

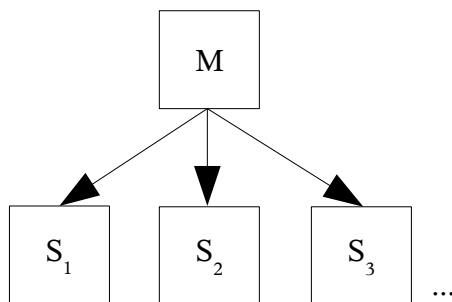
MySQL Multi-Master – Single-Slave – Replication (aka Saskia)

Oli Sennhauser

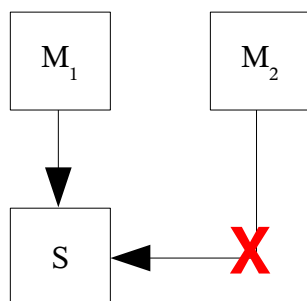
Rebenweg 6
CH – 8610 Uster
Switzerland
oli.sennhauser@bluewin.ch

Introduction

MySQL provides its replication for High Availability (HA) and for read Scale-out. Generally it is known that in a MySQL replication you can only replicate from one Master to many slaves.

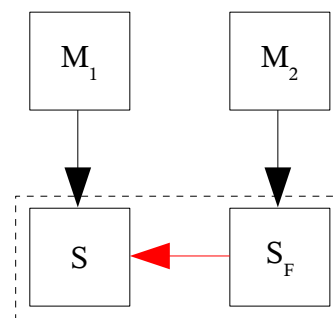


It is told, that it is NOT possible to replicate from several masters to a single slave. It is also told, that this will be solved in future releases of MySQL.



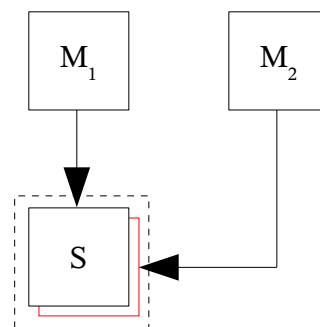
But possibly some MySQL users cannot or do not want to wait. Thus we were thinking about a possible simple solution with MySQL's own tools. The idea behind it is quite simple (and thus matches to the KISS concept): One master replicates to a simple slave. The second master replicates to an other slave which is connected to the first slave

through simple FEDERATED TABLES...



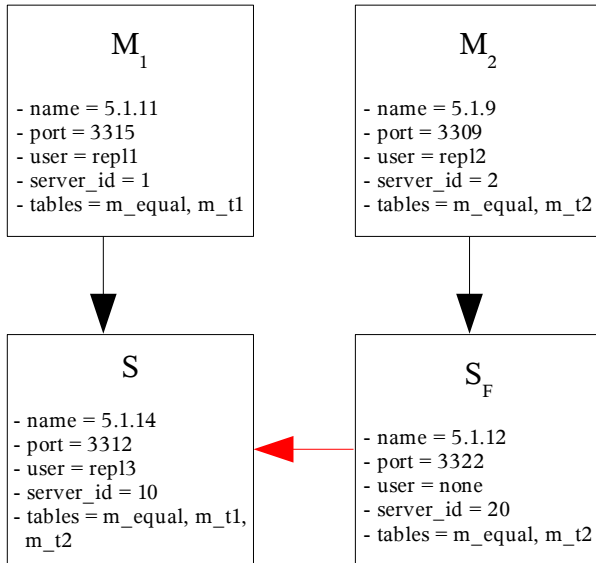
The “federated Slave” (S_F) can be put on the same server as the real Slave (S). From the application point of view everything looks transparent. And from the server infrastructure it also does not look to complicated. This concept does not work, if you have a write I/O bottleneck on the slave server.

Replication should not have a problem with this configuration and should work as usual. If somebody finds any error in this concept, please let me know.



Prof of Concept (PoC)

For proving, if this concept works, the following playground was built:



The name is an internal name of my set-up (corresponds to the underlying MySQL version). Because all 4 databases (instances) are running on the same host different ports have to be chosen for each database. On both masters a replication account has to be created. The replication account on the Slave (S) is used for the FEDERATED tables to propagate the data. To prevent any conflicts four different server_id's are chosen as usual for replication.

On both Masters M₁ and M₂ two tables are created. One table to simulate a table which exists in both masters (m_equal) and one table to simulate tables which only exist on one of both masters (m_t1 and m_t2).

All three tables have to exist on the Slave (S). The tables on the federated Slave have to be created differently.

Set-up

Let us start with the two masters: First the server_id and binary logging (log_bin) have to be set and the databases have to be restarted.

Then the normal replication accounts have to be created on the masters:

```

master1> GRANT REPLICATION SLAVE
         ON *.*
         TO 'repl1'@'%'
         IDENTIFIED BY 'repl1';

master2> GRANT REPLICATION SLAVE
         ON *.*
         TO 'repl2'@'%'
         IDENTIFIED BY 'repl2';
  
```

This step can be checked as follows:

```

master1> SHOW VARIABLES LIKE 'server%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| server_id     | 1     |
+-----+-----+

master1> SHOW MASTER STATUS;
+-----+-----+
| File          | Position |
+-----+-----+
| bin_log.000009 | 362     |
+-----+-----+

master2> SHOW VARIABLES LIKE 'server%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| server_id     | 2     |
+-----+-----+

master2> SHOW MASTER STATUS;
+-----+-----+
| File          | Position |
+-----+-----+
| bin_log.000001 | 98      |
+-----+-----+
  
```

We skip the step to do a proper backup here because we only want to prove that the concept works. In fact it is a little bit trickier than setting up a simple replication (mysqldump --skip-add-drop-table, ...). In this area some more investigation has to be done.

Now the tables have to be set up. On both masters (M₁ and M₂) as well as on the Slave (S) this table has to be created:

```

mysql> CREATE TABLE m_equal (
      id INT UNSIGNED NOT NULL AUTO_INCREMENT
      PRIMARY KEY
      , ts TIMESTAMP DEFAULT
      CURRENT_TIMESTAMP
      , data VARCHAR(32)
);
  
```

On Master 1 (M₁) and the Slave (S) the following table was created:

```

mysql> CREATE TABLE m_t1 (
      id INT UNSIGNED NOT NULL AUTO_INCREMENT
      PRIMARY KEY
      , ts TIMESTAMP DEFAULT
      CURRENT_TIMESTAMP
      , data VARCHAR(32)
);
  
```

On Master 1 (M₁) and the Slave (S) the analog table was created:

```
mysql> CREATE TABLE m_t2 (
  id INT UNSIGNED NOT NULL AUTO_INCREMENT
  PRIMARY KEY
  , ts TIMESTAMP DEFAULT
  CURRENT_TIMESTAMP
  , data VARCHAR(32)
);
```

Now nearly everything is set up except our federated Slave (S_F). For setting up the federated Slave we first need an account on the main Slave (S):

```
slave> GRANT INSERT, UPDATE, DELETE
  ON *.*
  TO 'feed_user'@'%'
  IDENTIFIED BY 'feed_user';
```

Then the two tables which belong to the Master 2 (M₂) have to be created in the federated Slave (S_F) as FEDERATED tables:

```
slave_f> CREATE TABLE m_t2 (
  id INT UNSIGNED NOT NULL AUTO_INCREMENT
  PRIMARY KEY
  , ts TIMESTAMP DEFAULT
  CURRENT_TIMESTAMP
  , data VARCHAR(32)
) ENGINE=FEDERATED
CONNECTION='mysql://rep13:rep13@master:3312/t
est/m_t2';

slave_f> CREATE TABLE m_equal (
  id INT UNSIGNED NOT NULL AUTO_INCREMENT
  PRIMARY KEY
  , ts TIMESTAMP DEFAULT
  CURRENT_TIMESTAMP
  , data VARCHAR(32)
) ENGINE=FEDERATED
CONNECTION='mysql://rep13:rep13@master:3312/t
est/m_equal';
```

Now the slaves can be started as usual:

```
slave> CHANGE MASTER TO
  master_host='master'
  , master_port=3309
  , master_user='rep12'
  , master_password='rep12'
  , master_log_file='bin_log.000009'
  , master_log_pos=362;

slave> START SLAVE;
```

```
slave_f> CHANGE MASTER TO
  master_host='master'
  , master_port=3309
  , master_user='rep12'
  , master_password='rep12'
  , master_log_file='bin_log.000001'
  , master_log_pos=98;

START SLAVE;
```

Testing

When the set up is finished we can start with some tests:

```
master1> INSERT INTO m_t1
  VALUES (1, NOW(), 'Bla');

master2> INSERT INTO m_t2
  VALUES (1, NOW(), 'Bla');

slave> SELECT *
  FROM m_t1 t1
  JOIN m_t2 t2 ON t1.id = t2.id;
```

id	ts	data	id	ts
1	2006-12-20 20:40:48	Bla	1	2006-12-20 20:41:02
		Bla		

A simple INSERT seems to work. :-)

No a little bit more difficult:

```
master1> INSERT INTO m_equal
  VALUES (1, now(), 'Bla');

master2> UPDATE m_equal
  SET data = 'Bla bla'
  WHERE id = 1;

slave> SELECT *
  FROM m_equal;
```

id	ts	data
1	2006-12-20 16:16:36	Bla bla

Also this seems to work correctly (but should NEVER be done because master 2 should not know anything about row 1). So it looks like everything works like in a normal replication...

Transactions

MySQL supports transactions on FEDERATED tables since release 5.1. Even transactions seems to work (the tables have to be change to InnoDB before):

```
slave> SELECT * FROM m_t2;
+-----+-----+-----+
| id | ts                | data      |
+-----+-----+-----+
| 1  | 2006-12-21 13:45:10 | Trx test 1 |
+-----+-----+-----+

master2> mysql> BEGIN;

master2> INSERT INTO m_t2 VALUES (2, '2006-12-21 13:49:10', 'Trx test 1');

slave> SELECT * FROM m_t2;
+-----+-----+-----+
| id | ts                | data      |
+-----+-----+-----+
| 1  | 2006-12-21 13:45:10 | Trx test 1 |
+-----+-----+-----+

master2> UPDATE m_t2 SET data = 'Trx test 2'
WHERE id = 1;

master2> COMMIT;

slave> SELECT * FROM m_t2;
+-----+-----+-----+
| id | ts                | data      |
+-----+-----+-----+
| 1  | 2006-12-21 13:49:10 | Trx test 2 |
| 2  | 2006-12-21 13:49:10 | Trx test 1 |
+-----+-----+-----+
```

Also a rollback works.

Conflicts

Additionally to the usual Master-Master conflicts we also have here some other conflict potential:

- `AUTO_INCREMENT` and `NOW()/CURRENT_TIMESTAMP` values are not propagated through the FEDERATED table. These values get lost and this causes "corrupt data" on the Slave (S). `Auto_increment_increment` and `Auto_increment_offset` do not help here. See also Bugs.
- Rows in the shared table can be deleted from the Master where they never have been inserted (see example above). This causes the other master to be out of sync and leads to possibly corrupt data.

Recommendations

Accessing the same table from different masters (`m_equal`) is tricky. Errors can creep in which are difficult to detect later. So it is recommended to use a shared table with care.

On the dedicated tables (`m_t1` and `m_t2`) the `AUTO_INCREMENT` should work, if the values are in sync. But they can easily get out of sync which causes corrupt data. Using this kind of set-up is pretty dangerous and has to be tested very careful.

Limitations

It looks like normal DML (`INSERT`, `UPDATE` and `DELETE`) works fine. DDL (`CREATE TABLE`, `ALTER TABLE`, etc.) will only work on the Master 1 (M_1) chain but not on the Master 2 (M_2) chain! `TRUNCATE TABLE` on the other hand seems to work properly on both chains.

Each FEDERATED table creates a permanent database connection to the data provider database (at least with release 5.1.12). This means, that this kind of replication will NOT work with too many tables. The limit will be around some dozens to some hundreds of federated tables. But this should be sufficient enough for most uses.

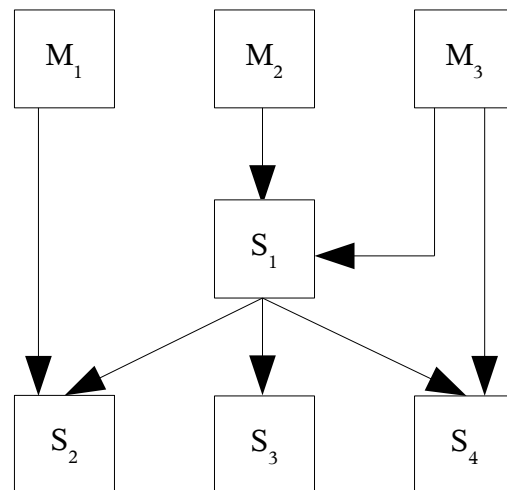
For FEDERATED tables and replication the usual restrictions and limitations are applicable [1], [2].

`SET` statements are not inherited through the FEDERATED storage engine. This makes it impossible to use features like `AUTO_INCREMENT` or `NOW()/CURRENT_TIMESTAMP`.

Row based replication, which is new in 5.1 does not work in the Master 2 chain.

Outlook

This kind of replication enables to design much more complex database architectures. It allows us to model some kind of data flows from several different databases to some others...



If someone introduces such a construct in production I would very appreciate it to get some feedback from out there...

Thanks to ...

... Ralf Gebhardt, Saskia Schweitzer and Jens Bollmann (who still hangs on an European Airport) for the contribution.

References

[1] **Limitations of the FEDERATED Storage Engine:** <http://dev.mysql.com/doc/refman/5.1/en/federated-limitations.html>

[2] **Replication Features and Known Problems:** <http://dev.mysql.com/doc/refman/5.1/en/replication-features.html>

Bugs

- #20724, FEDERATED does not honour SET INSERT ID: <http://bugs.mysql.com/bug.php?id=20724>

- **#20026, FEDERATED lacks support for auto_increment_increment and auto_increment_offset:**
<http://bugs.mysql.com/bug.php?id=20026>