mult timp decât a durat execuția bucății anterioare. Acest lucru asigura ca nu vor fi blocate operațiile normale de pe baza de date.

După ce replicarea are loc un simplu query poate verifica daca slave-ul este in concordanta cu masterul. *pt-table-checksum* poate descoperi salve-urile rulează query-ul pe fiecare server si expune rezultatul automat. Următoarea comandă va pătrunde pe o adâncime de 10 nivele in ierarhia salve-urilor, rulata pe master si va arata tabelele care diferă de master:

**$ pt-table-checksum --replicate=test.checksum --replcheck 10 <master\_host>**

**4. Tehnologii utilizate**

**4.1 HTML**

**4.2 CSS**

**4.3 JavaScript**

**4.4 jQuery**

**4.5 Java**

**4.5.1 Introducere**

**Ce este Java**? Java este un limbaj de programare de nivel înalt, dezvoltat in anul 1995 de către SUN Microsystems, si utilizat astăzi pe scara larga atât in aplicațiile desktop sau internet cat si pentru cele destinate dispozitivelor mobile.

**De ce Java?** Java a fost de la bun început un limbaj gândit pentru ușurința si eficienta creării de software, eliminând neajunsurile altor limbaje (de ex. C++). Calitatea sa principala este independentă de platformă - un program scris o dată poate fi rulat pe diverse arhitecturi (Intel x86, SPARC etc.) fără a fi necesara ajustarea lui.

**Unde este secretul?** Dintotdeauna, una din problemele majore ale limbajelor a fost portabilitatea. Java rezolvă această problemă prin faptul ca programele, odată compilate, generează cod mașina care nu este destinat direct procesorului calculatorului gazda, ci unui procesor virtual Java, simulat de către un software ce poarta numele de Java Virtual Machine (JVM). JVM „traduce” codul mașina Java in instrucțiuni înțelese de către procesorul gazdă. Existând cate o varianta JVM pentru toate sistemele de operare si arhitecturile de procesor - însă toate simulând același procesor virtual Java! - codul mașină Java este practic independent de platforma si arhitectura.

Mașina virtuala Java

Compilare

|elemente independente de platforma | |elemente strict dependente de platforma|

1. **Platforme disponibile**

Date fiind direcțiile multiple de utilizare Java si necesitățile lor diferite, Oracle oferă in acest moment trei platforme Java:

* **Java SE** (Standard Edition) - platforma Java de baza pentru aplicații desktop sau server. Cuprinde componentele necesare pentru orice aplicație Java, inclusiv design de interfețe grafice complexe.
* **Java EE** (Enterprise Edition) - platforma Java dedicata aplicațiilor server, cu grad sporit de complexitate. Java EE include resursele disponibile in Java SE si adaugă elemente necesare serverelor de aplicație folosite pentru aplicații distribuite.
* **Java ME** (Micro Edition) - platforma Java dedicata dispozitivelor portabile sau cu resurse limitate (telefoane mobile, PDA-uri etc). Platforma include facilitați built-in de adaptare a platformei Java la platforma hardware pe care va rula, facilitați de comunicare in rețea si de realizare de interfețe grafice etc.

1. **Crearea unui program - Compilare/Interpretare**

Pentru a putea scrie si rula programe Java in mod eficient, programatorul are nevoie de următoarele resurse:

* **Java Runtime Environment** (JRE) - reprezintă totalitatea resurselor necesare pentru a putea rula programe Java. Printre ele se numără mașina virtuala si un set minimal de clase predefinite Java, corespunzătoare operațiilor des întâlnite in programare (ex: lucrul cu șiruri de caractere, operații matematice uzuale etc).
* **Java Development Kit** (JDK) - pachetul conține resursele necesare creării de programe Java: compilator, generator de documentație, arhivator si o întreagă serie de alte utilitare implicate direct sau indirect in procesul de dezvoltare. In general JDK conține si JRE, pentru a putea rula aplicațiile create.
* **un mediu de dezvoltare Java** (IDE - Integrated Development Environment). Deși codul Java poate fi scris folosind orice editor de text, un mediu de dezvoltare oferă multiple facilități care sporesc eficienta programatorului (un editor avansat, managementul facil al proiectelor si resurselor atașate acestora, debugging avansat etc). Putem privi IDE-ul ca pe o interfață la uneltele puse la dispoziție de JDK; IDE-ul se folosește de ele si nu poate funcționa in lipsa lor.

Sunt disponibile doua categorii de medii de dezvoltare:

* medii de dezvoltare open-source - sunt softuri public disponibile, care pot fi descărcate si rulate de către orice utilizator. Proeminente sunt NetBeans (care beneficiază de sprijin din partea Sun si ulterior Oracle) si Eclipse (proiect inițiat si sprijinit de IBM)
* medii de dezvoltare proprietare - sunt softuri disponibile contra cost. Exemple: JBuilder, IntelliJ Idea, MyEclipse etc.

Atât JRE cat si JDK sunt public disponibile pe http://java.sun.com. JDK poate fi descărcat in cel puțin doua variante – de sine stătător („stand-alone”) sau la pachet cu mediul de dezvoltare NetBeans („bundle”).

**Imaginea de ansamblu:**

Programele scrise in limbajul Java iau forma unui ansamblu de fișiere text cu extensia *.java,* ce conțin așa-numitul *cod sursa* (instrucțiunile Java ce compun programul). Aceste fișiere sunt apoi compilate cu ajutorul compilatorului *javac,* rezultând un set de instrucțiuni in cod mașină ce se va executa pe procesorul virtual Java si care poarta denumirea de *bytecode.* Bytecode-ul este plasat într-un fișier având același nume cu cel sursă, dar cu extensia *.class.* La rulare, mașiniivirtuale i se pasează ca argument numele clasei ce reprezintă punctul de intrare al programului; mașina virtuala încarcăbytecode-ul clasei corespunzătoare (fișierul .class) si executa codul cuprins in fișier.

Program.java Program.class Monitor PC

public class Program{

public static void main(String[] args){

System.out.println(„Hello!”);

}

}

Hello!

10101000101011110101010111001010100010101000011011111010

java

javac

Conform imaginii de mai sus, deducem ca Java este atât un limbaj compilat (codul sursa este transformat in bytecode) cat si unul interpretat (bytecode-ul este interpretat de către mașina virtuala Java astfel încât sa poată fi executat pe procesorul gazda).

**4.5.3 Sintaxa Java**

Java a este un limbaj concurent orientat pe clase si obiecte si special proiectat pantru a avea cat mai putine dependente. In 2014 este cel mai utilizat limbaj de programare cu peste 9 miliane de programatori.Intentia este ca programatorii sa poata scrie o data si sa ruleze oriunde („write once; run anywhere”). Limbajul este asemanator cu cel al C/C++ dar avant mai putine functionalotati de nivel jos decat ele.

**4.5.4 OOP**

Java a fost creat ca un limbaj obiect orientat inca de la inceput.

**4.5.5 Integrarea cu baza de date**

Integrarea cu baza de date in mod clasic se face in Java folosind libraria JDBC. In acest proiect s-a folosit JPA si Hibernate pentru a usura accesul la baza de date. Intotdeauna a fost o problema trecerea datelor din format tabelar in obiecte. De aceea au fost creeate librariile ORM (object relational mapping) care permit accesul mai usor la baza de date din limbajul de programare. In Java exista specificatia JPA care este o serie de interfete ce defineste aceasta mapare. Pentru aceasta interfata exista o serie de implementari diferite printre care cea mai cunoscuta Hibernate.

**5. Proiectarea aplicației**

**5.1 Prezentare generala a interfeței**

**5.1.1 Sectiunea de autentificare**

**5.1.2 Secțiunea de stiri si articole**

**5.1.3 Secțiunea evenimente**

**5.1.4 Sectinea de intrebari si raspunsuri**

**5.1.5 Sectiunea de administrator**

**5.2 Structura bazei de date**

**5.3 Testarea replicării**

**6. Concluzii**

**7. Bibliografie**

* High Performance MySQL\_ Optimization, Backups, Replication, and More - Baron Schwartz - 2012
* Baze de date relationale si aplicatii - Felicia Ionescu – 2004
* Java How to Program (early objects) (9th Edition) – Paul Deitel, Harvey Deitel – 2012
* Spring in action - Craig Walls – 2011
* Pro JPA 2 - Mike Keith, Merrick Schincariol - 2013
* Oracle – <http://www.oracle.com/technetwork/topics/newtojava/unravelingjava-142250.html>
* Java API - [http://docs.oracle.com/javase/6/docs/api](http://docs.oracle.com/javase/6/docs/api/)
* <http://www.wikipedia.com>
* <http://www.w3schools.com>

**8. Anexe**