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

Introducere

* 1. Scurt istoric
  2. Noțiuni generale
  3. Tipuri de date

1. MariaDB
   1. Introducere
   2. Scurt istoric
   3. Concepte
   4. MariaDB vs. MySQL
   5. MariaDB vs. NoSQL
   6. Replicarea
   7. Avantaje si dezavantaje
2. Java
   1. Introducere
   2. Scurt Istoric
   3. Sintaxa Java
   4. OOP
   5. Integrarea cu baza de date
3. Tehnologii user-interface
   1. HTML
      1. Introducere
      2. Structura unei pagini HTML
      3. Tag-uri, elemente si atribute
   2. CSS
      1. Introducere
      2. Selectori in CSS
      3. Proprietăți CSS
   3. JavaScript
   4. jQuery
4. Proiectarea aplicației
   1. Prezentare generala a interfeței
      1. Secțiunea X
      2. Secțiunea Y
   2. Structura bazei de date
   3. Testarea replicării
5. Concluzii
6. Bibliografie
7. Anexe

**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:



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.

**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.

**Replicarea mai în amănunt**

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**