MySQL Performance Tuning and Benchmarking

... or learn how to make your MySQL go faster, perform better, find trouble pain points, etc.

Colin Charles, Community Relations Manager, APAC
colin@mysql.com | http://bytebot.net/blog/

Who am I?

- Community Relations Manager, APAC
  - Distribution Ombudsman
  - Community Engineering
  - Summer of Code
  - Forge Dude
  - Generalised Dolphin Wrangler
- Previously:
  - Fedora Project FESCO and PowerPC hacker
  - OpenOffice.org contributor

Pre-requisite knowledge

- A lot of examples used here will cite the Sakila sample database
  - http://forge.mysql.com/wiki/SakilaSampleDB
  - http://dev.mysql.com/doc/
    - look for the sakila DB, there's also a world DB (training use) and menagerie (Beginning MySQL book use)

Agenda

- An Introduction to Benchmarking
- Data Structures
- Query Optimisation and Query Cache
- Indexes
- Storage Engines
- my.cnf options
- Real World MySQL Use
- Getting the code

Why Benchmark?

- Allows *tracking of performance over time*
  - application
  - SQL snippet
  - application script or web page
- You get *load* and *stress* information
- Ever wondered if for the job InnoDB or MyISAM would be better? Or if running on Linux or FreeBSD made more sense?

The Good Scientists Guide to Benchmarking

- The scientific method suggests changing only one variable at a time
  - configuration variable, adding an index, schema modification, SQL snippet change
- The scientific method suggests repetition, more than once to verify results. If results vary greatly, think about taking averages.
  - **Repeat, rinse, repeat, rinse!**
  - (do it at least 3 times)

The Good Scientists Guide to Benchmarking II

- Isolate your environment
  - beware network traffic analysers
  - non-essential services
  - MySQL's very own query cache
- Use a different MySQL instance
  - Use the `--socket` configuration variable for instance differentiation
- Save ***all*** configurations!

Benchmarking Tools

- super-smack
  - http://vegan.net/tony/supersmack/
  - Flexible tool for measuring SQL script performance
- `mysqlslap` (like `ab`; in MySQL 5.1)
- MyBench
  - http://jeremy.zawodny.com/mysql/mybench/
- SysBench
  - http://sysbench.sourceforge.net/
  - For raw comparisons of different MySQL versions/platforms
- Apache Bench

Benchmarking Tools II

- SHOW commands in MySQL
  - `SHOW PROCESSLIST | STATUS | INNODB STATUS`
  - `SHOW PROFILE` – in 5.0.37 and above, Community Contribution, Linux only
- `EXPLAIN` and the Slow Query Log
- MyTop
  - http://jeremy.zawodny.com/mysql/mytop/
- vmstat/ps/top/gprof/oprofile (and contents of procinfo)

SHOW PROFILE

- SELECT @@profiling;
  - Turn it on: SET @@profiling=1;
- SELECT * FROM store;
- SHOW PROFILE SOURCE;
- SHOW PROFILE ALL;

```
(root@hermione) [sakila]> show
profile;
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| (initialization)     | 0.000036 |
| Opening tables       | 0.000012 |
| System lock          | 0.000005 |
| Table lock           | 0.000008 |
| init                 | 0.000016 |
| optimizing           | 0.000005 |
| statistics           | 0.000012 |
| preparing            | 0.000008 |
| executing            | 0.000003 |
| Sending data         | 0.000103 |
| end                  | 0.000005 |
| query end            | 0.000003 |
| freeing items        | 0.000007 |
| closing tables       | 0.000013 |
| logging slow query   | 0.000003 |
+----------------------+----------+
15 rows in set (0.00 sec)
```

Slow Query Log

- `log_slow_queries=/var/lib/mysql/slow-queries.log`
- `long_query_time = 2`
- Then, use `mysqldumpslow`
- In 5.1, you can log these details directly to a table, and obviously doesn't require a server restart
  - Currently, when editing my.cnf, you need to restart the server to incorporate changes
- Slow Query Log Filter: http://code.google.com/p/mysql-log-filter/

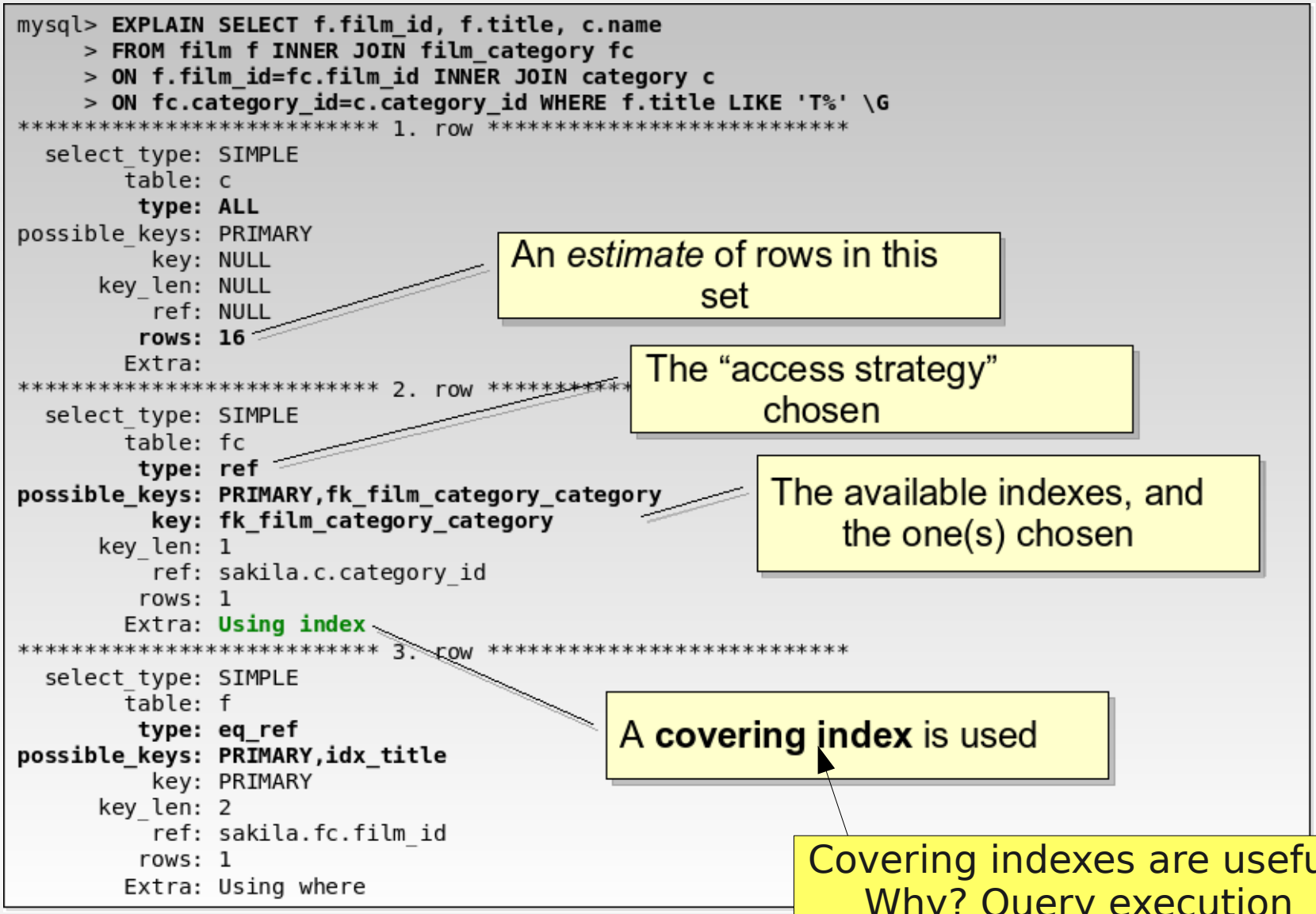
EXPLAIN basics

- Provides the execution plan chosen by the MySQL optimiser for a specific `SELECT` statement
- Usage is easy! Just append `EXPLAIN` to your `SELECT` statement
- Each row represents information used in `SELECT`
  - real schema table
  - *virtual* (derived) table or temporary table
  - subquery in `SELECT` or `WHERE`
  - union sets

EXPLAIN columns

- **select_type** – type of “set” the data in row contains
- **table** – alias (or full table name) of table or derived table from where data in this set comes from
- **type** - “access strategy” used to grab data in set
- **possible_keys** – keys available to optimiser for query
- **keys** – keys chosen by the optimiser
- **rows** – estimate of number of rows in set
- **extra** – information optimiser chooses to give you
- **ref** – shows column used in join relations

EXPLAIN example



http://www.mysqlperformanceblog.com/2006/11/23/covering-index-and-prefix-indexes/

Ranges

```
SELECT * from room
WHERE room_date BETWEEN '2007-09-11' AND
  '2007-09-12'\G;
```

- ensure index is available on field operated upon by range operator
- too many records to return? Range optimisation won't be used and you get an index or full table scan

Scans and seeks

- A **seek**, jumps into a random place (on disk or in memory) to fetch needed data
- A **scan** will jump to the start of the data, and sequentially read (from either disk or memory) until the end of the data
- Large amounts of data?
  - Scan operations are probably better than multiple seek operations

When do you get a full table scan?

- No `WHERE` condition
- No index on any field in `WHERE` condition
- When your range returns a large number of rows, i.e. too many records in `WHERE` condition
- Pre-MySQL 5, using `OR` in a `WHERE` clause
  - now fixed with an index merge, so the optimiser can use more than one index to satisfy a join condition
- `SELECT * FROM`

Subqueries

- Don't use them; replace with a `JOIN`
  - `unique_subquery`: results are known to be distinct
  - `index_subquery`: otherwise
- Co-related subqueries are worse
  - executed once for each matched row in outer set of information
    - kills scalability/performance
    - rewrite as a `JOIN`

```
WHERE p.payment_date = (
 SELECT MAX(payment_date)
  FROM payment
   WHERE payment.customer_id
   = p.customer_id
```

Indexes

- Covering index: all fields in `SELECT` for specific table are contained in index
  - when using `EXPLAIN`, notice “Using index”
- Remember that when using InnoDB, use a small PK value (as it is appended to every record in the secondary index)
  - If you don't add a PK, InnoDB adds one automatically
    - Its a 6-byte integer!
- Always, add a Primary Key

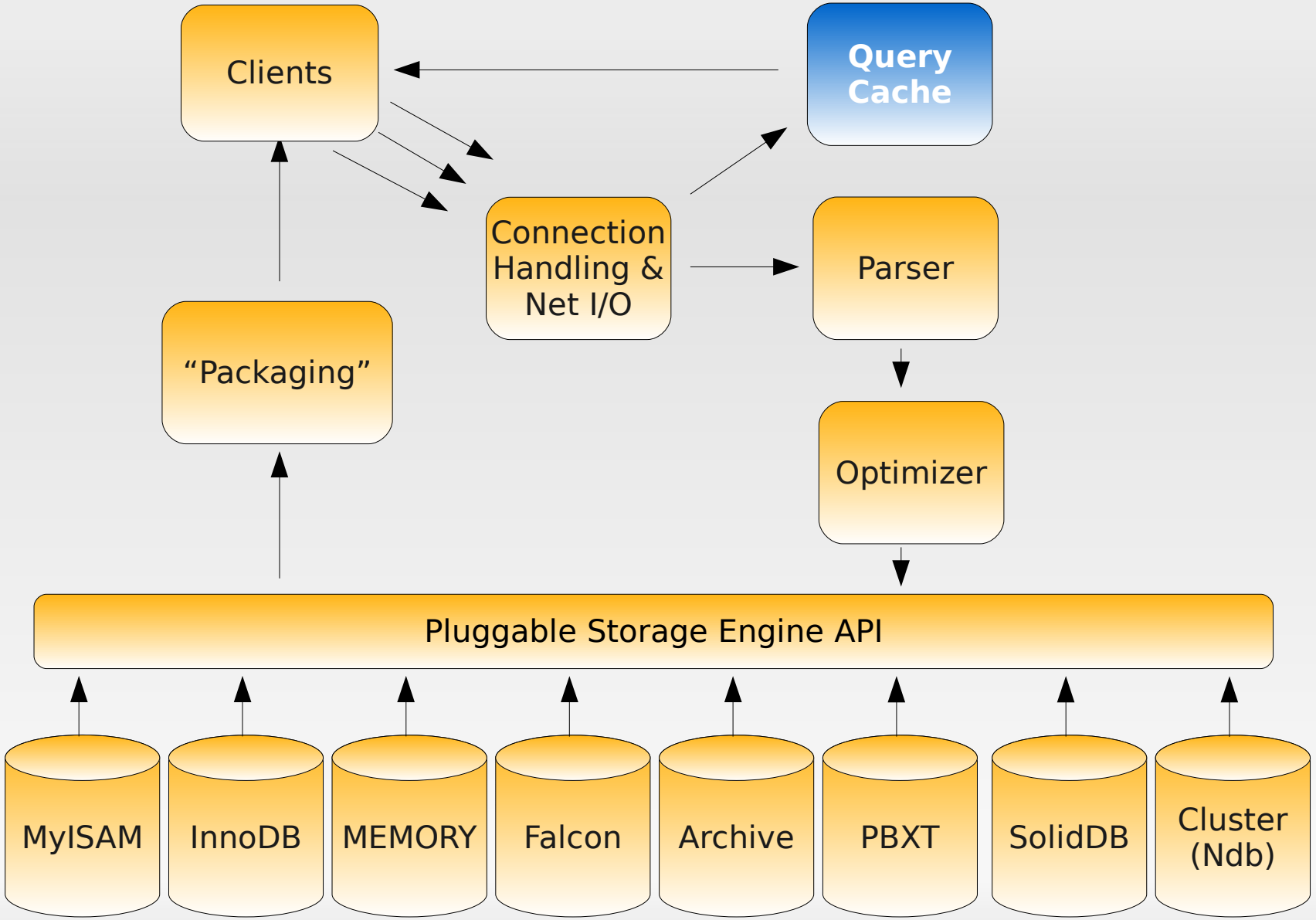
Good Schema Practice

- Use small data types
  - Is a `BIGINT` really required?
- Small data types allow more index and data records to fit into a single block of memory
- Normalise first, de-normalise later
  - Generally, 3NF works pretty well

Storing IP addresses

- IP addresses always become an `INT UNSIGNED`
- Each subnet corresponds to an 8-byte division of the underlying `INT UNSIGNED`
- From string to int? Use `INET_ATON()`
- From int to string? Use `INET_NTOA()`
- We're looking at native types for IPv6, thanks to the Google Summer of Code 2007
  - We have native types for IPv6, in MySQL 6.0-beta

Query Cache


Clients
Query Cache
Connection Handling & Net I/O
Parser
"Packaging"
Optimizer
Pluggable Storage Engine API
MyISAM
InnoDB
MEMORY
Falcon
Archive
PBXT
SolidDB
Cluster (Ndb)

Query Cache

- Understand your applications read/write ratio for most effective use
- Compromise between CPU usage and read performance
- Remember that the bigger your query cache, you may not see better performance, even if your application is **read heavy**

Query Cache Invalidation

- *Coarse* invalidation designed to prevent CPU overuse
  - Happen during finding and storing cache entries
- Thus, *any* modification to any table referenced in the SELECT will invalidate *any* cache entry which uses that table
  - Use vertical table partitioning as a fix
- Query Cache is flushed on each update

Choosing a Storage Engine

- MySQL's strong point: many engines
- Use InnoDB for most operations (esp. OLTP), except:
  - big, read only tables
  - high volume streaming tables (logging)
  - specialised needs (have special engines)
- Tune InnoDB wisely
  - http://www.mysqlperformanceblog.com/files/presentations/UC2007-Innodb-Performance-Optimization.pdf

Choosing a Storage Engine

- MyISAM
  - Has excellent insert performance, small footprint
    - No transactions, FK support
  - Good for logging, auditing, data warehousing
- Archive
  - Very fast insert and table scan performance
  - Read only. Good for archiving, audit logging
- Memory
  - Great for lookup tables, session data, temporary tables, calculation tables

Quick InnoDB Tuning Tips

- `innodb_file_per_table` – splits InnoDB data into a file per table, rather than one large contiguous file
  - allows `optimize table` `` `table` `` to clear unused space
- `innodb_buffer_pool_size = (memory*0.80)`
- `innodb_flush_log_at_trx_commit` – logs flushed to disk at each transaction commit. ACID guarantees, but expensive
- `innodb_log_file_size` – keep it high (64-512MB), however recovery time increases (4GB is largest)

Quick my.cnf tuning tips

Good reference from MySQL Camp: http://mysqlcamp.org/?q=node/39

- `key_buffer_size` – About (memory*0.40) for MyISAM (which uses OS cache to cache data) tables. Dependant on indexes, data size, workloads.
- `table_cache` – Act of opening tables = expensive. Size cache to keep most tables open. 1024 for a few hundred tables
- `thread_cache` – Creation/destruction during connect/disconnect = expensive. 16?
- `query_cache_size` – 32-512MB is OK, but don't keep it too large

Real World MySQL Use (RWMU)

- Run many servers
  - Your serious application cannot run on “the server”
- “Shared nothing” architecture
  - Make no single point of contention in the system
  - Scales well, just by adding cheap nodes
  - If it works for Google, it will work for you!

RWMU: State and Session Information

- Don't keep state within the application server
- Key to being stateless: session data
  - Don't store it locally
  - The Web isn't session based, its request following requests
  - Store session data in the database!
  - Harness memcached
- Cookies are best validated by checksums and timestamps (encrypting is a waste of CPU cycles)

RWMU: Caching

- Not good for dynamic content, especially per user content (think modern Web applications)
- Cache full pages, all in application, and include the cookie (as the cache key)
- Use mod_cache, squid, and the Expires header to control cache times
- A novel way: cache **partial pages**!
  - pre-generate static page snippets, then bind them in with dynamic content into cached page

RWMU: Data Partitioning

- Replication is great for read heavy applications
- Write intensive applications should look at partitioning
- Partition with a global master server in mind
  - Give out global PKs and cache heavily (memcached)
  - It should also keep track of all the nodes with data
- Consider the notion of summary databases
  - Optimised for special queries like full-text search, or different latency requirements

RWMU: Blobs

- Large binary object storage is interesting
  - Image data is best kept in the filesystem, just use metadata in DB to reference server and path to filename
  - Try the Amazon S3 storage engine?
  - Store them in (MyISAM) tables, but split it so you don't have larger than 4GB tables
  - Metadata might include last modified date

RWMU: Misc. tips

- Unicode – use it
  - What's the most frequently used language in blogs?
  - http://dev.mysql.com/doc/refman/5.1/en/faqs-cjk.html
- Use UTC for time
  - Think about replication across geographical boundaries
- `sql_mode` might as well be strict
- Keep configuration in version control
  - Then consider puppet or slack for management of various servers

Getting the bleeding edge code

- We *still* use BitKeeper
- It is non-free software, and *very* expensive
- However, BitMover provides `bkf`, a free tool that allows cloning, and pulling updates
  - It doesn't allow committing code
- Our trees are public!
  - ... as long as the synchronisation doesn't break, they're also very up-to-date
- http://mysql.bkbits.net/

bkf 101

- `bkf clone bk://mysql.bkbits.net/mysql-5.0-community mysql-5.0-community`
  - clones the tree, to a local directory
- `bkf pull`
  - Updates the tree with the latest changes
- `bkf clone -rTAG bk://mysql.bkbits.net/mysql-5.0-community mysql-5.0-community-TAG`
  - replace TAG with mysql-5.0.45 or something, to get actual releases

Building MySQL 101

- Before making changes, build MySQL and ensure tests pass
- `BUILD/compile-dist`
  - builds mysql, as it would be built upstream
- `make test`
- `make dist`
  - source tarball generation
  - make dist --ignore ndb
- `scripts/make_binary_distribution`

Testing MySQL

- Use the **MySQL Sandbox**
- http://sourceforge.net/projects/mysql-sandbox
- Its really, MySQL in a one-click install
- `./express_install.pl mysql-5.0.45-linux-powerpc64.tar.gz`
- Check `~/msb_5.0.45` and run `./use`
- Linux/OSX only, sorry Windows folk
- Does not require root privileges, so can be run remotely on shell accounts, etc.

Resources

- MySQL Forge and the Forge Wiki
  - http://forge.mysql.com/
- MySQL Performance Blog
  - http://www.mysqlperformanceblog.com/
- Planet MySQL
  - http://planetmysql.org/
- #mysql-dev on irc.freenode.net
  - chat with developers, and knowledgeable community members

Thanks! Questions?

E-mail me:
colin@mysql.com
Catch me on IRC, at #mysql-dev:
ccharles