



PERCONA
Performance Consulting Experts

Make Your Life Easier with Maatkit

Ryan Lowe

San Francisco MySQL Meetup

About the Speaker

- Senior Consultant at Percona
 - Performance
- Background
 - Telecommunications
 - Health Care
 - Defense
- F/OSS
 - OmniSQL
 - MyDBDoc
 - check-mysql-all
 - partition_balancer
 - MMM

What Is Maatkit?

- A collection of Perl scripts for DBAs
 - Current Count: 24?
- Automate manual tasks
- Fill in MySQL's missing features
- Get Maatkit tools quickly:
 - `wget http://www.maatkit.org/get/<toolname>`
 - `wget http://www.maatkit.org/trunk/<toolname>`
- Documentation:
 - `perldoc`
 - `http://www.maatkit.org/doc/<toolname>.html`

General Knowledge

- mk-*
- DSN
 - h=localhost,D=test,t=source
 - They inherit from previous DSNs
- CLI
 - --test/--execute, --pid/--daemonize, --sleep/--sleep-coef
- %> export MKDEBUG=[0|1]
- #maatkit on Freenode
- maatkit-discuss on Google Groups
- Monthly release cycle

Today's Agenda

- How I use Maatkit
 - Audits
- Questions?
- What Lies Ahead?

Maatkit Scripts

mk-archiver

mk-audit

mk-checksum-filter

mk-deadlock-logger

mk-duplicate-key-checker

mk-fifo-split

mk-find

mk-heartbeat

mk-log-player

mk-log-server

mk-parallel-dump

mk-parallel-restore

mk-profile-compact

mk-query-digest

mk-query-profiler

mk-show-grants

mk-slave-delay

mk-slave-find

mk-slave-move

mk-slave-prefetch

mk-slave-restart

mk-table-checksum

mk-table-sync

mk-visual-explain

Maatkit Scripts

mk-archiver

mk-audit

~~mk-checksum-filter~~

mk-deadlock-logger

mk-duplicate-key-checker

~~mk-fifo-split~~

mk-find

mk-heartbeat

~~mk-log-player~~

~~mk-log-server~~

mk-parallel-dump

mk-parallel-restore

~~mk-profile-compact~~

mk-query-digest

~~mk-query-profiler~~

~~mk-show-grants~~

~~mk-slave-delay~~

mk-slave-find

mk-slave-move

~~mk-slave-prefetch~~

mk-slave-restart

mk-table-checksum

mk-table-sync

mk-visual-explain



mk-audit

mk-audit

- Overall System Information
- High-Level MySQL Information
- Replaces a whole host of commands
- Provides recommendations
 - BUT BE WARY OF THEM
- Helps guide me where to look next
- Lengthy output

Server Specs (1)

Server Specs

OS: linux Red Hat Enterprise Linux Server release 5.2 (Tik Architecture: 64-bit

CPU: Intel(R) Xeon(R) CPU E5430 @ 2.66GHz Architecture: 64-bit

Speed: MHz: 2666 2666 2666 2666 2666 2666 2666 2666

Cache: 6144 KB

Count: 2

Cores: 8

Memory: used 15.59G of 15.68G total (85.84M free)

Buffers: 470.28M

Cached: 4.70G

Shared: 0

Server Specs (2)

Storage:

1 RAID controllers detected: aacraid

Controller Model : Adaptec 3805

Controller Status : Optimal

Installed memory : 128 MB

Temperature : 49 C/ 120 F (Normal)

Defunct disk drive count : 0

Write-cache mode : Enabled (write-back)

Write-cache setting : Enabled (write-back) when protected by battery

Controller Battery Information -----

Status : Optimal

Over temperature : No

Capacity remaining : 100 percent

Time remaining (at current draw) : 3 days, 1 hours, 52 minutes

Server Specs (3)

LVM volume groups: VG #PV #LV #SN Attr VSize VFree

data 1 2 0 wz--n- 558.00G 258.00G

Filesystem Type Size Used Avail Use% Mounted on

/dev/sda2 ext3 9.7G 7.5G 1.8G 82% /

/dev/sda5 ext3 55G 7.1G 45G 14% /var

/dev/sda1 ext3 99M 17M 77M 18% /boot

tmpfs tmpfs 7.9G 0 7.9G 0% /dev/shm

/dev/mapper/data-backup

ext3 50G 18G 32G 36% /raid10/backups

/dev/mapper/data-mysql

ext3 247G 239G 8.1G 97% /raid10/mysql

libc: 2.5

Compiled by: GNU CC version 4.1.2 20071124 (Red Hat 4.1.2-41).

Threading: NPTL

GNU libpthread version: NPTL 2.5

Server Specs (4)

PROBLEMS

- Not default value `/proc/sys/net/ipv4/icmp_ignore_bogus_error_responses:`

`set=1`

`default=0`

- Not default value `/proc/sys/net/ipv4/icmp_ratelimit:`

`set=1000`

`default=100`

- Not default value `/proc/sys/net/ipv4/tcp_rmem:`

`set=4096_87380_4194304`

`default=8192_87380_174760`

- Not default value `/proc/sys/net/ipv4/tcp_wmem:`

`set=4096_16384_4194304`

`default=4096_16384_131072`

MySQL Instance (1)

MySQL Instance 1

Version: 5.0.75-percona-highperf-b12-log

Architecture: 64-bit

Uptime: 2+14:33:54

ps vals: user mysql cpu% 5.7 rss 9.78G vsz 10.31G syslog: No

Bin: /usr/sbin/mysqld

Data dir: /raid10/mysql/

PID file: /raid10/mysql/db1.pid

Socket: /var/lib/mysql/mysql.sock

Port: 3306

Log locations:

Error: /var/lib/mysql/mysqld.log

Relay:

Slow: 1000000 /var/lib/mysql/mysql-slow.log

Config file location:

MySQL Instance (2)

SCHEMA

#DATABASES	#TABLES	#ROWS	#INDEXES	SIZE DATA	SIZE INDEXES
14	1989	1.27G	3563	101.64G	66.87G

Key buffer size : 1.00G

InnoDB buffer pool size: 8.00G

Top 5 largest databases:

DATABASE	SIZE DATA
db1	101.56G
db2	43.46M
db3	35.52M
db4	2.12M
test	1.84M
Remaining 8	2.20M (281.16k average)

MySQL Instance (3)

Top 5 largest tables:

DB.TBL	SIZE DATA	SIZE INDEX	#ROWS	ENGINE
db1.tbl1	21.76G	0	8.57M	InnoDB
db1.tbl2	12.96G	4.93G	130.26M	InnoDB
db1.tbl3	7.50G	8.33G	134.37M	InnoDB
db1.tbl4	7.42G	6.00G	110.79M	InnoDB
db1.tbl5	6.66G	7.75G	98.09M	InnoDB
Remaining 1984	45.34G (23.40M average)			

Engines:

ENGINE	SIZE DATA	SIZE INDEX	#TABLES	#INDEXES
MyISAM	8.27G	7.55G	1263	2194
InnoDB	93.37G	59.32G	674	1278

Triggers, Routines, Events:

DATABASE	TYPE	COUNT
----------	------	-------

No triggers, routines, or events

MySQL Instance (4)

PROBLEMS

Out of sync system variables (online value differs from config value):

VARIABLE	ONLINE VALUE	CONFIG VALUE
pid_file	/raid10/mysql/db1.pid	/raid10/mysql/db01.pid
innodb_ibuf_max_size	4294967296	9223372036854775807
long_query_time	1000000.000000	0.000000
read_only	OFF	TRUE
replicate_same_server		FALSE
skip_slave_start		TRUE
open_files_limit	25000	20000

Things to Note:

- InnoDB: buffer pool too small
- max_connections has been modified from its default (100): 5000
- InnoDB: zero free buffer pool pages

MySQL Instance (5)

Aggregated PROCESSLIST

FIELD	VALUE	COUNT	TOTAL TIME (s)
db			
	db1	1	0
	NULL	3	61751
user			
	writer	4	48
	mmm	1	20418
	system user	2	41333
host			
	10.0.0.1	4	48
state			
	waiting for master to sen	1	20696
command			
	connect	2	41333

mk-duplicate-key-checker

mk-duplicate-key-checker

```
# #####  
# db1.wp_comments  
#####  
# comment_approved is a left-prefix of comment_approved_date_gmt  
# Column types:  
#   `comment_approved` varchar(20) NOT NULL default '1'  
#   `comment_date_gmt` datetime NOT NULL default '0000-00-00 00:00:00'  
# To remove this duplicate key, execute:  
ALTER TABLE `db1`.`wp_comments` DROP KEY `comment_approved`;  
# #####  
# Summary of keys  
#####  
# Size Duplicate Keys  0  
# Total Duplicate Keys 1  
# Total Keys          68
```



mk-archiver

mk-archiver

- Copies rows from one table to another
 - Minimal locking, minimal blocking, efficient queries
 - Source & destination can be on different servers
 - Does not do INSERT... SELECT, to avoid locking
- Archive and/or purge
 - Delete after/instead of copying (--purge)
 - Write to a file (--file) that you can LOAD DATA INFILE
- Extensible
 - Has hooks for your custom plugins
- Is a complex tool, with many options for fine control
 - Handles a lot of edge cases that could hurt your data

Simple archiving

- Before

```
$ mysql -ss -e 'select count(*) from test.source'  
8192  
$ mysql -ss -e 'select count(*) from test.dest'  
0
```

- Execute the command

```
$ mk-archiver --source h=localhost,D=test,t=source \  
--dest t=dest --where 1=1
```

- After

```
$ mysql -ss -e 'select count(*) from test.dest'  
8191  
$ mysql -ss -e 'select count(*) from test.source'  
1
```

Why is one row left?

- So the `AUTO_INCREMENT` doesn't get lost.
 - Lots of edge cases!
- This job did one row at a time, and was kind of slow.
- Can we speed it up? Of course.

```
$ mk-archiver --source h=localhost,D=test,t=source \  
  --dest t=dest --where 1=1 --commit-each --limit 100
```

Purging with a file copy

- Let's archive data to a file

```
$ mk-archiver --source h=localhost,D=test,t=source \  
  --dest t=dest --where l=1 --file '%Y-%m-%d-%D.%t'
```

- This is rather safely and carefully coded.

```
~$ tail 2009-04-18-test.source  
8182  
8183  
8184  
8185  
8186  
8187  
8188  
8189  
8190  
8191
```

Let's archive orphaned rows

- Set up a “parent” table with some missing rows

```
mysql> create table parent (p int primary key);  
mysql> insert into parent  
    > select * from source where rand() < .5;
```

- Now archive only rows that don't exist in the parent

```
$ ... --where 'NOT EXISTS(SELECT * FROM parent WHERE p=a)'
```

- Useful for archiving data with dependencies

In real life

- mk-archiver is a key tool for lots of people
- Archiving and purging are key ways to deal with data growth
- Doing it without causing replication lag or blocking the application are non-trivial
- Archiving data with complex dependencies is also hard
- Treat mk-archiver as a wrapper that eliminates a lot of code and handles the nasty edge cases for you



mk-query-digest

mk-query-digest

- Once upon a time, there were too many log-parsing tools
 - All reinventing the wheel in more or less broken ways
 - None of them combined power and ease of use
- Along came mk-log-parser (now mk-query-digest)
- Hard to believe, but it is
 - Very powerful
 - Easy to use
 - Flexible and somewhat generic

This is not a log parser

- We renamed it because it isn't a log parser
 - It still plays one on TV, though
- It is a series of filters and transformations for “query events”
 - Very much inspired by Unix pipes-and-filters data flow
- A query event can come from various places
 - Slow query logs
 - SHOW PROCESSLIST
 - The output of tcpdump

What can you do with it?

- Sniff queries from a production server and keep a passive standby's caches warm
- Determine minimum necessary privileges for a user
- And analyze server workload, of course
 - Find most expensive queries
 - Find most important tables
 - Find most active users
 - Display a timeline of changes a user performed
 - Record workload metrics in tables, for future analysis

Keep a server's caches warm

- Why on earth?
 - Faster failover. An idle standby is NOT “hot”
 - <http://tinyurl.com/warm-mysql-failover>

```
$ mk-query-digest --processlist h=localhost \  
  --execute h=some-other-host \  
  --filter '$event->{fingerprint} =~ m/^select/'
```

Discover minimal privileges

```
$ mk-query-digest --processlist h=127.0.0.1,P=2900 \  
  --timeline distill \  
  --filter '$event->{user} eq "appuser" '  
^C# Caught SIGINT.  
# #####  
# distill report  
# #####  
#      1240099409      0:00      1 SELECT mysql.user  
#      1240099430      0:00      1 SELECT mysql.db
```

Log analysis

- By default, mk-query-digest analyzes a slow log
- Groups queries into “classes”
- Tracks any desired statistic per-class
- Prints out a user-friendly report
 - All kinds of information about the class of queries: min/max/average/median/stddev/count of exec time, lock time, etc
- Knows about the Percona patches to the slow log
 - A rich source of information about your server's workload, which is not available any other way. If you aren't using a Percona server build, you should be.

The query analysis report

```

# Query 1: 0 QPS, 0x concurrency, ID 0x6BF9BDF51F671607 at byte 0 _____
# This item is included in the report because it matches --limit.
#           pct    total      min      max      avg      95%  stddev  median
# Count           100         1
# Exec time       100         1s       1s       1s       1s       1s       0       1s
# Lock time        0         0         0         0         0         0         0         0
# Users            1 msandbox
# Time range 1240099675 to 1240099675
# Query_time distribution
#  1us
#  10us
# 100us
#   1ms
#   10ms
#  100ms
#   1s #####
#  10s+
# Tables
#   SHOW TABLE STATUS FROM `mysql` LIKE 'db'\G
#   SHOW CREATE TABLE `mysql`.`db`\G
# EXPLAIN
select sleep(1) from mysql.db limit 1\G

```

Execution time Histogram

Lots of Info about execution

Copy/Paste ease for EXPLAINing queries

Historical Trending

- What if we could store the results in a database table?
- What if each subsequent run stored a new row full of stats about the class of query?
 - Wouldn't it be nice to do this as part of logrotate?
- What if we had little helper tools to show us new queries, or queries whose performance changed?

```
$ mk-query-digest /path/to/slow.log \  
  --review h=localhost,D=test,t=query_review \  
  --review-history t=query_review_history \  
  --createreview --create-review-history
```

Peek at the review table

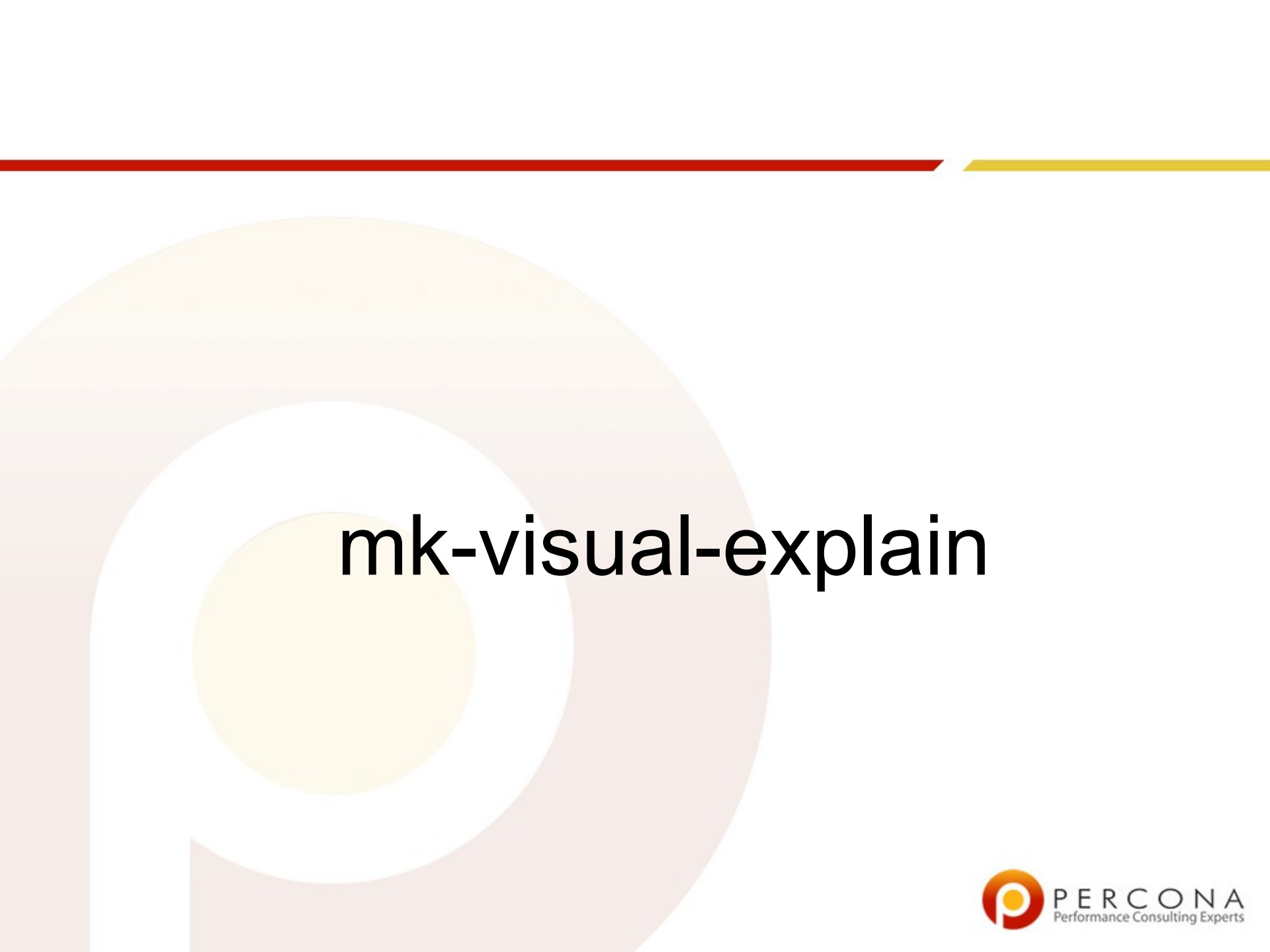
```
mysql> select * from query_review\G
***** 1. row *****
checksum: 1248277301523238490
fingerprint: replace into source select * from fill
sample: replace into source select * from fill
first_seen: 2009-04-18 16:22:58
last_seen: 2009-04-18 16:22:58
reviewed_by: NULL
reviewed_on: NULL
comments: NULL
1 row in set (0.00 sec)
```

Peek at the review history table

```
mysql> select * from query_review_history\G
***** 1. row *****
checksum: 1248277301523238490
sample: replace into source select * from fill
ts_min: 2009-04-18 16:22:58
ts_max: 2009-04-18 16:22:58
ts_cnt: 1
Query_time_sum: 1
Query_time_min: 1
Query_time_max: 1
Query_time_pct_95: 1
Query_time_stddev: 0
Query_time_median: 1
Lock_time_sum: 0
Lock_time_min: 0
Lock_time_max: 0
Lock_time_pct_95: 0
[omitted: lots more columns of statistics]
```

Implementation Practices

- 15, 30, 60-minutes per day set `long_query_time=0`
- Integrate `mk-query-digest` with `logrotate`
- Careful with `--processlist` (mutex issues in early 5.0)
- Make it a daily exercise
- Integrate it with your [JIRA|OTRS|etc] system



mk-visual-explain

mk-visual-explain

```
mysql> EXPLAIN SELECT * FROM `sakila`.`film_actor` JOIN `sakila`.`film` USING (`film_id`)\G
```

```
***** 1. row *****
```

```
id: 1
```

```
select_type: SIMPLE
```

```
table: film
```

```
type: ALL
```

```
possible_keys: PRIMARY
```

```
key: NULL
```

```
key_len: NULL
```

```
ref: NULL
```

```
rows: 994
```

```
Extra:
```

```
***** 2. row *****
```

```
id: 1
```

```
select_type: SIMPLE
```

```
table: film_actor
```

```
type: ref
```

```
possible_keys: idx_fk_film_id
```

```
key: idx_fk_film_id
```

```
key_len: 2
```

```
ref: sakila.film.film_id
```

```
rows: 1
```

```
Extra:
```

```
2 rows in set (0.00 sec)
```

mk-visual-explain

JOIN

+ - Bookmark lookup

| +- Table

| | table film_actor

| | possible_keys idx_fk_film_id

| +- Index lookup

| key film_actor->idx_fk_film_id

| possible_keys idx_fk_film_id

| key_len 2

| ref sakila.film.film_id

| rows 2

+ - Table scan

rows 952

+ - Table

table film

possible_keys PRIMARY

mk-deadlock-logger

mk-deadlock-logger

- SHOW ENGINE INNODB STATUS\G
- Stores deadlocks in a database table
- Daemonizable
- Can integrate with Nagios/Munin/Zenoss

```
mk-deadlock-logger \
```

```
--source u=user,p=pass,h=server \
```

```
--dest D=test,t=deadlocks \
```

```
-m 4h -i 30s
```

mk-table-checksum

mk-table-checksum

- MySQL replication can easily run off the rails
 - That means *different data on the slave*, not just “the slave is falling behind the master”
- There are lots of reasons
 - Foot-gun features (binlog-do-db, I'm looking at you)
 - Bad setup, poor administration
 - Non-deterministic update statements
 - Misbehaving applications or programmers
- In many cases *you will never know*
 - This is like not knowing you have heart disease
 - Once you know, you can decide whether you care

Let's break replication

- On the master:

```
mysql> create table test.test(a int not null primary key);  
mysql> insert into test(a) values(1), (2), (3);
```

- On the slave:

```
mysql> delete from test.test where a = 1;
```

- Run a checksum of the data:

```
$ mk-table-checksum -d test h=127.0.0.1,P=12345 P=12346  
--checksum  
2050879373          127.0.0.1.test.test.0  
3309541273          127.0.0.1.test.test.0
```

A different strategy

- That was a parallel checksum of both servers
- It requires locking to ensure consistency
- How to get a consistent checksum on servers whose data changes constantly, with minimal locking?
- Run the checksums through replication!

Checksums through replication

- Run the checksums on the master

```
$ mk-table-checksum -d test h=127.0.0.1,P=12345 \
  --checksum --replicate test.checksum --createreplicate
f4dbdf21          127.0.0.1.test.test.0
```

- Check the slave against the master

```
$ mk-table-checksum -d test h=127.0.0.1,P=12345 \
  --replicate test.checksum --replcheck 1
```

Differences on P=12346,h=127.0.0.1

DB	TBL	CHUNK	CNT_DIFF	CRC_DIFF	BOUNDARIES
test	test	0	-1	1	1=1

Checksum on the master

```
mysql> select * from test.checksum\G
***** 1. row *****
      db: test
      tbl: test
      chunk: 0
boundaries: 1=1
      this_crc: f4dbdf21
      this_cnt: 3
      master_crc: f4dbdf21
      master_cnt: 3
      ts: 2009-04-18 17:27:23
```

Checksum on the slave

```
mysql> select * from test.checksum\G
***** 1. row *****
      db: test
      tbl: test
      chunk: 0
boundaries: 1=1
      this_crc: 77073096
      this_cnt: 2
      master_crc: f4dbdf21
      master_cnt: 3
      ts: 2009-04-18 17:27:23
```

What else can it do?

- Checksum in chunks
- Checksum only as fast as the slave can keep up
- Get arguments from a table (per-table arguments)
- Checksum only certain columns, or ignore columns
- Compensate for floating-point differences
- Filter out certain databases, tables, storage engines
- Checksum only some portions of the data
- Resume partial jobs
- Checksum with a WHERE clause

Performance!

- Yes, “Performance Is Everything”™
- MySQL does not have a high-performance way to checksum data
 - Google knows this too; they built their own way
- Maatkit can use non-built-in functions such as UDFs
 - Maatkit ships with FNV_64
 - And MurmurHash too, thanks to Mike Hamrick
- Otherwise, it falls back to CRC32
 - I have seen CRC32 collisions in the real world
- If you don't want a hash collision, use MD5 or SHA1
 - But you should really just install FNV_64



mk-table-sync

Fixing the Slave

- So how do we fix the slave?
 - While it is running?
 - Without interrupting service or stopping replication?
 - Conveniently?
 - Safely, without destroying data on the master*?
 - While dealing with edge cases like master-master replication?
- In other words, what's the silver bullet?
 - Hint: it is not always “discard the slave and start afresh”
 - It is also not “start the slave with `--slave-skip-errors=ALL`”

Fixing the Slave – Silver Bullet

```
$ mk-table-sync h=127.0.0.1,P=12346 \  
  --replicate test.checksum --synctomaster --print --execute  
REPLACE INTO `test`.`test`(`a`) VALUES (1);
```

- There's actually a lot going on there
 - Discover portions of tables that need syncing by looking at test.checksum
 - Discover the server's master and connect to it
 - Print & execute SQL to sync the data
 - This SQL is executed **on the master**
 - It is not safe to change data on the slave, that's what caused this mess!



mk-slave-restart

mk-slave-restart

- Ultimate foot-gun: `mk-slave-restart`
- Repeatedly restarts one or many slaves by skipping errors, refetching relay logs, etc
- Useful in dire circumstances where you want to live with busted data until you can fix it

```
$ mk-slave-restart -h 127.0.0.1 -P 12346
```

```
$ mk-slave-restart -h 127.0.0.1 -P 12346 --recurse 5
```



mk-slave-delay

mk-slave-delay

- Want to be able to “roll back” unwanted statements?
 - You can't. Sorry.
- There is no substitute for good backups
- But having a delayed slave is still good for some purposes
- The mk-slave-delay tool does this efficiently

```
$ mk-slave-delay --delay 1h localhost
```



mk-heartbeat

mk-heartbeat

- **SHOW SLAVE STATUS** is not a good check
 - It lies in lots of cases, doesn't work on slaves of slaves, and if the slave is broken, it doesn't say anything useful
- Don't devise elaborate schemes: get direct evidence!
 - Replicate some data, then check the replicated data

```
$ mk-heartbeat -D test --update -h 127.0.0.1 -P 2900
```

```
$ mk-heartbeat -D test --monitor -h 127.0.0.1 -P 2900
```

```
1s [ 0.02s, 0.00s, 0.00s ]
```

```
1s [ 0.03s, 0.01s, 0.00s ]
```

```
1s [ 0.05s, 0.01s, 0.00s ]
```

```
1s [ 0.07s, 0.01s, 0.00s ]
```

```
1s [ 0.08s, 0.02s, 0.01s ]
```

```
$ mk-heartbeat -D test --check -h 127.0.0.1 -P 2900 --recurse 1
```

```
127.0.0.1      1
```

```
127.0.0.1      1
```

```
127.0.0.1      1
```

mk-slave-*

Solutions in search of a problem

```
$ mk-slave-find --host 127.0.0.1 --port 2900
127.0.0.1:2900
+- 127.0.0.1:2903
+- 127.0.0.1:2901
  +- 127.0.0.1:2902
```

```
$ mk-slave-move h=127.0.0.1,P=2902 --sibling-of-master
```

```
$ /tmp/2902/use -e 'start slave'
```

```
$ mk-slave-find --host 127.0.0.1 --port 2900
127.0.0.1:2900
+- 127.0.0.1:2901
+- 127.0.0.1:2903
+- 127.0.0.1:2902
```

mk-parallel-[dump|restore]

mk-parallel-[dump|restore]

```
$ mk-parallel-dump -h 127.0.0.1 -P 2900 --tab  
default: 18 tables, 18 chunks, 18 successes, 0 failures,  
0.54 wall-clock time, 0.87 dump time
```

```
mysqldump: 75m18s  
maatkit: 8m13s  
mydumper: 6m44s <-- http://dammit.lt/2009/02/03/mydumper/
```

```
$ mk-parallel-restore -h 127.0.0.1 -P 2900 --tab default  
18 tables, 36 files, 18 successes, 0 failures, 0.88 wall-clock  
time, 1.68 load time
```



mk-find

mk-find

- INFORMATION_SCHEMA is great. MySQL's implementation of it is not.
 - It's a quick way to harm a production server!

```
$ mk-find test --engine MyISAM
`test`.`dest`
`test`.`fill`
`test`.`parent`
`test`.`query_review`
`test`.`query_review_history`
`test`.`source`
`test`.`table_2`
```

```
# Convert tables to InnoDB
```

```
~$ mk-find test --engine MyISAM \  
--exec 'ALTER TABLE %D.%N ENGINE=InnoDB'
```

What Lies Ahead?

- Immediate Priorities
 - Testing & 100% Code Coverage
 - mk-audit beef up
- Long Term
 - mk-backup-wrapper
 - mk-schema-audit
 - mk-check-config
 - To replace: `mysqld --help --verbose`
 - Mk-data-generator