

# How I Used Query Analysis to Speed Up My Applications

Martin MC Brown, Sun Microsystems



# Key Points for Good QA

- ✦ Check the query
- ✦ Check the table/index
- ✦ Check a range of WHERE clauses
- ✦ Check over a long period of time
- ✦ Check returned rows/examined rows
- ✦ Turn off the query cache



# Using EXPLAIN

- ✦ Good starting points
- ✦ Shows expected execution
- ✦ Shows index usage, row counts, sorting performance
- ✦ When using EXPLAIN EXTENDED use SHOW WARNINGS



# EXPLAIN No Index

```
      id: 1
select_type: SIMPLE
      table: logs_amavis
      type: ALL
possible_keys: NULL
      key: NULL
      key_len: NULL
      ref: NULL
      rows: 301449
  filtered: 100.00
      Extra: Using temporary; Using
filesort
```



# EXPLAIN With Index

```
        id: 1
select_type: SIMPLE
      table: logs_amavis
       type: index
possible_keys: NULL
        key: pmpmpt
    key_len: 38
         ref: NULL
        rows: 302078
      Extra: Using index
1 row in set, 1 warning (0.00 sec)
```



# Getting the best from EXPLAIN

- ✦ Generate a query log
- ✦ Run it multiple times
- ✦ Run it on every query type
- ✦ Run it with a range of different values
- ✦ Check index usage, filesort, temporary tables



# Slow Query Log

- ✦ Easy to use
- ✦ Enable at startup
- ✦ Or use MySQL 5.1 and enable on the fly
- ✦ Set your threshold
- ✦ Leave it on for a **long** time



# Slow Query Log Sample

```
Time: 090415 9:53:23
# User@Host: root[root] @ localhost [127.0.0.1]
# Query_time: 0.559289 Lock_time: 0.000068
Rows_sent: 8 Rows_examined: 301449
use intranet_mcsip;
SET timestamp=1239785603;
select process_mode,process_type,count(message_id)
as cnt from logs_amavis group by
process_mode,process_type;
```



# Summarized Output

Query	Time	RowsR	RowsS
select process_mode,process_type,count(message_id) as cnt from logs_amavis group by process_mode,process_type;	0.25 0.52 2.79	8/8/8	302078 302078 302078
select count(message_id) as cnt,date_format(from_unixtime(datetime),'%Y-%m') as fmtdate,process_mode,process_type from logs_amavis where datetime >= 1208101059 and datetime <= 1239595200 group by process_mode,process_type,fmtdate			
select inhost,path,(avg(avail)/(1024*1024*1024)) as cnt,date_format(logtime,'%Y%m%d %h:%i') as fmtdate, date_format(logtime,'%d/%m/%y %h:%i') as date_fmtd from statmon_machine_disk left join (statmon_machine_statdate) on (statmon_machine_disk.statid = statmon_machine_statdate.statid) left join statmon_machines on (statmon_machine_statdate.machineid = statmon_machines.machineid) where inhost='bear' and logtime >= '2008-10-14 11:28:00' and path != '/' boot" and logtime <= '2009-04-14 11:28:00' group by fmtdate,path			



# Using MySQL Proxy

- ✦ Record queries and times
- ✦ Requires a potential change of config
- ✦ Lua Script to output times, rows returned
- ✦ Can't go deeper



# Using DTrace

- ✧ Passive
- ✧ Easy to switch on and off
- ✧ Can give more detailed info
  - ✧ Lock times, QC, row counts
- ✧ With care, slow query log and EXPLAIN all in one



# Aggregates

```
insert into t4 values (1,'hello')
```

	value	----- Distribution -----	
count	32768		0
	65536	@@@@	4
	131072	@@@@	1
	262144		0

```
select * from t4
```

	value	----- Distribution -----	
count	134217728		0
	268435456	@@@@	1
	536870912		0



# DTrace and Filesort

Dur mus Query

11335469 Filesort on t1

11335787 select \* from t1 order by i limit 100

466734378 create index t1a on t1 (i)

26472 select \* from t1 order by i limit 100



# Probing everything

St	QC	ConnID	Dur ms	Bytes	Matched	Changed	Query
--							
3		6					Command start
							Parsing: insert into t4 (select *
							from t4)
0			0				-> Parsing Complete
0			0				-> Write lock: t4
0			0				-> Write lock completed
0			0				-> Read lock: t4
0			0				-> Read lock completed
0			0				-> Unlock completed
			36521				-> Lock duration for: t4
							-> Unlock: t4
0			0				-> Unlock completed
33	0	6	36521		400944	0	insert into t4 (select * from t4)
		6	0	0			-> Net read
		6	0	54			-> Net write
		6	34128				-> Row ops
0							Command done
--							
			36521				-> Lock duration for: t4
							-> Unlock: t4



# Collate, Aggregate, Pinpoint

- ✦ Collect as much info as you can
- ✦ Work out the rule
- ✦ Work out the exceptions
- ✦ Rework your structure



# QA Limitations and Issues

- ✦ Must monitor multiple hosts
- ✦ Multiple queries/values/hosts = complexity
- ✦ Tracking queries over time is time consuming
- ✦ Tracking queries over time generates a lot of data

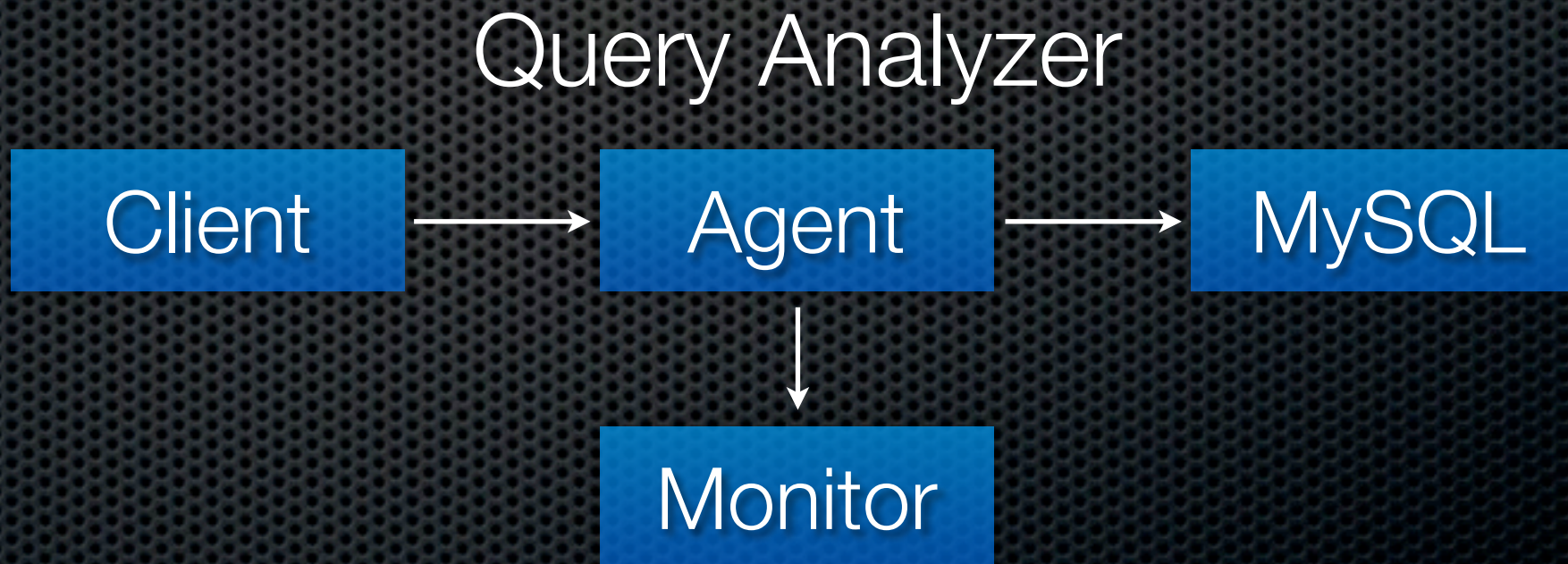


# Query Analyzer

- ✦ Collates queries for you
- ✦ Show rows returned, bytes returned
- ✦ View by query, user, database, client



# Query Analyzer Structure





# Demo



# Use Case: Too Much Info

- ✦ Graph Query
- ✦ Worked fine for 6 months
- ✦ At 12 months, 15 secs for a page
- ✦ Combination of DB and App
- ✦ DB Performance Minimal



# Use Case: Nested Queries

- ✦ Reporting app with compound queries
- ✦ QA pinpointed the query group
  - ✦



# Use Case: Summary Tables

- ✦ Mail reporting application
- ✦ EXPLAIN showed high row counts
- ✦ Slow Query Log showed low row returns
- ✦ Created multiple VIEW on the data after update



# Use Case: Long Queries

- ✦ Used Quan to monitor server
- ✦ Background/Batch jobs slow
- ✦ Realigned the execution times



# Where next

- ✦ Visit <http://coalface.mcslp.com>
- ✦ Try out Query Analyzer and find your most expensive queries
  - ✦ <http://www.mysql.com/trials/>



# Questions?