# MySQL Optimization

Tips and Tricks from the MySQL class

# Thanks To Dyn, Inc

- Cole's day job is at Dynamic Network Services, Inc (http://www.dynamicnetworkservices.com/)
- Got to go to Sun's MySQL Optimization class
- Did a presentation on some of the things learned for coworkers
- Was given permission to give the same presentation for LUG groups.

# Topics of Discussion

- General tips/tricks
- Data Types
- Indexes
- General Server Settings
- MyISAM vs InnoDB
- MyISAM
- InnoDB

# Next Topic

General Tips and Tricks

# General tips and tricks

- This all applies to 5.0, 5.1 – may not apply to 4.1
- Recommend 5.1 for new installations
- Version 4.0 EOL'd, 4.1 likely to be EOL'd by end of year

# Client safety trick

- i-am-a-dummy in .my.cnf, under [client]
  - Prevents updates/deletes without a where
  - Only 1000 rows returned from any query, unless limit by is specified
  - Can also be specified as the far less amusing ”safe-updates”

# Faster Dumping/Loading of Tables

Store and load data in tab delimited format

- `SELECT * FROM table_name INTO OUTFILE '/path/to/file'`
- `LOAD DATA INFILE '/path/to/file' INTO TABLE table_name`

# Next Topic

Data Types

# Help Choosing Data Types

- Requires a table to already be populated
- Do a: `SELECT <field(s) or *> FROM table PROCEDURE ANALYSE()`
- Won't work if the table is too big (takes too much memory)
- Won't work with i-am-a-dummy turned on (row limit prevents it)

# Fixing Misconceptions

- enums and sets are well behaved now (and more efficient for lookups than char/varchar(1))
- Using smaller int size for primary key is a lookup speed win (not just data size difference)

# Other considerations

- Always use `not null` if possible. Helps lookup speed and storage space
- Static width tables (char vs varchar, no text types) are faster for lookups ... only for MyISAM
- Store ips as unsigned int
  - Use inet_ntoa/inet_aton to access from libs
  - IPv4 only
  - Saves 11 bytes per row
  - Numeric (vs string) lookups – much faster

# Data compression

- Archive table engine
- Myisampack

# Archive table engine

- Supports insert/select, not update/delete
- Better compression than myisampack
- Don't have to go offline to convert (`ALTER TABLE table_name ENGINE=ARCHIVE`)
- Full table scans – no indexes supported
- Have to make sure it's supported/active with `SHOW ENGINES`. Not activated by default in MySQL 5.0

# myisampack

- Supported for MyISAM tables only
- Have to lock table completely (better to shut down db)
- Run shell command to pack, then another to fix indexes
- Table becomes read only
- Indexes still work

# Next Topic

Indexes

# Order matters

- Index on col_a, col_b, col_c
- Will **not** be used by: `SELECT ... WHERE col_b = x, col_c = y`
- **Will** be used by: `SELECT ... WHERE col_a = x, col_b = y` (partial index)
- **Will** be used by: `SELECT ... WHERE col_a = x order by col_b` (partial index)

# Covering indexes

- Happens when an index covers all fields selected as well as all fields in where clause
- Doesn't have to retrieve data – it's in the index
- If ”using index” shows up in ”Extra” field of an explain, we're using a covering index

# Index merge optimization

- Query against a single table can potentially use multiple indexes
- Examples:
  - `SELECT * FROM tbl_name WHERE key1 = 10 OR key2 = 20;`
  - `SELECT * FROM tbl_name  WHERE (key1 = 10 OR key2 = 20) AND non_key=30;`
- In ”Explain” output, shows up as ”index_merge” in the ”type” column

# Next Topic

General Server and Client settings

# Open Tables

- `show global status like 'open%tabl%'`
  - open_tables – tables currently open
  - opened_tables – how many times tables have been opened
- If opened_tables is always increasing, should probably consider upping the table_open_cache
- `show open tables`

# Query Cache

- Independent of storage engine
- Hash key is literal md5sum of unaltered query so case, order, spacing are all significant
- Use of user functions or stored procedures mean no qcache entry
- Functions like NOW(), RAND(), etc prevent using the qcache
- Prepared statements do not go in qcache

# Qcache Settings

- Query cache defaults to 'on' (query_cache_type var) but with a 0 size (query_cache_size var) – so no qcache by default!
- According to MySQL trainer, query_cache_size > 128MB typically starts making qcache not as useful
- `show global variables like 'query%'`
- `show global status like 'qc%'`

# Qcache Settings cont'd

- query_cache_limit (var) – maximum size of single result set in qcache
- query_cache_min_res_unit (var) – minimum block size for qcache results
- qcache_free_blocks (status) – Lots means fragmented query cache
- qcache_lowmem_prunes (status) – how many times queries had to be removed from cache because qcache was running out of memory

# Optimizing Qcache

- `flush query cache`
  - ”defrags” query cache
  - Does **not** empty query cache
  - Can take a few seconds to run
- `reset query cache`
  - Empties the current query cache

# Calculating qcache usage

- Compare these two:
  - `show global status like 'qcache_hits'`
  - `show global status like 'com_select'`
- Qcache hit rate is qcache_hits/ (qcache_hits+com_select)

# Thread caching

- Each client gets a thread, there is some minor overhead to creating a thread
- `show global status like 'thread%'`
- If threads_created variable increasing quickly, try upping thread_cache_size until it is increasing slowly or not at all

# max_connections

- Default is 100
- 500 – 1000 is reasonable for dedicated system
- Each connection requires file descriptor, memory for sort buffer, join buffer, read buffer, etc

# Sort Buffer

- Per client connection
- sort_buffer_size – memory allocated per connection (full size **always** allocated by each connection)
- Used by 'order by' and 'group by' second passes
- If data is too big for sort buffer, a  disk based sort is used

# Sort Buffer cont'd

- Things to watch to help decide on proper sort_buffer_size:
  - sort_merge_passes (status): number of merge passes performed – a higher number indicates needing to up sort buffer
  - sort_range (var) – displays sorts done using ranges
  - sort_rows (var) – the number of sorted rows
  - sort_scan (var) – number of sorts done by scanning the table (bigger is **bad**)

# Join/Read buffers

- join_buffer_size (var) – size of memory allocated when doing joins that don't use indexes.  A well designed SQL database makes very little use of this
- read_buffer_size (var) – Buffer for caching row data during full table scans.  Again, well designed databases will not use this much.

# Tmp tables

- Used whenever an explain mentions ”using tmp” in the extras column, often used for group by clauses, etc
- `show global status like '%tmp%tab%'` (gives an idea how often these go to disk instead of staying in memory)
- tmp_table_size (var) – max size of data for memory based tmp tables.  Bigger tmp tables will go to disk

# Next Topic

MyISAM vs InnoDB

# MyISAM Pros/Cons

## Pros

- Fast in many scenarios
- Smaller data storage
- Easy to backup

## Cons

- Data more easily corrupted
- No ACID compliance
- Cache only stores indexes
- Table locks

# InnoDB Pros/Cons

## Pros

- Row locking
- ACID compliance
- Cache holds indexes **and** data

## Cons

- Typically a little slower
- More difficult to back up
- More disk space

# Index comparison

## MyISAM

- B-Tree (links to data from nodes as well as from leaves)
- No auto balance
- `optimize table` will rebalance tree
- No good way to know when to rebalance

## InnoDB

- B+Tree (links to data from leaves only)
- Auto balances tree
- Additional automatic hash index on primary key

# Good uses for MyISAM Tables

- Logging apps
- Read only/mostly apps
- Bulk data loads, data crunching
- Read/write with low concurrency
- Data warehouses

# Good uses for InnoDB

- If reliability is required
- If consistent crash recovery and repair times are important
- If there is a performance benefit from the database caching writes itself (esp if there are lots of updates in small periods of time)
- High concurrency apps
- Where foreign keys or transactions are required

# Concurrent MyISAM/InnoDB use

- Making use of both MyISAM and InnoDB in a single database is non-optimal
- They can't share cache space
- The query optimizer can get confused on joins between tables of different engine types

# Next Topic

MyISAM Optimization

# Inserts into MyISAM

- Fixed row size vs variable row size
- Data stored in insertion order, except:
  - When there's deleted data, leaving empty blocks big enough to insert new row
  - When using the concurrent insert facility of MyISAM
- For faster population, set session var bulk_insert_buffer_size (defaults to 8MB), and do inserts using multiple values in single insert query. Also works with: `LOAD DATA INFILE ...`

# Concurrent inserts

- Table locks normally only allow 1 writer at a time
- Global var concurrent_insert.  Values:
  - 0 – (Off) Tells the server to not allow any concurrent inserts
  - 1 – (Default) Enables concurrent insert for MyISAM tables that don't have holes
  - 2 – Enable concurrent inserts for all MyISAM tables.  If the table has a hole and is in use by another thread the new row will be inserted at the end of the table.  If the table is not in use, it obtains a normal WRITE lock and inserts the new row into the hole.

# Key Cache (aka Key Buffer)

- Caches index data
- Defaults to a mere 16MB, ideal size would be the sum of 'du *.MYI'
- `show global status like 'key_read_requests'` – number of requests for index data
- `show global status like 'key_reads'` – how many requests had to go to disk
- Hit rate is (requests - reads) / requests – want this to be > 0.95 ideally

# Key Cache cont'd

- Make sure you keep at least a small (non zero) key_buffer_size, even if using primarily InnoDB because system tables are all MyISAM

# Multiple Key Caches

- Reduces key cache lock contention between threads
- Can specify which indexes go into which caches
- All caches share the total key_buffer_size
- Can preload indexes into a specific cache

# Recommended Strategy

- For busy databases, the following is recommended:
  - ”static data” cache: about 20% of total size for tables that are never (or rarely) modified
  - ”heavy update” cache: about 20% of total size for tables that are frequently modified and/or updated
  - ”default” cache: remaining 60% to use for everything else
- Use 'init_file' mysqld param to point at sql file that sets up these caches, and preloads any indexes into them

# Pet Idea

An idea I'd had was to keep indexes on a ram disk, providing the data set was small enough. Talking with the instructor in the class, he thought it would work.

Downside is that you have to rebuild indexes after every reboot.  For each table, have to:

- `repair table <table_name> use_frm;`
- `check table <table_name>`

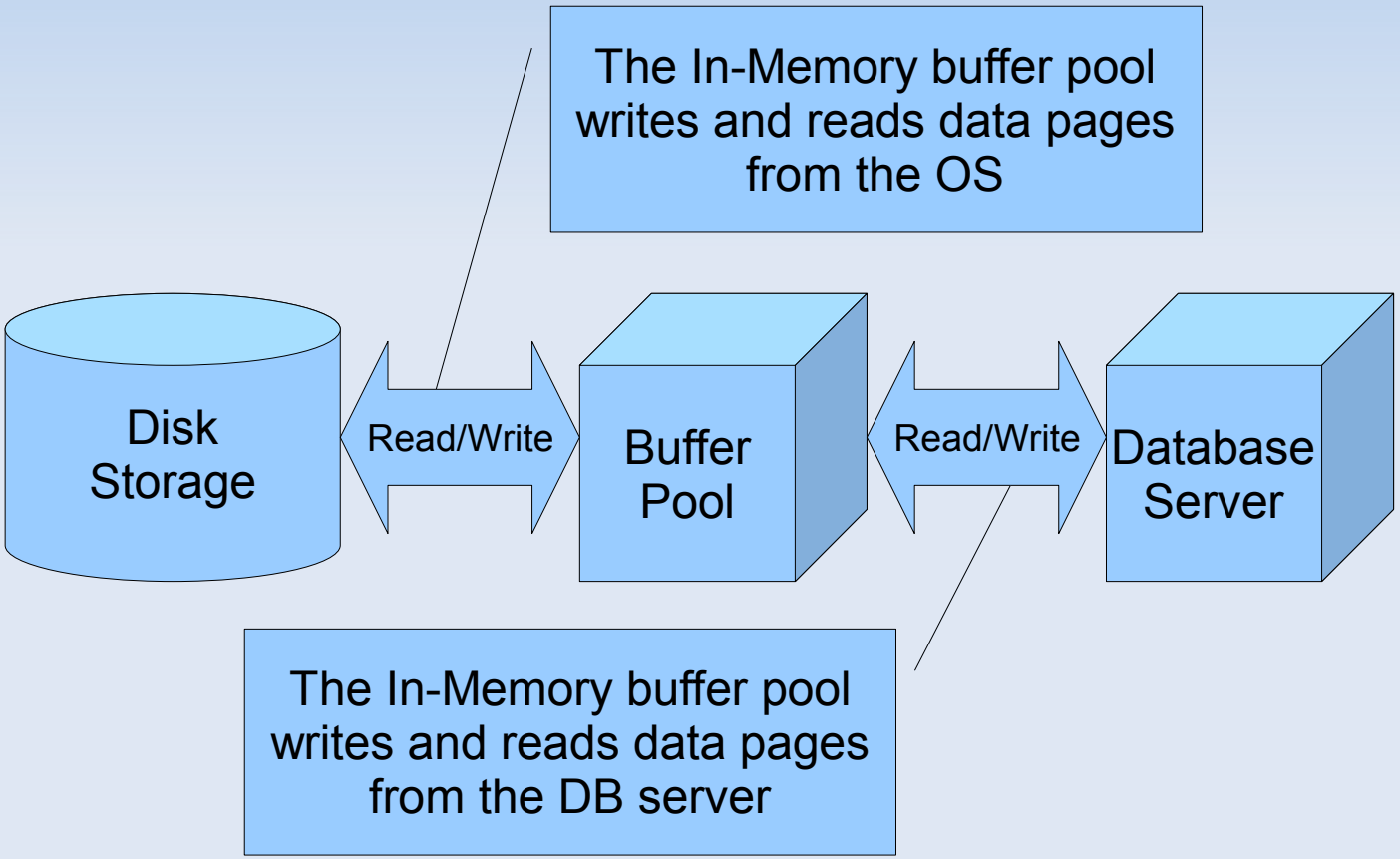
# Next Topic

InnoDB Optimization

# Things to Know about InnoDB

- Performance improved **significantly** between 4.1 and 5.0
- Do **not** let InnoDB run out of disk space. ”Bad things” will happen.
- Too many foreign keys can slow down a database.
- Optimize table only really needs to be run (at most) once a month on tables with lots of deletes. B+Trees auto rebalance.
- Mysqldump isn't the best backup strategy because reimport is **slow**

# Buffer Pool


The In-Memory buffer pool writes and reads data pages from the OS
Disk Storage
Read/Write
Buffer Pool
Read/Write
Database Server
The In-Memory buffer pool writes and reads data pages from the DB server

# Buffer Pool Setting

- Buffer pool caches data as well as indexes
- Ideally (if InnoDB exclusively) want to set innodb_buffer_pool_size to 60% to 80% of memory
- Don't set it so large that you start swapping

# Log File Size

- Bigger means more history, so more stability
- Bigger means less checkpoint file/io
- Bigger means longer recovery time on crash
- According to optimization instructor:
  - 64 MB log file – recovery time ”a few minutes”
  - 1 GB log file – recovery time ”an hour or so”
  - 256 MB considered good ”middle of the road”

# Log buffer

- Used to buffer writes to the InnoDB log files (used for repairing innodb tables on crash)
- innodb_log_buffer_size – how much data can be buffered from a single transaction before writing to disk
  - Defaults to 1MB
  - Set as high as 8MB if you have large transactions to reduce disk I/O
  - 4MB Seems to be good setting unless you are using large blobs

# InnoDB disk writes

- Log buffer writes to disk at COMMIT or at checkpoint
- Buffer pool writes to disk at checkpoints
- innodb_flush_log_at_trx_commit
  - 0: Logs flushed about 1/second
  - 1: Commit initiates flush
  - 2: log_buffer written to file, but only fsync'd every second

# Indexes

- B+Trees where index page is (default) 16KB
- InnoDB tries to leave 1/16 of page free for future inserts and updates
- Insertion into table in primary key order helps this
- Random order insertion can lead to pages only ½ full (innefficient)
- If fill factor is less than ½, InnoDB tries to contract the index tree.

# InnoDB best practices

- Use short, integer primary keys
- Load/Insert data in primary key order
- Increase log file size – if log file is full, all changes to buffer pool have to get written to disk immediately
- Avoid large rollbacks
- Avoid mass inserts – insert data in chunks

# One last thing about InnoDB

- innodb_additional_mem_pool_size
  - Size of the dictionary cache for meta data about tables.  Defaults to 1MB, almost **never** needs to be more than 8MB

# MySQL Optimization

The End