PalominoDB

Proven Database Excellence

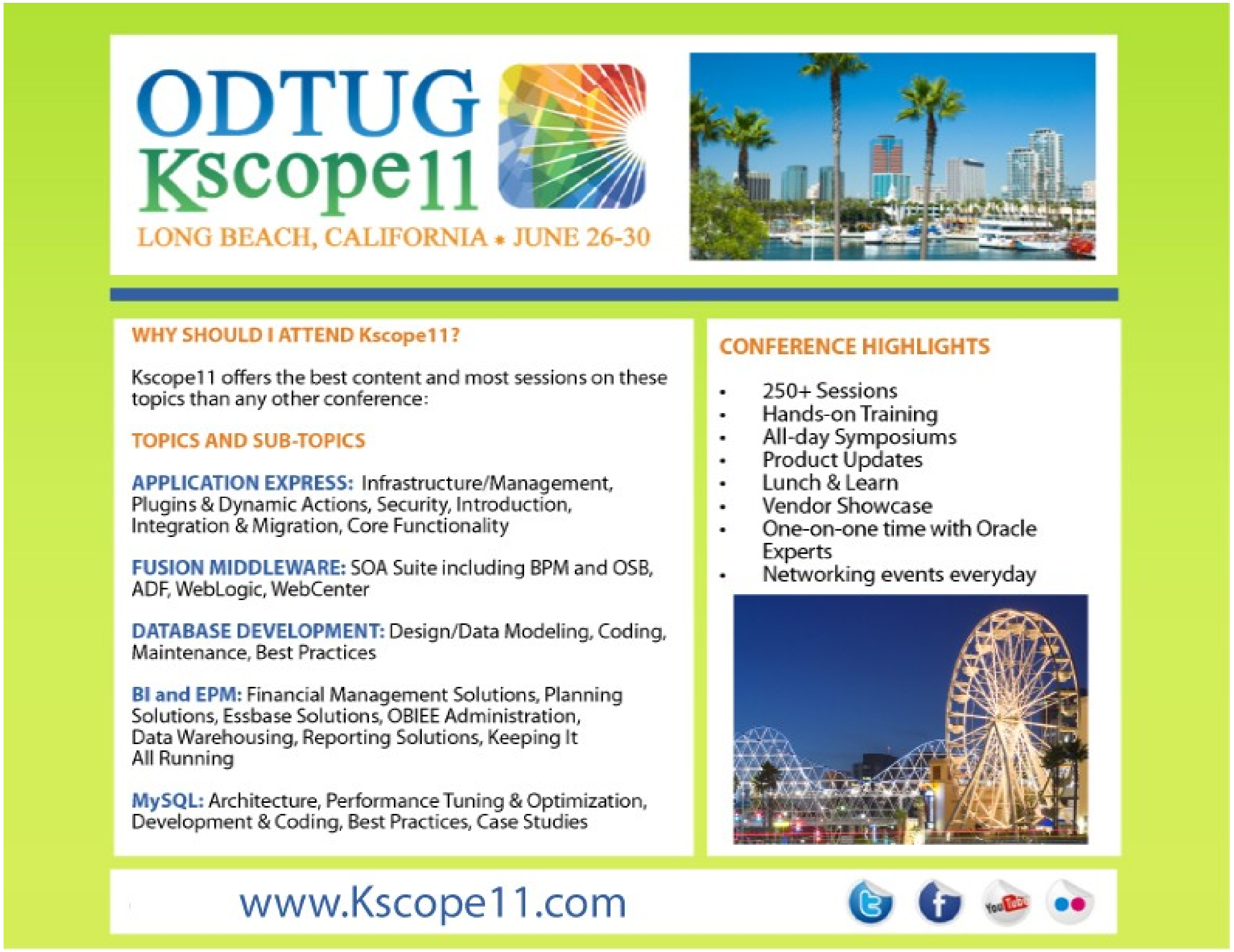
# Find Query Problems Proactively with Query Reviews

Presented by:
Sheeri K. Cabral
Database Operations Manager
www.palominodb.com

Register today!

Don't miss my 3-hour master class:

"Blazingly Fast MySQL Queries"



ODTUG Kscope11

LONG BEACH, CALIFORNIA • JUNE 26-30

**WHY SHOULD I ATTEND Kscope11?**

Kscope11 offers the best content and most sessions on these topics than any other conference:

**TOPICS AND SUB-TOPICS**

**APPLICATION EXPRESS:** Infrastructure/Management, Plugins & Dynamic Actions, Security, Introduction, Integration & Migration, Core Functionality

**FUSION MIDDLEWARE:** SOA Suite including BPM and OSB, ADF, WebLogic, WebCenter

**DATABASE DEVELOPMENT:** Design/Data Modeling, Coding, Maintenance, Best Practices

**BI and EPM:** Financial Management Solutions, Planning Solutions, Essbase Solutions, OBIEE Administration, Data Warehousing, Reporting Solutions, Keeping It All Running

**MySQL:** Architecture, Performance Tuning & Optimization, Development & Coding, Best Practices, Case Studies

**CONFERENCE HIGHLIGHTS**

- 250+ Sessions
- Hands-on Training
- All-day Symposiums
- Product Updates
- Lunch & Learn
- Vendor Showcase
- One-on-one time with Oracle Experts
- Networking events everyday

www.Kscope11.com

## Query Review

- What is it?
  - Systematic review of all queries
- Why do it?
  - Find queries before they become a problem
  - Often a sample query is non-trivial to find

# Query Review

- Who should do it?
  - Optimization knowledge
- When and where should it be done?
  - dev → test,load test,staging → production

## Main tool

- mk-query-digest
  - “query fingerprint”
- Can be used on:
  - Slow query logs
  - Binary logs
  - General query logs

# More mk-query-digest sources

- Direct database querying
  - Uses SHOW FULL PROCESSLIST
- pglog (Postgres)
- Parsing tcpdump for traffic:
  - MySQL
  - memcached
  - HTTP

# Getting mk-query-digest

- wget maatkit.org/get/mk-query-digest
  - Easiest
  - Not always up-to-date!
- http://code.google.com/p/maatkit/
  - More work
  - You get all the maatkit tools, not just one
  - Most up to date

## What is reported on

- Default setup uses --limit 95%:20
  - To see all queries, --limit 100%
- No --filter by default
- --filter
  - Any attribute at http://code.google.com/p/maatkit/wiki/EventAttributes
  - User, host, database, process id, lock_time, Memc_miss, Rows_sent, Rows_examined, Rows_affected, Rows_read, Query_time, insert_id

## Other filters

- If using Percona's patches, you can filter on queries that cause:
  - Filesorts, disk filesorts
  - Temp tables, Temp disk tables
  - Full table scan, full join
  - Query cache hit
  - and more...

# Output

- Overall summary
- Detailed report of matching queries
- Query Analysis Summary

- Commands run for examples:

```
perl mk-query-digest --limit 100% \
--review h=127.0.0.1,P=3307,D=maatkit,t=query_review,u=user,p=pass \
--create-review-table --type genlog genlog127.sql > genlogoutput.txt

perl mk-query-digest --limit 100% \
--review h=127.0.0.1,P=3307,D=maatkit,t=query_review,u=user,p=pass \
--type binlog binlog325.sql > binlogoutput.txt
```

## Overall summary (genlog)

```
# 229.7s user time, 860ms system time, 94.79M rss, 145.48M vsz
# Overall: 906.22k total, 720 unique, 143.84 QPS, 0x concurrency_________
#                    total     min     max     avg     95%  stddev median
# Exec time              0       0       0       0       0       0      0
# Time range          2010-03-12 10:45:01 to 2010-03-12 12:30:01
# bytes            242.78M       5  69.06k  280.91  563.87  819.66 112.70
```

# Overall summary (binlog)

```
# 390.2s user time, 1.8s system time, 62.70M rss, 113.45M vsz
# Overall: 1.07M total, 252 unique, 245.71 QPS, 5.69Gx concurrency________
#                     total min        max        avg    95% stddev median
# Exec time   24786256998598s    0 4294967295s 23168970s 992ms 302909074s 0
# Time range          2010-04-10 07:14:17 to 2010-04-10 08:26:51
# @@session               86       0        1     0.50     0.99     0.50   0.99
# @@session              585       1        4     3.42     3.89     0.68   3.89
# @@session            3.44k       8       33    20.57    31.70    12.00  31.70
# @@session            1.34k       8        8        8        8        0      8
# @@session                1       1        1        1        1        0      1
# @@session         837.08k 837.08k 837.08k 837.08k 837.08k        0837.08k
# @@session               85       0        1     0.50     0.99     0.50      0
# bytes             415.05M       5    1.02M   349.05   563.87    1.34k 537.02
# error cod                0       0        0        0        0        0      0
```

## Query analysis part 1 (genlog)

```
# Query 9: 1.69 QPS, 0x concurrency, ID 0x188B27831A9DE05B at byte
268215186

# This item is included in the report because it matches --limit.

#              pct   total     min     max     avg     95%  stddev  median

# Count          1   10647

# Exec time      0       0       0       0       0       0       0       0

# Databases              1 proddb

# Time range 2010-03-12 10:45:02 to 2010-03-12 12:30:01

# bytes          0 613.45k      59      59      59      59       0      59
```

## Query analysis part 1 (binlog)

```
# Query 5: 3.90 QPS, 297.34Mx concurrency, ID 0x188B27831A9DE05B at byte
596881917

# This item is included in the report because it matches --limit.

#               pct   total     min     max     avg     95%  stddev median

# Count           1     16829

# Exec time     5 1284195222560s 0 4294967295s 76308469s 992ms 546294873s 0

# Databases                1 proddb

# Time range 2010-04-10 07:14:52 to 2010-04-10 08:26:51

# bytes         0 969.38k      58      59   58.98   56.92       0   56.92

# error cod     0       0       0       0       0       0       0       0
```

## Query analysis part 2 (genlog)

```
# Query_time distribution
#   1us
#  10us
# 100us
#   1ms
#  10ms
# 100ms
#    1s
#  10s+
# Review information
#    first_seen: 2010-03-12 10:45:02
#     last_seen: 2010-03-12 12:30:01
#   reviewed_by:
#   reviewed_on:
#      comments:
```

# Query analysis part 2 (binlog)

```
# Query_time distribution
#   1us
#  10us
# 100us
#   1ms
#  10ms
# 100ms
#    1s  ############################################################
#  10s+  ################
# Review information
#    first_seen: 2010-03-12 10:45:02
#     last_seen: 2010-04-10 08:26:51
#   reviewed_by:
#   reviewed_on:
#      comments:
```

## Query analysis part 3 (genlog)

```
# Tables
#    SHOW TABLE STATUS FROM `proddb` LIKE 'colors'\G
#    SHOW CREATE TABLE `proddb`.`colors`\G
update colors set publishable_flag = true where id = 267354\G
# Converted for EXPLAIN
# EXPLAIN
select  publishable_flag = true from colors where  id = 267354\G
```

# Query analysis part 3 (binlog)

```
# Tables

#    SHOW TABLE STATUS FROM `proddb` LIKE 'colors'\G

#    SHOW CREATE TABLE `proddb`.`colors`\G

update colors set publishable_flag = true where id = 284297\G

# Converted for EXPLAIN

# EXPLAIN

select  publishable_flag = true from shopping_events where  id =
284297\G
```

# Query analysis part 1 (binlog)

```
# Query 5: 3.90 QPS, 297.34Mx concurrency, ID 0x188B27831A9DE05B at byte
596881917

# This item is included in the report because it matches --limit.

#              pct   total     min     max     avg     95%  stddev median

# Count          1     16829

# Exec time      5 1284195222560s 0 4294967295s 76308469s 992ms 546294873s 0

# Databases                1 proddb

# Time range 2010-04-10 07:14:52 to 2010-04-10 08:26:51

# bytes          0 969.38k      58      59   58.98   56.92       0   56.92

# error cod      0       0       0       0       0       0       0       0



update colors set publishable_flag = true where id = 284297\G
```

# Query Analysis Summary

```
# Profile
# Rank Query ID           Response time                Calls  R/Call
Item
# ==== ================== ============================ ====== ===============
#    1 0x85FFF5AA78E5FF6A 9856949962471.0000 39.8% 177057   55671054.8720
                                                                    BEGIN
#    2 0x8F345B7550CA9147 4664334749763.0000 18.8% 686030    6799024.4592
                                              INSERT user_events_live
#    3 0xCACEE7C0CF15B39B 2619930057821.0000 10.6%  63756   41093074.5000
                                                             UPDATE skus
#    4 0x308A3C4E761F5834 1378684503375.0000  5.6%  17845   77258868.2194
                                                 UPDATE shopping_events
```

**`#    5 0x188B27831A9DE05B 1284195222560.0000  5.2%  16829   76308468.8668`**
**`                                                          UPDATE colors`**

```
#    6 0xD8F78067CE3F07AB 1279900255360.0000  5.2%  18180   70401554.2002
                                                             UPDATE offers
#    7 0x3C70600B502E3A08 1215475745855.0000  4.9%  16829   72225072.5447
                                                          UPDATE products
```

# The query_review table

- Remember, we did the command:

```
perl mk-query-digest --limit 100% \
--review h=127.0.0.1,P=3307,D=maatkit,t=query_review,u=user,p=pass \
--create-review-table --type binlog binlog325.sql > binlogoutput.txt
```

- What does the query review table look like?

```
mysql> select * from query_review where checksum=0x188B27831A9DE05B\G
*************************** 1. row ***************************
   checksum: 1768550722713804891
fingerprint: update colors set publishable_flag = true where id = ?
     sample: update colors set publishable_flag = true where id =
100563
 first_seen: 2010-03-12 10:45:02
  last_seen: 2010-04-10 08:26:51
reviewed_by: NULL
reviewed_on: NULL
   comments: NULL
1 row in set (0.00 sec)
```

## How do we review a query?

- EXPLAIN, SHOW CREATE TABLE, etc.
- Now what?

```
mysql> update query_review set reviewed_by='Sheeri',
reviewed_on=now(), comments='This query is OK, it uses the primary
key to search on.' where checksum=1768550722713804891;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

- One query down.....

```
mysql> select count(*) from query_review where reviewed_on is null;
+----------+
| count(*) |
+----------+
|      769 |
+----------+
1 row in set (0.00 sec)
```

- 769 to go!

## Systematic approach

- You can look at a few queries per day
- Reviewed queries do not appear in subsequent reports of mk-query-digest
  - If you have something in reviewed_by
  - Unless you specify --report-all

# Query review

- --no-report to just parse a log to the database:

```
perl mk-query-digest --limit 100% --no-report -review \
h=127.0.0.1,P=3307,D=maatkit,t=query_review,u=user,p=pass \
--type binlog mybinlog.txt
```

- Can save counts, etc to an historical table

```
perl mk-query-digest --limit 100% --no-report -review \
h=127.0.0.1,P=3307,D=maatkit,t=query_review,u=user,p=pass \
--create-review-history-table -review-history \
h=127.0.0.1,P=3307,D=maatkit,t=qr_history,u=user,p=pass \
--type genlog mygenlog.txt
```

# Query review history

```
  mysql> select * from qr_history where checksum=0x188B27831A9DE05B\G

*************************** 1. row ***************************
            checksum: 1768550722713804891
              sample: update colors set publishable_flag = true where id
= 284297
              ts_min: 2010-04-10 07:14:52
              ts_max: 2010-04-10 08:26:51
              ts_cnt: 16829
      Query_time_sum: 1.2842e+12
      Query_time_min: 0
      Query_time_max: 4.29497e+09
   Query_time_pct_95: 0.992137
   Query_time_stddev: 5.46295e+08
   Query_time_median: 0
       Lock_time_sum: NULL
       Lock_time_min: NULL
       Lock_time_max: NULL
    Lock_time_pct_95: NULL
    Lock_time_stddev: NULL
    Lock_time_median: NULL

       Rows_sent_sum: NULL
       Rows_sent_min: NULL
       Rows_sent_max: NULL
    Rows_sent_pct_95: NULL
    Rows_sent_stddev: NULL
    Rows_sent_median: NULL
   Rows_examined_sum: NULL
   Rows_examined_min: NULL
   Rows_examined_max: NULL
Rows_examined_pct_95: NULL
Rows_examined_stddev: NULL
Rows_examined_median: NULL
```

# Query review history

```
mysql> select * from qr_history where checksum=0x188B27831A9DE05B\G
*************************** 1. row ***************************
            checksum: 1768550722713804891
              sample: update colors set publishable_flag = true where id
= 284297
              ts_min: 2010-04-10 07:14:52
              ts_max: 2010-04-10 08:26:51
              ts_cnt: 16829
      Query_time_sum: 1.2842e+12
      Query_time_min: 0
      Query_time_max: 4.29497e+09
   Query_time_pct_95: 0.992137
   Query_time_stddev: 5.46295e+08
   Query_time_median: 0
```

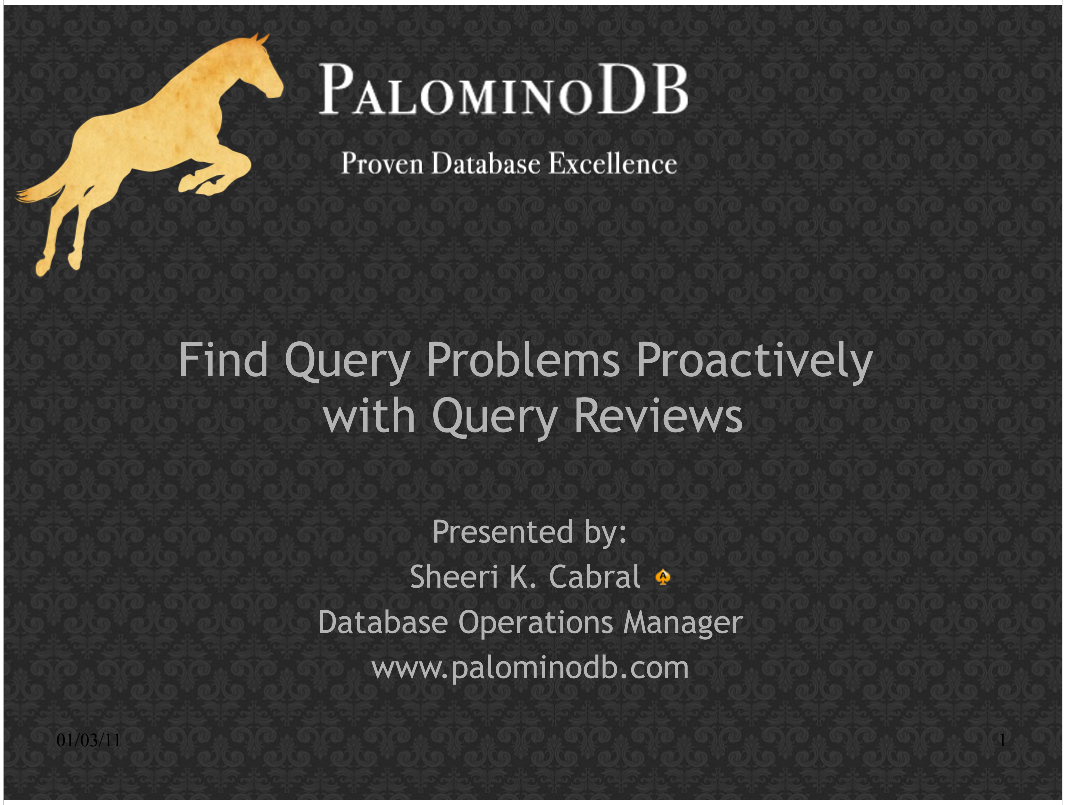
```
************* 2. row *************
checksum: 1768550722713804891
sample: update colors set
publishable_flag = true where id =
279850
ts_min: 2010-03-24 10:45:01
ts_max: 2010-03-24 12:30:00
ts_cnt: 7109
Query_time_sum: 0
Query_time_min: 0
Query_time_max: 0
Query_time_pct_95: 0
Query_time_stddev: 0
Query_time_median: 0
```

## What I'd like to see

- Besides query reviews being common practice...
- More fields in the query_review table
  - what index(es) are used – fields, index type
  - Tables involved and their approx row count
  - Approx rows examined from EXPLAIN
- More fields in the query_review_history table
  - Source (genlog, binlog, etc)
  - When the review was done

## Start Today!

- Grab a log
- Find a test machine with a database
- Start EXPLAINing all your queries
- mk-query-digest has tons of other great features other than query reviews.....



PALOMINODB

Proven Database Excellence

# Find Query Problems Proactively with Query Reviews

Presented by:
Sheeri K. Cabral
Database Operations Manager
www.palominodb.com

Ie, if a dev finds a problem, he can't always give you the exact query that's causing the issue.

Ie, if a dev finds a problem, he can't always give you the exact query that's causing the issue.

Whoever is responsible for the query review should have a working knowledge of query and schema optimization.

In an ideal world, you'd have the rule “no query gets put into production without having a qualified person EXPLAIN it first” and think of all the ramifications of if/when the table gets bigger.

It should probably be done in testing or staging – definitely before the code is released into the wild, although looking at the production queries every once in a while is a good idea too, if you can manage it, because you may find that some actions are more popular than you planned!

--limit specifies the queries to show; you can put in a percentage or a number, or both separated by a colon.  If you do both, it will pick whichever comes first.  This is the top % of worst queries to show, or the top N worst queries to show.  So by default it shows you the top 95% worst queries.

You can also filter by keywords in the query, like SELECT or a certain table.

So this tool is very useful to filter out only queries that had certain attributes – maybe you wan to to look at all queries that examined over X amount of rows.  Maybe you want to see what memcached is missing, or find queries that are locked for long periods of time.  Obviously this tool is VERY powerful!!!

I'm showing both the binlog and the genlog because the genlog doesn't show times and often times are important. Of course if your system has different resources like CPU speed and available RAM, then the times may be completely different. But doing this type of analysis on a load testing server is a great way to find potentially bad queries, and/or to get lots of queries.

I do suggest, if you can, using the general log for completeness' sake. You could also set the slow query log to a very low threshold to get more completeness for a query review.

If you can't see this text, move closer now.

This particular output is from using –limit 100%, using a 256 Mb general log file from a production machine.

There is no introductory line, or a line stating what the report was called with, which would be nice (ie, --limit 100%)

The first 2 fields are the time it took to parse the data. The last 2 fields on the first line are how much memory it used to parse.

Rss = size of resident, non-swapped memory
Vsz = memory usage of entire process incl. RSS

Then we have how many queries, how many unique queries, the query per second processing time and the concurrency. This used the general log, which doesn't have times, so those aggregates aren't shown, and neither is concurrency

Note that bytes is the size of the file I parsed. Those aggregates are working fine.

This particular output is from using –limit 100%, using a 256 Mb general log file from a production machine.

There is no introductory line, or a line stating what the report was called with, which would be nice (ie, --limit 100%)

The first 2 fields are the time it took to parse the data.  The last 2 fields on the first line are how much memory it used to parse.

Rss = size of resident, non-swapped memory
Vsz = memory usage of entire process incl. RSS

Then we have how many queries, how many unique queries, the query per second processing time and the concurrency.  This used the general log, which doesn't have times, so those aggregates aren't shown.

Note that bytes is the size of the file I parsed.  Those aggregates are working fine.

This is just the first part of the detailed query analysis...

Pct is % of queries that are this fingerprint
Total is count of queries that are this fingerprint

Again, no times in the general log.

Same first part analysis of the same query, from the binlog.   Different bytes b/c different time.

2nd part of the detailed analysis is the query time distribution and review information. Obviously for the general log, this is very boring.

Note that the review information comes from the database we specified with the --review command.  This shows what it looks like after the first run – the dates are automatically put in using the timestamps in the log.

If we hadn't put –review this 2nd part would just have the query time distributions

The time analysis. One thing you could do is set the slow query log to log almost everything, and then you'd have a time analysis. Of course to be proactive you'd want to do this on staging, or even testing, so that the query can be changed to be optimal even **before** you get to production.

Finally we get to some table information, and the query itself.

The binlog is the same.

Now, let's look at the query again, in context.

That's a long time to be executing for a simple update statement. It's actually a bug in MySQL, which has been reported and confirmed.

As you can see the summary is also useful. You can order by other items, by the way, the default is to order by most count.

I showed you the report so that I could show you this – the query review table doesn't have any of the aggregate information (but you can save it, we'll see that later).  Just the query itself, and whether or not it has been reviewed.

See the blazingly fast SQL 3-hour master class at Kscope to learn how to do this, also to get optimization tips.

So if you've reviewed all queries in the top 10 and you ask for the top 10, you won't see anything in the output.

You can cheat by not putting anything in reviewed by, just put reviewed on and the comments.  You will see the reviewed_on and comments if you do that.

This is 1 row, the next slide will show the non-null from both rows....

Non-null from both runs

Fields and index types – fields in the indexes, and whether they're primary/unique or not.

Tables involved and row count – you could query information_schema to find queries that maybe you should re-review now that a table is much bigger.  Also if you're considering adding an index you could easily come up with all the queries that use that table.