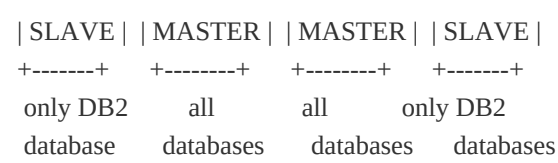
MySQL replication: Cross-Database updates using Procedures and dynamically-switched row-based Replication

What a lengthy title to describe what this is about. Dough!

Well, I wanted to solve an interesting problem: Imagine a multi-master MySQL cluster, where databases are only fully replicated between masters, but only partially between connected stub slaves.

Now, assume there is two databases, one is called **DB1**, the other is called **DB2**.
Both master servers carry both databases, while the connected slaves get only **DB2** (enforced by a **replicate-wild-do-table=DB2.%** config statement in the slave's my.cnf config file).



Now, let's image a query, which populates entries in **DB2** by selecting data from DB1.
INSERT INTO DB2.destTable1 SELECT * FROM DB1.sourceTable1;

The way how MySQL replication works, this query is about to fail on the slaves because the referenced source table does not exist.
Of course, this is depending heavily on the actual configuration of the master, and how the log updates in the binary log are handled.

If statement-based replication is enabled, the replication will totally blow up.
On the other hand, if row-based replication is chosen, it is likely to complete.

Why? Because for statement-based replication, the actual statement is replicated as shown above.
With row-based replication, the master will, while writing the logs, expand the statement into the actual actions carried out, so the exact result can be passed on to the slaves.

Let's see what this looks like:
SET INSERT_ID=1734227546/*!*/;
INSERT INTO destTable1 (field1, field2, field3, field4) VALUES ('val1', 'val2', 'val3', 'val4');

The advantage: The query can be actually carried out on the slave even if the referred to source Table would not exist.
This is was row-based replication is about.

Now, there is a third option for MySQL, which is mixed-mode replication. Actually, in more recent versions of MySQL, this is the default.
In mixed-mode, MySQL will choose by itself, if and under what conditions, statement-based or row-based replication applies.

This is actually a good default in most cases and I use it on all my setup. But in said scenario above, mixed-mode replication will cause unexpected replication failure.

So, it would be fine if there is a way to runtime-switch the replication mode, wouldn't it?
Luckily there is.

One may either do this globally, or more feasible in this case, per-session.
So first force MySQL into row-based replication before executing the actual query, and then revert-back to mixed-mode replication afterwards after the query has been run.

```
set session binlog_format = 'ROW';
INSERT INTO DB2.destTable1 SELECT * FROM DB1.sourceTable1;
set session binlog_format = 'MIXED';
```

This can even be used within a stored procedure. So I can infact add a stored procedure on DB1, which carries out changes to DB2.

```
CREATE DEFINER=`root@localhost` PROCEDURE `procSampleProcedure`()
    MODIFIES SQL DATA
BEGIN
set session binlog_format = 'ROW';
INSERT INTO DB2.destTable1 SELECT * FROM DB1.sourceTable1;
set session binlog_format = 'MIXED';
END
```

This is very nice indeed as I can still run MySQL in mixed-mode by default, and only force it into RBR (row-based replication) when needed.
Essentially, this works around potential replication lockups due to missing dependant tables.