MySQL table wide structure migration/synchronization v1.3

This how-to is dedicated to all Software Developers, Software Engineers, Database Developers, Database Engineers and all other IT people around the world, having problems to migrate big complex unclean system ‘s, applications (database’s or table’s) to a new clean structure.

The scenario for my sample test case:

We like to refactor an application, the code of the application should be changed step per step and also the old antiquated unclean database structure should be convict to a new one.

Now the problem on this case is, to keep the old and the new database (table) with different structure synchronously. Old modules of the application need the old database and new modules of the application need the new database, but with the same content. A really good way to do this is to use the trigger functionality of MySQL combined with a timestamp accurate to milliseconds or microseconds. Here MySQL has a small stumble; the timestamp is only accurate to seconds and not useful for a clean synchronization.

The trick to fix this problem is to use a CHAR field for the timestamp and add the first 8 character of the UUID (http://en.wikipedia.org/wiki/Universally_Unique_Identifier), so we get a unique timestamp for the synchronization.

Here a very small, but plain sample:

The structure difference is highlighted!

The old unclean table “told”:

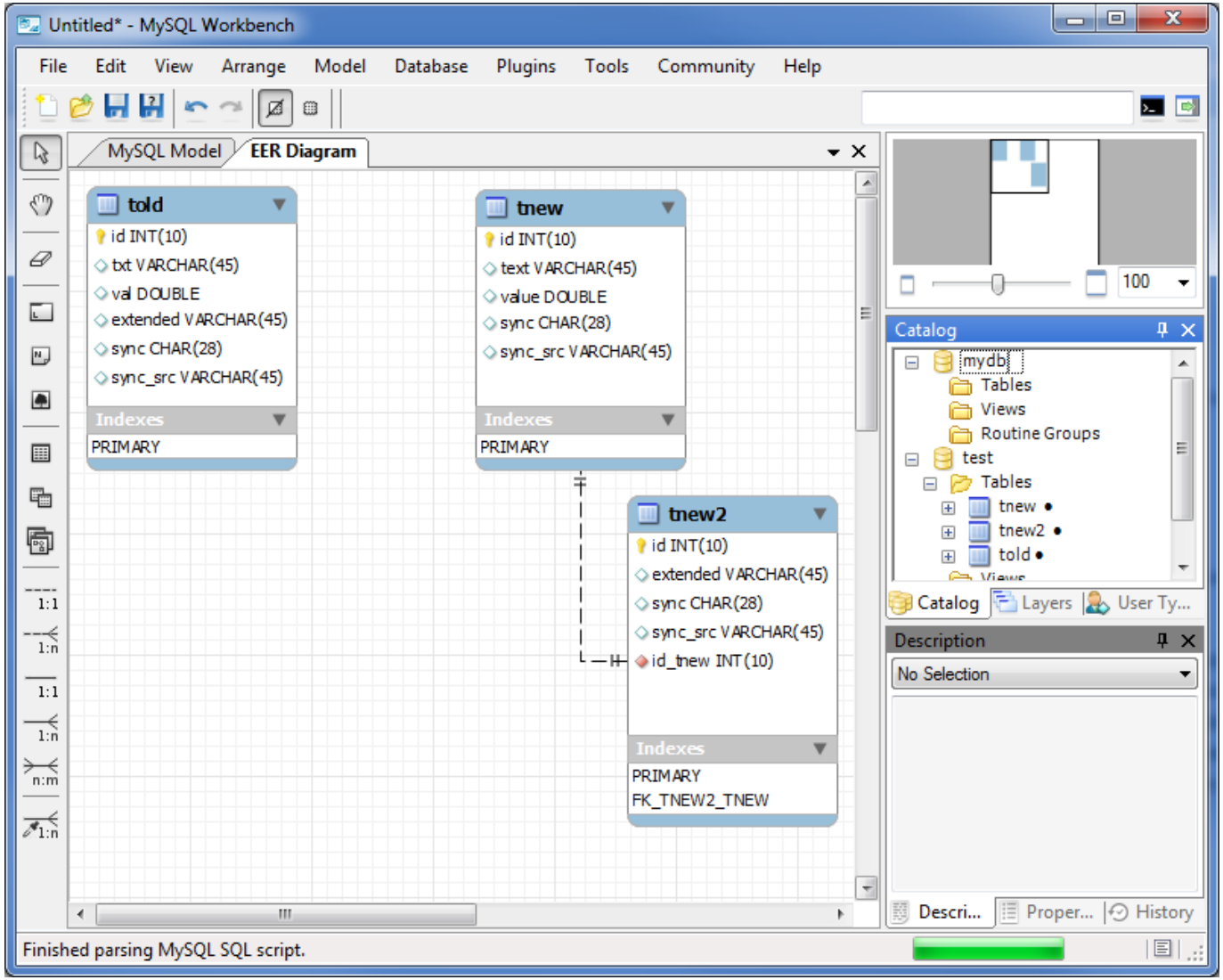
```
DROP TABLE IF EXISTS `test`.`told`;
CREATE TABLE  `test`.`told` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `txt` varchar(45) DEFAULT NULL,
  `val` double DEFAULT NULL,
  `extended` varchar(45) DEFAULT NULL,
  `sync` char(28) DEFAULT NULL,
  `sync_src` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

The new clean table “tnew”:

```
DROP TABLE IF EXISTS `test`.`tnew`;
CREATE TABLE  `test`.`tnew` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `text` varchar(45) DEFAULT NULL,
  `value` double DEFAULT NULL,
  `sync` char(28) DEFAULT NULL,
  `sync_src` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

The child table “tnew2” of the table “tnew”:

```
DROP TABLE IF EXISTS `test`.`tnew2`;
CREATE TABLE  `test`.`tnew2` (
 `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
 `extended` varchar(45) DEFAULT NULL,
 `sync` char(28) DEFAULT NULL,
 `sync_src` varchar(45) DEFAULT NULL,
 `id_tnew` int(10) unsigned NOT NULL,
 PRIMARY KEY (`id`) USING BTREE,
 KEY `FK_TNEW2_TNEW` (`id_tnew`),
 CONSTRAINT `FK_TNEW2_TNEW` FOREIGN KEY (`id_tnew`) REFERENCES `tnew` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE ) ENGINE=InnoDB;
```

All tables need the timestamp and the source field for the synchronization.

Here the triggers to keep the tables with different structure synchronously:

```
USE `test`;
DELIMITER //

##################################################################################################################
# ----------------------------------------------------------------------------------------------------------------
# old table
DROP TRIGGER IF EXISTS test.told_before_update
//
CREATE TRIGGER test.told_before_update BEFORE UPDATE ON told
FOR EACH ROW
MAIN:
BEGIN
   # update the synchronization timestamp and set the source for the synchronization
   IF NEW.sync <= OLD.sync THEN
      SET NEW.sync = CONCAT(CONCAT(NOW(),'.'), SUBSTRING(UUID(),1,8));
      SET NEW.sync_src = 'told';
   END IF;
END;
//
# ----------------------------------------------------------------------------------------------------------------
DROP TRIGGER IF EXISTS test.told_after_update
//
CREATE TRIGGER test.told_after_update AFTER UPDATE ON told
FOR EACH ROW
MAIN:
BEGIN
   # skip the update/synchronize, if the timestamp of the new table is higher (to not play ping-pong)
   IF (SELECT sync FROM tnew WHERE id = OLD.id) >= NEW.sync THEN
      LEAVE MAIN;
   END IF;

   # here you can put each logic for the migration
   UPDATE test.`tnew` SET id = NEW.id, `text` = NEW.`txt`, `value` = NEW.`val`, sync = NEW.`sync`, sync_src = NEW.sync_src WHERE id =
NEW.id;
   UPDATE test.`tnew2` SET `extended` = NEW.`extended`, sync = NEW.sync, sync_src = NEW.sync_src WHERE id_tnew = NEW.id;
END;
//
```



```
# ------------------------------------------------------------------------------------------------------------------
DROP TRIGGER IF EXISTS test.told_before_insert
//
CREATE TRIGGER test.told_before_insert BEFORE INSERT ON told
FOR EACH ROW
BEGIN
   SET NEW.sync = CONCAT(CONCAT(NOW(),'.'), SUBSTRING(UUID(),1,8));
   SET NEW.sync_src = 'told';
END;
//
# ------------------------------------------------------------------------------------------------------------------
DROP TRIGGER IF EXISTS test.told_after_insert
//
CREATE TRIGGER test.told_after_insert AFTER INSERT ON told
FOR EACH ROW
MAIN:
BEGIN
   IF (SELECT COUNT(*) FROM tnew WHERE id = NEW.id) != 0 THEN
      LEAVE MAIN;
   END IF;

   # here you can put each logic for the migration
   INSERT test.`tnew` SET id = NEW.id, `text` = NEW.`txt`, `value` = NEW.`val`, sync = NEW.`sync`;
   INSERT test.`tnew2` SET `extended` = NEW.`extended`, sync = NEW.`sync`, sync_src = NEW.`sync_src`, id_tnew = NEW.id;
END;
//
# ------------------------------------------------------------------------------------------------------------------
DROP TRIGGER IF EXISTS test.told_after_delete
//
CREATE TRIGGER test.told_after_delete AFTER DELETE ON told
FOR EACH ROW
BEGIN
   IF (SELECT COUNT(*) FROM tnew WHERE id = OLD.id) = 1 THEN
      DELETE FROM test.`tnew` WHERE id = OLD.id;
   END IF;
END;
//
```



```
##############################################################################################################
# ------------------------------------------------------------------------------------------------------------
# new table (parent table)
DROP TRIGGER IF EXISTS test.tnew_before_update
//
CREATE TRIGGER test.tnew_before_update BEFORE UPDATE ON tnew
FOR EACH ROW
BEGIN
   # update the synchronization timestamp and set the source for the synchronization
   IF NEW.sync <= OLD.sync THEN
      SET NEW.sync = CONCAT(CONCAT(NOW(),'.'), SUBSTRING(UUID(),1,8));
      SET NEW.sync_src = 'tnew';
   END IF;
END;
//
# ------------------------------------------------------------------------------------------------------------
DROP TRIGGER IF EXISTS test.tnew_after_update
//
CREATE TRIGGER test.tnew_after_update AFTER UPDATE ON `tnew`
FOR EACH ROW
MAIN:
BEGIN
   # skip the update/synchronize, if the timestamp of the new table is higher or if the update come from a child table (to not play
ping-pong)
   IF (SELECT sync FROM told WHERE id = OLD.id) >= NEW.sync OR NEW.sync_src = 'tnew2' THEN
      LEAVE MAIN;
   END IF;

   # here you can put each logic for the migration
   UPDATE test.`told` SET id = NEW.id, `txt` = NEW.`text`, `val` = NEW.`value`, sync = NEW.`sync`, sync_src = NEW.sync_src WHERE id =
NEW.id;
END;
//
```



```
# ------------------------------------------------------------------------------------------------------------------
DROP TRIGGER IF EXISTS test.tnew_before_insert
//
CREATE TRIGGER test.tnew_before_insert BEFORE INSERT ON tnew
FOR EACH ROW
BEGIN
   SET NEW.sync = CONCAT(CONCAT(NOW(),'.'), SUBSTRING(UUID(),1,8));
   SET NEW.sync_src = 'tnew';
END;
//
# ------------------------------------------------------------------------------------------------------------------
DROP TRIGGER IF EXISTS test.tnew_after_insert
//
CREATE TRIGGER test.tnew_after_insert AFTER INSERT ON tnew
FOR EACH ROW
MAIN:
BEGIN
   IF (SELECT COUNT(*) FROM told WHERE id = NEW.id) != 0 THEN
      LEAVE MAIN;
   END IF;

   # here you can put each logic for the migration
   INSERT test.`told` SET id = NEW.id, `txt` = NEW.`text`, `val` = NEW.`value`, sync = NEW.`sync`, sync_src = NEW.sync_src;
END;
//
# ------------------------------------------------------------------------------------------------------------------
DROP TRIGGER IF EXISTS test.tnew_after_delete
//
CREATE TRIGGER test.tnew_after_delete AFTER DELETE ON tnew
FOR EACH ROW
BEGIN
   IF (SELECT COUNT(*) FROM told WHERE id = OLD.id) = 1 THEN
      DELETE FROM test.`told` WHERE id = OLD.id;
   END IF;
END;
//

##################################################################################################################
```



```
# ------------------------------------------------------------------------------------------------------------------
# new table 2 (child table of tnew)
DROP TRIGGER IF EXISTS test.tnew2_before_update
//
CREATE TRIGGER test.tnew2_before_update BEFORE UPDATE ON `tnew2`
FOR EACH ROW
MAIN:
BEGIN
   # update the synchronization timestamp and set the source for the synchronization
   IF NEW.sync <= OLD.sync THEN
      SET NEW.sync = CONCAT(CONCAT(NOW(),'.'), SUBSTRING(UUID(),1,8));
      SET NEW.sync_src = 'tnew2';
      # update the synchronization timestamp of the parent table, because it's controlling the synchronization
      UPDATE test.`tnew` SET sync = NEW.sync, sync_src = NEW.sync_src WHERE id = NEW.id_tnew;
   END IF;
END;
//
# ------------------------------------------------------------------------------------------------------------------
DROP TRIGGER IF EXISTS test.tnew2_after_update
//
CREATE TRIGGER test.tnew2_after_update AFTER UPDATE ON `tnew2`
FOR EACH ROW
MAIN:
BEGIN
   # skip the update/synchronize, if the timestamp of the new table is higher (to not play ping-pong)
   IF (SELECT sync FROM told WHERE id = OLD.id_tnew) >= NEW.sync THEN
      LEAVE MAIN;
   END IF;

   # here you can put each logic for the migration
   UPDATE test.`told` SET `extended` = NEW.`extended`, sync = NEW.sync, sync_src = NEW.sync_src WHERE id = NEW.id_tnew;
END;
//




# ------------------------------------------------------------------------------------------------------------------
```



```
DROP TRIGGER IF EXISTS test.tnew2_before_insert
//
CREATE TRIGGER test.tnew2_before_insert BEFORE INSERT ON tnew2
FOR EACH ROW
MAIN:
BEGIN
   # skip the update/synchronize, if the insert come from the old table (to not play ping-pong)
   IF NEW.sync_src = 'told' THEN
      LEAVE MAIN;
   END IF;

   SET NEW.sync = CONCAT(CONCAT(NOW(),'.'), SUBSTRING(UUID(),1,8));
   SET NEW.sync_src = 'tnew2';
   UPDATE test.`tnew` SET sync = NEW.sync, sync_src = NEW.sync_src WHERE id = NEW.id_tnew;
END;
//
# ----------------------------------------------------------------------------------------------------
DROP TRIGGER IF EXISTS test.tnew2_after_insert
//
CREATE TRIGGER test.tnew2_after_insert AFTER INSERT ON `tnew2`
FOR EACH ROW
MAIN:
BEGIN
   # skip the update/synchronize, if the record not exists on old table or if the insert come from the old table (to not play ping-
pong)
   IF (SELECT COUNT(*) FROM told WHERE id = NEW.id_tnew) = 0 OR NEW.sync_src = 'told' THEN
      LEAVE MAIN;
   END IF;
   # here you can put each logic for the migration
   UPDATE test.`told` SET `extended` = NEW.`extended`, sync = NEW.sync, sync_src = NEW.sync_src WHERE id = NEW.id_tnew;
END;
//
```

