Dealing with schema changes on large data volumes

Danil Zburivsky
MySQL DBA, Team Lead
Why Companies Trust Pythian

• Recognized Leader:
  • Global industry-leader in remote database administration services and consulting for Oracle, Oracle Applications, MySQL and SQL Server
  • Work with over 150 multinational companies such as Forbes.com, Fox Interactive media, and MDS Inc. to help manage their complex IT deployments

• Expertise:
  • One of the world’s largest concentrations of dedicated, full-time DBA expertise.

• Global Reach & Scalability:
  • 24/7/365 global remote support for DBA and consulting, systems administration, special projects or emergency response
Agenda

• Why schema changes are painful on large data sizes?
• Using standby for schema migrations
• “Shadow” tables approach
Schema changes are slow

- Most of the schema changes on InnoDB tables require table rebuild:
  - Add index
  - Add new column
  - Drop column
  - Rename column
- Table is locked during rebuild
- Becomes a problem when table is 20G in size
- There are some improvements in InnoDB plugin, but they don’t solve the problem in general
On Standby:

• `SET SQL_LOG_BIN = 0;`
• Apply schema changes on standby
• Failover application to standby
Using standby for schema changes

- Works fine for adding indexes
- Not as good for adding new columns
- Doesn’t work for renaming or dropping columns: replication will break
“Shadow” table approach

• Create new empty table with similar structure
• Apply schema changes on new table
• Copy data from original table to new table
• Synchronize using triggers
• Swap tables
Use case

• System with about 500G of InnoDB data
• Major application release affecting about 30 tables
• Largest one is 20G in size
• Database changes included: new columns, renaming columns, deleting columns, new indexes and complex data transformations
• Estimated time for applying all changes directly was 7 hours
• Using “shadow” tables database changes were applied in about 1 hour
The process

CREATE TABLE `t_original` (  `id` int(11) NOT NULL,  `A` varchar(50) DEFAULT NULL,  `B` varchar(50) DEFAULT NULL,  PRIMARY KEY (`id`) ) ENGINE=InnoDB DEFAULT CHARSET=latin1

CREATE TABLE t_new LIKE t_original;
ALTER TABLE t_new ADD COLUMN AB VARCHAR (100);
Triggers to keep data in sync

LOCK TABLE t_original WRITE;

CREATE TRIGGER t_original_ai AFTER INSERT ON t_original
FOR EACH ROW
REPLACE INTO t_new (id, A, B, AB) VALUES
(NEW.id, NEW.A, NEW.B, CONCAT (A,',',B));
Triggers to keep data in sync

CREATE TRIGGER t_original_ad AFTER DELETE ON t_original
FOR EACH ROW
DELETE FROM t_new WHERE id = OLD.id;

CREATE TRIGGER t_original_au AFTER UPDATE ON t_original
FOR EACH ROW
UPDATE t_new SET id = NEW.id,
A = NEW.A,
B = NEW.B,
AB = CONCAT (A,',',B)
WHERE id = OLD.id;

UNLOCK TABLES;
Copy data

INSERT IGNORE INTO t_new
(....)
SELECT ... FROM t_original
WHERE id>=? LIMIT N
Copy data. Sample code

```perl
$lastId=$minid;
$sql=<<SQL;
INSERT IGNORE INTO t_new(id, A, B, AB)
(SELECT id, A, B, CONCAT(A,',',B) )
FROM t_original
WHERE id>=? LIMIT 5000)
SQL

my $sth1 = $dbh->prepare($sql);
while ($rv > 1)
{
$dbh->do('START TRANSACTION');
$sth1->execute($lastId);
```
Copy data. Sample code

```php
$ssth = $dbh->prepare("SELECT id FROM t_original WHERE id >=".$lastId." LIMIT 5000");
$ssth->execute();
$rv = $ssth->rows;

while ((my $nextId) = $ssth->fetchrow_array())
{
    $lastId = $nextId;
    $totalrows = $totalrows + 1;
}
$dbh->do('COMMIT');
print "Print rows inserted $totalrows. Next id= $lastId\n";
```
Basic checks

```sql
SELECT COUNT(*)
FROM t_new
UNION
SELECT COUNT(*)
FROM t_original;

SELECT MAX(id), MIN(id)
FROM t_new
UNION
SELECT MAX(id), MIN(id)
FROM t_original;
```
During the release

RENAME TABLE t_original TO t_old;
RENAME TABLE t_new TO t_original;
DROP TRIGGERS;
Limitations

- Requires a unique key
- No existing triggers
- Need enough disk space for “shadow” tables
- Foreign keys
Foreign key issue

If there is an existing FK on t_original:
FOREIGN KEY (`fkId`) REFERENCES `t_original` (`id`)

It will be changed after RENAME to:
FOREIGN KEY (`fkId`) REFERENCES `t_old` (`id`)

Solution is a hack:
SET FOREIGN_KEY_CHECKS=0;
DROP TABLE t_old;
RENAME TABLE t_original TO t_old;
RENAME TABLE t_old TO t_original;
Existing solutions

Q & A

Thank you!

• zburivsky@pythian.com
• Twitter: @zburivsky