MySQL Schema Migrations: Exploring the Options

Santa Clara, California | April 23th – 25th, 2018
Brian Cain
Gillian Gunson
Mark Filipi
About Mark

- Senior Database Administrator at SurveyMonkey
- Formerly PalominoDB, Garmin
- Based in Kansas City
About Brian

- Senior Database Engineer at SurveyMonkey
- Formerly Dropbox, PalominoDB, Zappos, EMusic
- Based in Seattle
About Gillian

- Senior Database Infrastructure Engineer at GitHub
- Formerly Okta, PalominoDB, Oracle, Disney
- Based in Vancouver, Canada
Lab Setup
Logging in

- Pick an EC2 instance from https://bit.ly/2HHonOp
- curl -L https://bit.ly/2qHm2f3 -o aws_key
- chmod 400 aws_key
- ssh -i aws_key ubuntu@[your EC2]
EC2 Instance Details: dbdeployer

DBdeployer [https://github.com/datacharmer/dbdeployer](https://github.com/datacharmer/dbdeployer)
- Golang rewrite of MySQL Sandbox

$ ls

<table>
<thead>
<tr>
<th>dbdeployer</th>
<th>master</th>
<th>pt-osc</th>
<th>repl1</th>
<th>scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>child.sql</td>
<td>ghost</td>
<td>master.cnf</td>
<td>replica1.cnf</td>
<td>sandboxes</td>
</tr>
</tbody>
</table>
EC2 Instance Details: Employees DB

Employees Sample Database
- Old test database and data from MySQL docs
- Now hosted on GitHub as test_db
  https://github.com/datacharmer/test_db

We are using 2 tables from it
- employees
- division

Auto-generated data has been added to make it larger
Getting the lab environment running

$ dbdeployer global start

# Running "start_all" on rsandbox_5_7_21
# executing 'start' on
/home/ubuntu/sandboxes/rsandbox_5_7_21
executing 'start' on master
. sandbox server started
executing 'start' on slave 1
. sandbox server started
Check the status

$ dbdeployer global status

# Running "status_all" on rsandbox_5_7_21
REPLICATION  /home/ubuntu/sandboxes/rsandbox_5_7_21
master : master on  -  port 16743 (16743)
nodel1 : nodel1 on  -  port 16744 (16744)
Access to the MySQL instances

$ ./master

Welcome to the MySQL monitor. Commands end with ; or \g.
...
master [localhost] {msandbox} ((none)) >

$ ./repl1

...
slave1 [localhost] {msandbox} ((none)) >
How to recover

```
$ recovery/reset_all.sh

[ abbreviated ]
Sandbox rsandbox_5_7_21 unlocked
Running /home/ubuntu/sandboxes/rsandbox_5_7_21/stop_all
Running rm -rf /home/ubuntu/sandboxes/rsandbox_5_7_21
Installing and starting master
Installing and starting slave 1
run 'dbdeployer usage multiple' for basic instructions'
Sandbox rsandbox_5_7_21 locked
```
Database Schema Migrations
Today’s topics

- Schema Changes
- In-place alters
- pt-online-schema-change
- gh-ost
- Scenarios
Schema Changes

- What is a schema change?
  - ALTER TABLE
  - ADD/DROP/MODIFY COLUMN
  - ADD/DROP INDEX
  - TABLE COMPRESSION
Reasons for changes

- Performance issues - indexes
- App feature changes - add/remove columns
- Disk space considerations - table compression, fragmentation, partitioning
- Continuous deployment - automated deploys
Resistance to change

- Long-running alters
- Blocking alters
- Resource-intensive alters
- Historically, very intrusive
Methods for maintenance

- Built-in options
- Promotion strategies
- External tools
- Schedule system downtime
Built-in Options
Version considerations

- Examples are given for Percona Server 5.7.21
- Some features are unavailable in older versions
MyISAM

- Your worst option
- Table-level locking
- Limited options to reduce impact
Promotion Strategies

- Through careful planning, alter tables on replicas
- Promote replica to master, alter master
- Use extreme caution with binlogging and promotions
  - `sql_log_bin=0`
- Can’t use when in some scenarios. I.e. reordering columns with RBR
InnoDB

- Your best option for OLTP
- Starting 5.6 Online DDL
- Specify alter algorithm
- Specify locking options
InnoDB Online DDL Options

- LOCK and ALGORITHM
- Both are optional!
- Engine chooses least restrictive for DML
ALTER TABLE employees ADD INDEX ind_foo (birth_date), ALGORITHM=INPLACE, LOCK=NONE;

- **ALGORITHM = COPY**
  - Forces a full table copy
- **ALGORITHM = INPLACE**
  - Does not copy data file
- Fails if method is not available
- Cannot use In-place for 5.5 tables.
InnoDB LOCK

ALTER TABLE employees ADD INDEX ind_foo (birth_date), ALGORITHM=INPLACE, LOCK=NONE;

- **LOCK ≡ EXCLUSIVE**
  - No reads or writes
- **LOCK ≡ SHARED**
  - Reads only
- **LOCK ≡ NONE**
  - Reads and writes
- Fails if method is not available
- Cannot use In-place for 5.5 tables.
InnoDB Online DDL Options

- The table does not need to be rebuilt if the stored pages aren’t changing
  - Changing the name of a column or index
  - Changing the auto-increment value or column default
  - Dropping an index
  - Adding an ENUM value to the end of the list
  - Adding or removing foreign key constraints
Innodb Online DDL Options

- **Online without copy - Change a default value**
  Query OK, 0 rows affected (0.07 sec)

- **Online with background operations - Changing an index**
  Query OK, 0 rows affected (21.42 sec)

- **Copying all rows - Change data types**
  Query OK, 1671168 rows affected (1 min 35.54 sec)
Built-in Demo
Demo Table

CREATE TABLE `employees` (  
`emp_no` int(11) NOT NULL,  
`birth_date` datetime NOT NULL,  
`first_name` varchar(50) NOT NULL,  
`last_name` varchar(50) NOT NULL,  
`hire_date` datetime NOT NULL,  
PRIMARY KEY (`emp_no`),  
KEY `idx_first_name` (`first_name`),  
KEY `idx_last_name` (`last_name`) ) ENGINE=InnoDB DEFAULT CHARSET=utf8
Add column at end of table

- ALTER TABLE employees.employees ADD COLUMN state ENUM('FTE', 'CONTRACT') NOT NULL;

```bash
ls -lh sandboxes/rsandbox_5_7_21/master/data/employees/employees.*
```
Modify column definition

- ALTER TABLE employees.employees MODIFY COLUMN state ENUM('FTE', 'CONTRACT', 'SEPARATED') NOT NULL;
Change data type

- ALTER TABLE employees.employees MODIFY COLUMN state INT, ALGORITHM = INPLACE;
Add column mid-table

- ALTER TABLE employees.employees ADD COLUMN city VARCHAR(50) AFTER last_name;
Add index

- ALTER TABLE employees.employees ADD INDEX idx_state(`state`);
Drop index

- ALTER TABLE employees.employees DROP INDEX idx_state;
Compress table

- ALTER TABLE employees.employees KEY_BLOCK_SIZE=4;
  - Expensive operation
Drop column

- ALTER TABLE employees.employees DROP COLUMN state;
Built-in DDL wrap-up

- Online DDL - Straightforward approach to simple changes with built-in tools
- Nothing new to install or learn
- Only on newer versions
- Replica promotions - Careful planning allows for more complex changes. Overcome online limitations.
pt-online-schema-change

Details and demos
pt-online-schema-change (pt-osc) agenda

- Overview
- Limitations
- A new demo table
- Process
- Safety measures
- Various items of note
Overview

- Percona Toolkit was forked from Maatkit and Aspersa in June 2011
- Online schema changes first implemented by Shlomi Noach in oak-online-alter-table
- Facebook later built another version OnlineSchemaChange.php
- pt-osc is a hybrid of both
- Perl
- Open source
- Our lab uses 3.0.8
Limitations

- Table needs to have a PRIMARY KEY or UNIQUE INDEX for copying data and building triggers
- Be careful adding a unique index with pt-osc
  - Copy uses INSERT IGNORE that can silently lose data on a uniqueness conflict
- New NOT NULL columns should specify a default
  - null-to-not-null flag allows MySQL to try to choose the data type default
- Cannot alter MyISAM tables
A new demo table

- `.master < child.sql`

```sql
CREATE TABLE `division` (  
  `division_name` varchar(50) NOT NULL,  
  `emp_no` int(11) NOT NULL,  
  PRIMARY KEY (`division_name`,`emp_no`),  
  KEY `idx_emp_no` (`emp_no`),  
  CONSTRAINT `fk_emp_no` FOREIGN KEY (`emp_no`) REFERENCES `employees` (`emp_no`)  
) ENGINE=InnoDB
```
And don’t forget the employees table

CREATE TABLE `employees` (  
  `emp_no` int(11) NOT NULL,  
  `birth_date` datetime NOT NULL,  
  `first_name` varchar(50) NOT NULL,  
  `last_name` varchar(50) NOT NULL,  
  `hire_date` datetime NOT NULL,  
  PRIMARY KEY (`emp_no`),  
  KEY `idx_first_name` (`first_name`),  
  KEY `idx_last_name` (`last_name`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8
Process

1. Create and alter the new table
2. Add insert, update, and delete triggers to the old table
3. Copy data in “chunks” from the old table
   a. At the end of each chunk check values and status
   b. Continue, pause, or abort based on the evaluations
4. Swap old and new tables
5. Drop the old table
6. Drop triggers
Process details

./pt-osc --execute --alter "ADD COLUMN state ENUM('FTE', 'CONTRACT', 'SEPARATED') NOT NULL"

[ Setup ]
SHOW VARIABLES LIKE 'innodb\_lock_wait_timeout';
SET SESSION innodb_lock_wait_timeout=1;
SHOW VARIABLES LIKE 'lock\_wait_timeout';
SET SESSION lock_wait_timeout=60;
SHOW GRANTS FOR CURRENT_USER();
Process details cont.

Found 1 slaves:
ip-172-31-18-119 -> localhost:16744
Will check slave lag on:
ip-172-31-18-119 -> localhost:16744
Process details cont.

[ Setup cont ]

SHOW GLOBAL STATUS LIKE 'Threads_running';
SHOW TABLES FROM `employees` LIKE 'employees';
SHOW TRIGGERS FROM `employees` LIKE 'employees';
SHOW CREATE TABLE `employees`.`employees`;
EXPLAIN SELECT * FROM `employees`.`employees` WHERE 1=1;
SELECT table_schema, table_name
FROM information_schema.key_column_usage WHERE
referenced_table_schema='employees'
AND referenced_table_schema='employees';
Process details cont.

Operation, tries, wait:
- analyze_table, 10, 1
- copy_rows, 10, 0.25
- create_triggers, 10, 1
- drop_triggers, 10, 1
- swap_tables, 10, 1
- update_foreign_keys, 10, 1
Process details cont.

Altering `employees`.`employees`...

Creating new table...

```sql
CREATE TABLE `employees`.`_employees_new` (  `emp_no` int(11) NOT NULL,  `birth_date` datetime NOT NULL,  `first_name` varchar(50) NOT NULL,  `last_name` varchar(50) NOT NULL,  `hire_date` datetime NOT NULL,  PRIMARY KEY (`emp_no`),  KEY `idx_first_name` (`first_name`),  KEY `idx_last_name` (`last_name`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

Created new table employees._employees_new OK.
Process details cont.

Altering new table...

```
ALTER TABLE `employees`._employees_new` ADD COLUMN state ENUM('FTE', 'CONTRACT', 'SEPARATED') NOT NULL
```

Altered `employees`._employees_new` OK.
2018-04-08T19:54:35 Creating triggers...

CREATE TRIGGER `pt_osc_employees_employees_del` AFTER DELETE ON `employees`.`employees` FOR EACH ROW DELETE IGNORE FROM `employees`.`_employees_new` WHERE `employees`.`_employees_new`.`emp_no` <=> OLD.`emp_no`

CREATE TRIGGER `pt_osc_employees_employees_upd` AFTER UPDATE ON `employees`.`employees` FOR EACH ROW REPLACE INTO `employees`.`_employees_new` (`emp_no`, `birth_date`, `first_name`, `last_name`, `hire_date`) VALUES (NEW.`emp_no`, NEW.`birth_date`, NEW.`first_name`, NEW.`last_name`, NEW.`hire_date`) VALUES (NEW.`emp_no`, NEW.`birth_date`, NEW.`first_name`, NEW.`last_name`, NEW.`hire_date`)

CREATE TRIGGER `pt_osc_employees_employees_ins` AFTER INSERT ON `employees`.`employees` FOR EACH ROW REPLACE INTO `employees`.`_employees_new` (`emp_no`, `birth_date`, `first_name`, `last_name`, `hire_date`) VALUES (NEW.`emp_no`, NEW.`birth_date`, NEW.`first_name`, NEW.`last_name`, NEW.`hire_date`)

2018-04-08T19:54:35 Created triggers OK.
2018-04-08T19:54:35 Copying approximately 32280 rows...

```
INSERT LOW_PRIORITY IGNORE INTO `employees`.`_employees_new` (`emp_no`, `birth_date`, `first_name`, `last_name`, `hire_date`) SELECT `emp_no`, `birth_date`, `first_name`, `last_name`, `hire_date` FROM `employees`.`employees` FORCE INDEX(`PRIMARY`) WHERE ((`emp_no` >= ?)) AND ((`emp_no` <= ?)) LOCK IN SHARE MODE /*pt-online-schema-change 18610 copy nibble*/
```

```
SELECT /*!40001 SQL_NO_CACHE */ `emp_no` FROM `employees`.`employees` FORCE INDEX(`PRIMARY`) WHERE ((`emp_no` >= ?)) ORDER BY `emp_no` LIMIT ?, 2 /*next chunk boundary*/
```

2018-04-08T19:54:36 Copied rows OK.

2018-04-08T19:54:36 Max rows for the rebuild_constraints method: 92946

Determining the method to update foreign keys...

2018-04-08T19:54:36 `employees`.`division`: 1 rows; can use rebuild_constraints
2018-04-08T19:54:36 Analyzing new table...
ANALYZE TABLE `employees`.`_employees_new` /* pt-online-schema-change */

2018-04-08T19:54:36 Swapping tables...
RENAME TABLE `employees`.`employees` TO `employees`.`_employees_old`, `employees`.`_employees_new` TO `employees`.`employees`

2018-04-08T19:54:36 Swapped original and new tables OK.

2018-04-08T19:54:36 Rebuilding foreign key constraints...
ALTER TABLE `employees`.`division` DROP FOREIGN KEY `fk_emp_no`, ADD CONSTRAINT `_fk_emp_no` FOREIGN KEY (`emp_no`) REFERENCES `employees`.`employees` (`emp_no`)

2018-04-08T19:54:36 Rebuilt foreign key constraints OK.
Process details cont.

2018-04-08T19:54:36 Dropping old table...
DROP TABLE IF EXISTS `_employees_old` /* generated by server */

2018-04-08T19:54:36 Dropped old table `employees`.`_employees_old` OK.

2018-04-08T19:54:36 Dropping triggers...
DROP TRIGGER IF EXISTS `employees`.`pt_osc_employees_employees_del`
DROP TRIGGER IF EXISTS `employees`.`pt_osc_employees_employees_upd`
DROP TRIGGER IF EXISTS `employees`.`pt_osc_employees_employees_ins`

2018-04-08T19:54:36 Dropped triggers OK.
Successfully altered `employees`.`employees`. 
Safety measures - Replication

- **Pause for replication lag**
  - max-lag (1 second)
  - check-slave-lag (specific DSN)
  - recursion-method
    - processlist (show processlist)
    - hosts (show slave hosts)
    - dsn=[dsn_table]
    - none

- **Unless overridden replication filters prevent pt-osc from running**
  - Replica may be missing a table or schema
### dsns table

CREATE TABLE `dsns` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `parent_id` int(11) DEFAULT NULL,
  `dsn` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB

<table>
<thead>
<tr>
<th>id</th>
<th>parent_id</th>
<th>dsn</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NULL</td>
<td>h=localhost,P=16744,S=/tmp/mysql_sandbox16744.sock</td>
</tr>
</tbody>
</table>
.\pt-osc and demo

pt-online-schema-change
P=16743,S=/tmp/mysql_sandbox16743.sock,u=msandbox,p=msandbox,D=employees,t=employees
--alter-foreign-keys-method=auto
--recursion-method='dsn=P=16743,S=/tmp/mysql_sandbox16743.sock,u=msandbox,p=msandbox,D=employees,t=dsns'
"$@

● Demo
  ○ ADD COLUMN state ENUM('FTE', 'CONTRACT', 'SEPARATED') NOT NULL
Safety measures - Status variables

- Pause or abort based on global status
  - max-load (default Threads_running=25)
  - critical-load (default Threads_running=50)

- Demo (hitting critical and max load)
  - ADD COLUMN separation_date datetime

--max-load Threads_connected=3 --critical-load Threads_connected=6
Safety measures - tries

- Retry various operations and wait between tries

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>TRIES</th>
<th>WAIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_triggers</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>drop_triggers</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>copy_rows</td>
<td>10</td>
<td>0.25</td>
</tr>
<tr>
<td>swap_tables</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>update_foreign_keys</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>analyze_table</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

- Demo (open transactions, default lock_wait_timeout=60)
  - DROP COLUMN state, DROP COLUMN separation_date

    --tries create_triggers:15:2,drop_triggers:1:1 --set-vars lock_wait_timeout=1
analyze-before-swap

ANALYZE TABLE `employees`._employees_new` /* pt-online-schema-change */

● Run by default
● Build optimizer stats before queried
Examining the child table

CREATE TABLE `division` (  `division_name` varchar(50) NOT NULL,  `emp_no` int(11) NOT NULL,  PRIMARY KEY (`division_name`,`emp_no`),  KEY `idx_emp_no` (`emp_no`),  CONSTRAINT `fk_emp_no` FOREIGN KEY (`emp_no`) REFERENCES `employees` (`emp_no`) ) ENGINE=InnoDB

- alter-foreign-keys-method
  - rebuild_constraints - alter the child table
  - drop_swap - drop the old table
  - auto - pick rebuild_constraints unless the child table is “large”
  - none - the child foreign keys stop working
Support for multiple triggers in 5.7

- If a table already has triggers, pt-osc cannot create its triggers unless on 5.7.2+.
- preserve-triggers
  - copy existing triggers to the new table before copying rows
  - drop them before copying
  - re-apply them at the end
Multi-column key caution

- Primary or unique keys made up of 4 or more columns used as the chunk index can cause inadvertent large scans
- chunk-index-columns
  - pick the first N leftmost columns of the index
- chunk-index
  - force a different unique key with less columns

```
INSERT LOW_PRIORITY IGNORE INTO `employees`._employees_new (`emp_no`, `birth_date`, `first_name`, `last_name`, `hire_date`) SELECT `emp_no`, `birth_date`, `first_name`, `last_name`, `hire_date` FROM `employees`.employees `FORCE INDEX(`PRIMARY`) WHERE ((`emp_no` >= ?)) AND ((`emp_no` <= ?)) LOCK IN SHARE MODE /*pt-online-schema-change 18610 copy nibble*/

SELECT /*!40001 SQL_NO_CACHE */ `emp_no` FROM `employees`.employees `FORCE INDEX(`PRIMARY`) WHERE ((`emp_no` >= ?)) ORDER BY `emp_no` LIMIT ?, 2 /*next chunk boundary*/
```
Abundance of caution

For a full scale dry-run, make the last steps manual

- no-drop-new-table
- no-swap-tables
- no-drop-triggers

If all is well, swap the old and new tables, drop the old table and the triggers

Otherwise, drop the new table and triggers
For the adventurous

- You can write plugins in Perl to interact with various hooks in pt-osc
- Gives more control over debugging or how to define functions

`init`
`before_create_new_table`
`after_create_new_table`
`before_alter_new_table`
`after_alter_new_table`
`before_create_triggers`
`after_create_triggers`
`before_copy_rows`
`after_copy_rows`

`before_swap_tables`
`after_swap_tables`
`before_update_foreign_keys`
`after_update_foreign_keys`
`before_drop_old_table`
`after_drop_old_table`
`before_drop_triggers`
`before_exit`
`get_slave_lag`
gh-öst: GitHub Online Schema Transmogrifier

Tutorial and Lab
gh-ost Agenda

- History/Overview
- Restrictions and Limitations
- Process
- Demo
- Features
History

- **Written by the Database Infrastructure team at GitHub in 2016**
  - Watch the Shlomi Noach talk on youtube

- **GitHub MySQL**
  - Required high write throughput, low replica lag
  - Replicas used for reads
  - Multiple table migrations per day

- **Used pt-online-schema-change for years**
History (cont.)

- **Problems scaled along with traffic**
  - triggers doubled the writes
  - high load could cause MySQL to freeze
  - significant replication lag on functioning migrations

- **Results**
  - avoided running migrations in heaviest traffic
  - migrations had to be babysat
  - some tables were impossible to migrate (or the failover method was used)
  - some [github.com](http://github.com) outages were from migrations
gh-ost: GitHub Online Schema Transmogrifier [https://github.com/github/gh-ost](https://github.com/github/gh-ost)

- Open source
- Written in go
gh-ost Features

- Using binary logs instead of triggers
  - Offload work from MySQL master to code
- Designed to be gentle on the master and replication
  - Minimal locking
  - Pausing removes all* activity from master
  - Built-in sub-second throttling (via gh-ost changelog table)
- Interactive commands
  - Pause, unpause, change settings during process
gh-ost Features cont.

- Delay the final table swap until ready
  - Avoid table locking during busy periods
  - Migration completion can be scheduled

- Can be tested on replicas
Requirements

1. **Primary or unique not null key**
   a. Key can be nullable so long as no null values
   b. Key must remain the same on both tables

2. **binlog_format=ROW, binlog_row_image=FULL**
   a. Must be set on the host used for binary log parsing
   b. Can be set on a replica
   c. gh-ost can set this on the host automatically

3. **log-slave-updates if using a replica for binary logs**
Limitations

Doesn’t allow:

- Foreign keys, triggers
- Virtual columns
- POINT, JSON (JSON okay except in primary key)
- Dropping the only primary or unique not null key
- Duplicate tables with the same name (e.g. foo, Foo)
- Master-master active/active replication
Process

**ghost** migration:
- creates **ghost** table on migrated server
- alters **ghost** table
- hooks up as a MySQL replica, streams binary log events
- interchangeably:
  - applies events on **ghost** table
  - copies rows from original table onto **ghost** table
- cut-over

Preferred setup:
- connects to replica
- inspects table structure, table dimensions on replica
- hooks as replica onto replica
- apply all changes on master
- writes internal & heartbeat events onto master, expects them on replica
Process: Operation Modes

gh-ost operation modes

a. connect to replica
b. connect to master
c. migrate/test on replica
Tables Created

- Ghost table is _table_gho
- Changelog table is _table_ghc
  - Sub-second heartbeats and status info written here
insert /* gh-ost */ into `github_production`._issue_events_ghc_
(id, hint, value)
values (NULLIF(1, 0), "heartbeat", "2018-03-28T20:48:55.07421166-07:00")
on duplicate key update last_update=NOW(), value=VALUES(value)

select hint, value from `github_production`._issue_events_ghc_
where hint = 'heartbeat' and id <= 255
Queries run: ghost table

```sql
insert /* gh-ost `github_production`.`issue_events` */ ignore into `github_production`.`_issue_events_ghost`(`id`, ...)  
  (select `id`, ... from `github_production`.`issue_events` force index (`PRIMARY`)  
   where (((`id` > _binary'1537984329')) and ((`id` < _binary'1537984429') or ((`id` = _binary'1537984429'))) lock in share mode)

replace /* gh-ost `github_production`.`_issue_events_gho` */ into `github_production`.`_issue_events_gho`(`id`, ...)  
values (1546246663, ..)

delete /* gh-ost `github_production`.`_issue_events_gho` */  
  from `github_production`.`_issue_events_gho`  
  where ((`id` = 1547038570))
```
Cut-over/Table swap

- Two step “atomic” swap
- Algorithm involves LOCK TABLE, a sentry table and swap via RENAME TABLE
  - There are blog posts and a doc page
- Backs off gracefully if it times out or a connection dies
- Can be postponed with a flag
Demo
Demo Setup: `ghost` script contents

```
gh-ost \
  --max-load=Threads_connected=10 \
  --critical-load=Threads_connected=100 \
  --chunk-size=100 --max-lag-millis=10000 \
  --panic-flag-file=/tmp/ghost.panic.flag \
  --host="127.0.0.1" --port="16744" \n  --throttle-control-replicas="127.0.0.1:16744" \n  --database="employees" \n  --table="employees" \n  ... 
  "$@
```
Demo Setup: Traffic generation

$ scripts/gen_traffic.sh

- Creates fake insert/update/select/delete traffic on the table on the master
- Writes to activity.log
- To kill the traffic run (and delete ~30 seconds after)

$ touch /tmp/kill.txt
Demo Setup

$ ./ghost --alter='engine=innodb' --execute [--anything-else]

- Without `--execute` it runs as a dry-run
  - gh-ost changelog and ghost tables are created/altered/dropped
  - Verification of migration syntax, permissions
Output: go-mysql

2018/04/23 11:10:25 binlogsyncer.go:79: [info] create BinlogSyncer with config {99999 mysql 127.0.0.1 16744 msandbox false false false <nil>}

2018/04/23 11:10:25 binlogsyncer.go:246: [info] begin to sync binlog from position (mysql-bin.000001, 1196518)

This is from the go-mysql replication library
Error messages can usually be ignored:

2018/04/23 11:10:39 binlogstreamer.go:47: [error] close sync with err: sync is been closing...
# Migrating `employees`.`employees`; Ghost table is `employees`.`_employees_gho`
executing on ip-172-31-18-119
# Migration started at Mon Apr 23 16:48:34 +0000 2018
# chunk-size: 100; max-lag-millis: 10000ms; dml-batch-size: 10; max-load:
Threads_connected=10; critical-load: Threads_connected=100; nice-ratio: 0.000000
# throttle-additional-flag-file: /tmp/gh-ost.throttle
# throttle-control-replicas count: 1
# panic-flag-file: /tmp/ghost.panic.flag
# Serving on unix socket: /tmp/gh-ost.employees.employees.sock
Output: copying

Copy: 0/33021 0.0%; Applied: 0; Backlog: 0/1000; Time: 0s(total), 0s(copy); streamer: mysql-bin.000001:5590611; State: migrating; ETA: N/A

...}

Copy: 28700/33021 86.9%; Applied: 36; **Backlog: 5/1000**; Time: 8s(total), 8s(copy); streamer: mysql-bin.000001:6651875; **State: throttled, max-load Threads_connected=10 >= 10**; ETA: 1s

...

Copy: 32746/32746 100.0%; Applied: 44; Backlog: 0/1000; Time: 9s(total), 8s(copy); streamer: mysql-bin.000001:6777375; State: migrating; **ETA: due**
Alter #1

$ ./ghost --alter="ADD COLUMN state ENUM('FTE', 'CONTRACT', 'SEPARATED') NOT NULL" --execute
gh-ost Features
Flag files

--postpone-cut-over-flag-file=/tmp/ghost.postpone.flag
  - gh-ost creates the file and doesn’t cut-over until it’s deleted

--throttle-additional-flag-file=/tmp/gh-ost.throttle
--panic-flag-file: /tmp/ghost.panic.flag
  -gh-ost checks if the file exists and acts on it
Interactive commands

Socket created as /tmp/gh-ost.schemaname.tablename.sock

$ echo sup | nc -U /tmp/gh-ost.employees.employees.sock
Copy: 32225/32225 100.0%; Applied: 0; Backlog: 0/1000; Time: 1m57s(total), 5s(copy); streamer: mysql-bin.000001:2613766; State: postponing cut-over; ETA: due

$ echo "chunk-size=?" | nc -U /tmp/gh-ost.employees.employees.sock
100

$ echo throttle | nc -U /tmp/gh-ost.employees.employees.sock
Alter #2: postpone flag

$ ./ghost --alter="ENGINE=INNODB" --execute --postpone-cutover-flag-file=/tmp/ghost.postpone.flag
Features: Hooks

- External scripts run at certain points in migration
- Can be used for notifications, cleanup
- Gh-ost sets up environment variables with migration info

gh-ost-on-startup*
gh-ost-on-validated*
gh-ost-on-before-row-copy*
gh-ost-on-status*
gh-ost-on-before-cut-over*
gh-ost-on-failure*
GitHub chatops

hubot APP 9:28 PM

GitHub chatops

hubot APP 9:45 PM

GitHub chatops

hubot APP 9:54 PM

GitHub chatops

ghunson 9:56 PM

GitHub chatops

ghunson 9:56 PM

GitHub chatops

ghunson 9:56 PM

GitHub chatops
Features: Testing on Replica

$ cat ghost-migration-users-1523536953.res

gh-ost testing users
gh-ost execution result: 0
count(*)
37108587
count(*)
37108587
checksums: 613d7dc9d4143e48da4b6f619fe35444 -, 613d7dc9d4143e48da4b6f619fe35444 -
users test result: success
Test result for users is: success
gh-ost done testing users
---
Migration Scenarios
Scenarios

#1
Site is down
Everything’s on fire

#2
Scheduled downtime for other reasons
Scenarios

#3
Adding index

#4
ALTER TABLE tbl ADD COLUMN foo... AFTER bar
Scenarios

#5
No replication

#6
Really low on disk space
Scenarios

#7
binlog_format=STATEMENT
Unclear data consistency on replicas

#8
FOREIGN KEYS
TRIGGERS
Scenarios

#9
No downtime but as fast as possible

#10
High load on master
Heavy writes
Scenarios

#11
Parallel migrations across thousands of schemas

#12
Table file heavily fragmented
Rate My Session

**Introducing gh-ost: triggerless, painless, trusted online schema migrations**

- Time: 11:20 → 12:10
- Location: Matterhorn 2

**Rate & Review**

- Tap a star to rate
- Feedback (optional)
- Anonymously
- Submit
Thank You!

Go forth and prosper