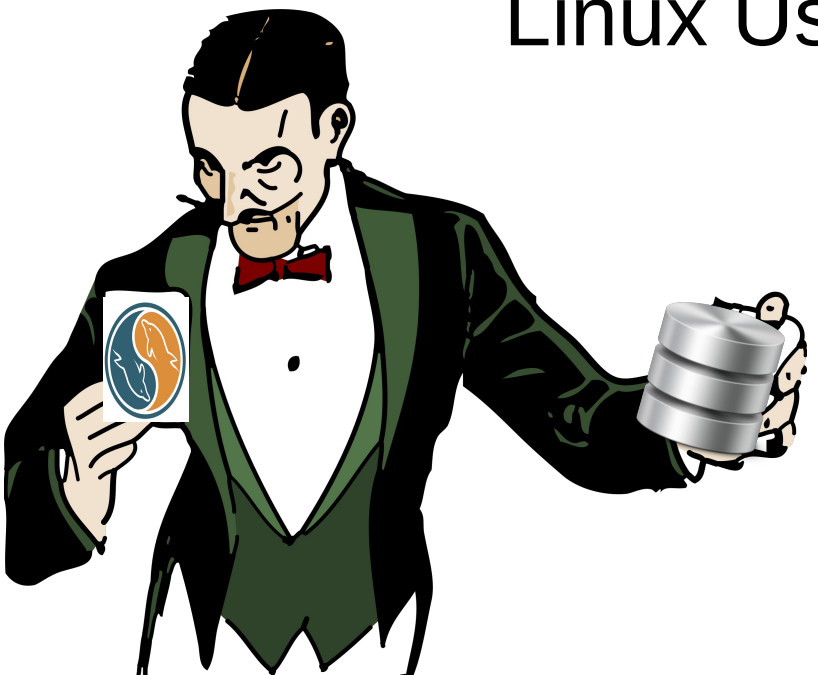
Some MySQL tricks: an open discussion

Bill Kendrick
CTO, Smashwords, Inc.

Linux Users' Group of Davis
2015-07-20



Preface

- *Database (DB)* = organized collection of data
- *Relational Database Management System (RDBMS)* = DB based on first-order predicate logic (Edgar F. Codd, 1969), used in mathematics, philosophy, linguistics, and computer science.
- *Structured Query Language (SQL)* = programming language for managing data in an RDBMS

Preface (cont'd)

- *MySQL* ('my s q l', or 'my sequel') = open source, SQL-based RDBMS, owned/sponsored by MySQL AB of Sweden, now owned by Oracle Corp. (It's also the “M” in “LAMP”)
  - #2 RDBMS after Oracle
  - #1 open source RDBMS
  - Initially released 1995
  - Linux, Windows, Mac OS X, FreeBSD, Solaris

Preface (cont'd)

- *Smashwords, Inc.* = self-serve ebook publishing & distribution platform
  - Launched in 2008
  - Nearly 360,000 books currently published (6/2015)
  - ~150,000 authors; plus small publishers & agents
  - ~1.5 million total users, when you add customers & affiliates
  - Browse, search & purchase at www.smashwords.com
  - Books ship to retailers & subscription services
    - Apple iBook Store, Barnes & Noble, Kobo, FlipKart, Scribd,Oyster, Sony (when they sold ebooks), and more, plus more always coming

You've come a long way, baby

- When I joined (2008), there were < 80 books published
- All services (web, ebook conversion, DB, email) ran on one small virtual server
- Things would get slow & break now & then
  - Growing pains (a good problem to have!)
- Today: load-balanced webservers, formal code review & deployment processes, automated server provisioning, master & slave DBs, increased use of “repository” design pattern, unit testing, and lots more
  - Goal: Keep the lights on, and authors & customers happy
  - How: Hired people smarter than I

Caveat

- I'm not an expert!
  - Things described here may or may not apply to your problems
  - Just here to share some of the interesting things we've learned along the way (that I can still remember)
  - I am not a DBA (database administrator) – can't answer configuration questions – I still consider myself a SQL rookie *(jack of all trades, master of none?)*
- I consider this talk an 'open discussion'; share your thoughts & experience with the rest of us, too! :)

Problem 1 – Can't ALTER that table

- I want to ALTER a table, but it has so many rows that it will take a long time
- DB is being accessed all the time by the website; locking it too long would break things
- Solution 1 – “Site maintenance” downtime
- Solution 2 – Percona Toolkit's “Online Schema Change” tool: ALTER tables without locking them
  - https://www.percona.com/doc/percona-toolkit/2.1/pt-online-schema-change.html

Problem 2 – More columns = bad

- You have users.facebook, users.twitter, now a new social network comes along. Don't add columns all over, normalize your data the RDBMS way!
  - CREATE TABLE social_network (
    user_id INT(10) UNSIGNED NOT NULL,
    network_id INT(10) UNSIGNED NOT NULL,
    url VARCHAR(256) NOT NULL DEFAULT '',
    PRIMARY KEY (user_id, network_id));
  - But what's “network_id?”...

Problem 2 – More columns = bad (cont'd)

- Add more networks whenever you want, no need to ALTER users, or ALTER this new social_network table to add more VARCHAR columns!
  - CREATE TABLE networks (
    id INT(10) NOT NULL AUTO_INCREMENT,
    name VARCHAR(64) NOT NULL DEFAULT '',
    PRIMARY KEY (id));
  - INSERT INTO networks (name) /* id will auto-inc! */ VALUES (“facebook”), (“twitter”), (“friendster”), (“orkut”);

Problem 2 – More columns = bad (cont'd)

- Add some network URLs for Joe (user id 1234):
  - INSERT INTO social_networks
    (user_id, network_id, url)
    VALUES
    (1234 /* Joe */, 2 /* twitter */, “http://www.twitter.com/joe”);
- What networks does Joe use?
  - SELECT networks.name, social_networks.url
    FROM social_networks
        JOIN networks
            ON networks.id = social_networks.network_id
    WHERE social_networks.user_id = 1234 /* Joe */;

Problem 3 – I want history AND fast results!

- You want a full history of rows for something going back forever, but *usually* you're only interested in the latest one.
  - e.g., for each shipment of a book, you want: “book 123 shipped to retailer 1 on YYYY-MM-DD”
  - Under *most* circumstances, you just want to know what the latest shipment is of a book to a particular retailer
    - or perhaps of *all* books to a particualr retailer
    - or perhaps of one particular book to *all* retailers
  - Sometimes (e.g. for auditing, debugging, customer support, etc.), you want the entire history (or some subset)

Problem 3 – I want history AND fast results! (cont'd)

- Create an historical log table
  - CREATE TABLE event_log (
    id INT(10) NOT NULL AUTO_INCREMENT,
    object_id INT(10) NOT NULL,
    event_id INT(10) NOT NULL,
    other_stuff VARCHAR(16) NOT NULL DEFAULT '',
    ts TIMESTAMP NOT NULL
    DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY(id));
- Put things in it when stuff happens:
  - INSERT INTO event_log (object_id, event_id, other_stuff)
    VALUES (123 /* book 123 */, 1 /* shipped to retailer 1 */, “blah”);

Problem 3 – I want history AND fast results! (cont'd)

- But whenever you do that, also keep track of the *latest* record in the log, in a “current” pointer table:
  - CREATE TABLE event_current (
    object_id INT(10) NOT NULL,
    event_id INT(10) NOT NULL,
    event_log_id INT(10) NOT NULL,
    PRIMARY KEY (object_id, event_id));
- You definitely want to do both the previous slide's INSERT INTO, and this slide's REPLACE INTO, within a transaction (atomic!)
  - REPLACE INTO event_current
    (object_id, event_id, event_log_id)
    VALUES
    (123, 1, LAST_INSERT_ID());

Problem 3 – I want history AND fast results! (cont'd)

- Get back the latest only:
  - SELECT event_log.*
    FROM event_current
        JOIN event_log
            ON event_log.id = event_current.event_log_id
    WHERE event_current.object_id = 123
    AND event_current.event_id = 1;
    - We'll get back the timestamp at the time we logged the event, along with that “blah” varchar
- Get back full history:
  - SELECT * FROM event_log
    WHERE object_id = 123 AND event_id = 1;

Problem 4 – Slow queries are slow

- Aside from smarter schema design, proper use of indexes (primary keys & otherwise), sometimes you need more
  - Replication database (“slave”)
  - Denormalization
    - Materialized views (DB tables)
    - Dropping JSON into TEXT cols.
    - “Baby-step” tables
  - Caching

Problem 4 – Slow queries are slow Part 1 - Replication

- Warning: I don't know how this is done, I've just benefited from it as “an end user”
  - (i.e., as developer writing queries in the application code)
- If the data you want doens't have to be up-to-the-millisecond, replicate it onto a server which is not constantly busy locking rows & tables due to INSERTs and UPDATEs
- Query that data from the so-called “slave” DB

Problem 4 – Slow queries are slow Part 2 - Denormalization

- Non-normalized schema; usually *bad*:
  - Users.id = 1
    Users.name = “Bob McKenzie”
    Users.country = “Canada”
    Users.facebook = “http://www.facebook.com/bob/”
    Users.twitter = “http://twitter.com/bobmc”
  - Users.id = 2
    Users.name = “Doug McKenzie”
    Users.country= “Canada” .. etc.
  - Books.id = 1
    Books.AuthorName = “Bob McKenzie” … etc.
  - Books.id = 2
    Books.AuthorName = “Bob McKenzie” … etc.
  - What if Bob changes his penname? :-(

Problem 4 – Slow queries are slow
Part 2 – Denormalization (cont'd)

- Normalized, looks better (but must JOIN tables a lot)
  - Users.id = 1
    Users.name = “Bob McKenzie”
    Users.country_id = 2
  - Networks.id = 1
    Networks.name = “Facebook” ...etc.
  - Social_networks.user_id = 1 /* Bob */
    Social_networks.network_id = 1 /* Facebook */
    Social_networks.url = “http://www.etc.”
  - Books.id = 1
    Books.author_id = 1 /* Bob McKenzie */ ...etc.
- Show all books, and their authors' names:
  - SELECT books.id, books.title, users.name AS author_name
    FROM books JOIN users ON users.id = books.author_id;

Problem 4 – Slow queries are slow Part 2a – Denorm. via Matviews

- Materialized views (matviews) are tables that contain the results of a query. Example: Yesterday's top 10 selling books
  - SELECT sales.book_id, books.title, users.name AS author_name
    FROM sales
    JOIN books ON books.id = sales.book_id
    JOIN users ON users.id = books.author_name
    WHERE sales.date =
        DATE_SUB(NOW(), INTERVAL 1 DAY)
    GROUP BY sales.book_id
    ORDER BY SUM(sales.qty) DESC LIMIT 10;
- Why run that every time someone hits a page?
- We could cache, but cache would expire & we'd likely run it many times per day.
  - We only need to calculate it ONCE, at around midnight! cronjob time!

Problem 4 – Slow queries are slow
Part 2b – Denorm. via JSON

- *JavaScript Object Notation (JSON)* = lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.
  - {
    “book_id”: 1,
    “author”: {
      “name”: “Bob McKenzie”,
      “country”: “Canada”
    },
    “title”: “Mutants of 2051 A.D.: The Novel”,
    “pubdate”: “1983-08-26”,
    “description”: “I was kinda like a one man force eh, like Charlton Heston in Omega Man. Did ya see it? It was a beauty!”,
    “formats”: [
      “epub”: true,
      “pdb”: false
    ]
  }
- Language-independent (not specific to JavaScript). Your PHP, Ruby, & Go code can all convert the JSON into internal data structures!

Problem 4 – Slow queries are slow Part 2c – Denorm. via “baby steps”

- Q: What are our top 100 selling books in Romance category, under Paranormal, Detective and Historical subcategories, published in the last year, which are part of a series that has a free series starter (book one), based on sales in the last 6 months across all retail channels? I want to make a blog post about them!
  - You could write one massive, multi-JOIN query that crashes the database.
  - You could dump a ton of data into tab-separate spreadsheets and throw it at a script written in the *R* statistical language
  - You can break it into bite-sized pieces (“baby steps”)

Problem 4 – Slow queries; Part 2c – Denorm. via “baby steps” (cont'd)

- Q: What are our top 100 selling books in **Romance category, under Paranormal, Detective and Historical subcategories**, **published in the last year**, which are part of a series that has a free series starter (book one), based on sales in the last 6 months across all retail channels? I want to make a blog post about them!
  - /* Find all books in the categories we care about, pub'd in the last year */
    CREATE TABLE analysis.romance_books AS
    SELECT book_category.book_id
    FROM book_category
    JOIN books ON books.id = book_category.id
    WHERE book_category.category_id IN (123, 456, 789)
    AND books.published = TRUE
    AND books.pubdate >= '2014-07-20';

Problem 4 – Slow queries; Part 2c – Denorm. via “baby steps” (cont'd)

- Q: What are our top 100 selling books in Romance category, under Paranormal, Detective and Historical subcategories, published in the last year, **which are part of a series that has a free series starter (book one)**, based on sales in the last 6 months across all retail channels? I want to make a blog post about them!
  - /* For all the books we found, find the series any of them are in */
    CREATE TABLE analysis.romance_books_series AS
    SELECT DISTINCT(book_series.series_id) AS series_id
    FROM analysis.romance_books AS rombook
    JOIN book_series ON book_series.book_id = rombook.book_id;
  - /* For each series, look at its “book one”; get all series where it's a free book */
    CREATE TABLE analysis.romance_books_free_starter_series AS
    SELECT romseries.book_series_id
    FROM analysis.romance_books_series AS romseries
    JOIN book_series AS book1_series
        ON book1_series.series_id = romseries.series_id
        AND book1_series.book_number = 1
    JOIN books ON books.id = book1_series.bookid
    WHERE books.price = 0.00;

Problem 4 – Slow queries; Part 2c – Denorm. via “baby steps” (cont'd)

- Q: What are our top 100 selling books in Romance category, under Paranormal, Detective and Historical subcategories, published in the last year, **which are part of a series that has a free series starter (book one)**, based on sales in the last 6 months across all retail channels? I want to make a blog post about them!
  - /* Consider only those which are part of series w/ a free “book one” */
    CREATE TABLE analysis.romance_books_eligible AS
    SELECT rombook.book_id
    FROM analysis.romance_books AS rombook
    JOIN book_series ON book_series.book_id = rombook.book_id
    JOIN analysis.romance_books_free_starter_series AS fs_series
        ON fs_series.series_id = book_series.series_id;

Problem 4 – Slow queries; Part 2c – Denorm. via “baby steps” (cont'd)

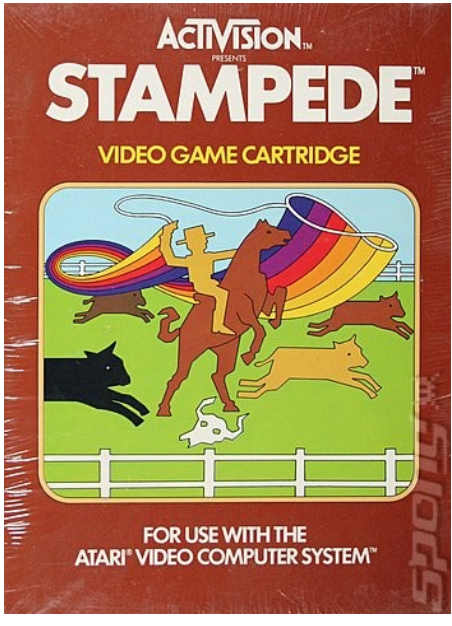
- Q: **What are our top 100 selling books** in Romance category, under Paranormal, Detective and Historical subcategories, published in the last year, which are part of a series that has a free series starter (book one), **based on sales in the last 6 months across all retail channels?** I want to make a blog post about them!
  - /* Rank the top 100 eligible books */
    CREATE TABLE analysis.romance_books_topsellers AS
    SELECT rombook.book_id, books.title, SUM(sales.qty) AS qty
    FROM analysis.romance_books_eligible AS rombook
    JOIN books ON books.id = rombook.book_id
    JOIN sales ON sales.book_id = rombook.book_id
    AND sales.date >= '2015-01-20'
    ORDER BY SUM(sales.qty) DESC LIMIT 100;

Problem 4 – Slow queries are slow Part 3 – Caching

- Warning: I'm not versed in setting up backends for this stuff. Again, just an “end user”.
- *Memcached* = Open source, high-performance, distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load. An in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.
- Set up your memcache, then use it to store results of queries. Run fewer queries, use your memcache (can be on a separate server!) to store things.
- function get_book_json($bookid) {
  $foo = memcached_get(“book_json:$bookid”);
  if ($foo == NULL) {
  $foo = db_query(“SELECT json_blob FROM book_json WHERE id = $bookid”);
  memcache_set(“book_json:$bookid”, $foo);
  }
  return ($foo);
  }

Problem 4 – Slow queries are slow
Part 3 – Caching (cont'd)

- What if it's a slow query? And what if it's on a popular page?
- Multiple invocations of your code will:
  - Check memcache
  - Not find anything
  - Invoke the slow query
- You're no better off than you were before!
- It becomes a cascading failure: “cache stampede”



Problem 4 – Slow queries are slow
Part 3 – Caching (cont'd)

- Use a *semaphore* (“variable or abstract data type that is used for controlling access, by multiple processes, to a common resource in a concurrent system”) to lock things down while the first process runs the query; other processes will wait for it to finish.
- function query_with_cache($key, $sql) {
  $lock_key = $key . “_lock”;
  $results = memcached_get($key);
  if ($results == NULL) {
    if (memcached_test($lock_key)) {
      /* Someone else is already running the query, just wait for the results to get saved to cache */
      while (memcached_test($lock_key) && $results == NULL) {
        sleep(1);
        $results = memcached_get($key);
      }
    } else {
      /* I'm the first to notice it's not cached, so create a semaphore, run the query, & save results to cache */
      memcache_set($lock_key, “xxx”);
      $results = db_query($sql);
      memcache_set($key, $results);
      memcache_remove($lock_key);
  }
  return ($results);
}

h/t: http://davedevelopment.co.uk/2012/01/13/defending-against-cache-stampedes.html

Fin

Thanks!

Time to discuss, Q&A, etc.!?