The MySQL Ecosystem at Scale

Jeremy Cole
Sr. Systems Engineer - SRE
Google Inc.

Jeremy Cole
@jeremycole

“Making MySQL Awesome” at Google
Worked at MySQL AB 2000-2004
Contributor since 3.23
Over 14 years in the MySQL community
Code, documentation, research, bug reports
Yahoo!, Proven Scaling, Twitter
Built a MySQL 5.5 fork at Twitter
Attended XLDB many times but haven’t spoken before

About this talk

Not really about Google, per se
Not academic or scientifically focused
Pragmatic, from industry experience only
Imperfect and non-ideal world

MySQL’s roots
High-scale usage scenarios
Strengths and weaknesses at scale
State and future of the MySQL ecosystem

Databases in industry

Always online, no downtime
Low risk or carefully managed risk from operations

Migration is the hardest part of any change
No downtime, minimal impact from changes
Usually 50-step online migration, not 2-step downtime
Rollback must also be online

Being up is much more important than being right
The business is more important than good database principles

Databases are fun

Until you use them...

A bit of MySQL history

A short history of the MySQL software

1994: Development started; some roots already present
2000: 3.23 + InnoDB, replication
2002: 4.0 + replication redesign, set operations
2004: 4.1 + subqueries
2005: 5.0 + stored procedures, views, triggers, XA
2008: 5.1 + partitioning, row-based replication
2010: 5.5 + stability, code cleanup, InnoDB scalability
2013: 5.6 + InnoDB scalability, performance, manageability

The MySQL commercial landscape

2003: Alzato (MySQL Cluster) acquired by MySQL
2005: Innobase Oy (InnoDB) acquired by Oracle
2006: Percona founded
2008: MySQL AB/Inc. acquired by Sun
2009: Monty Program (MariaDB) founded
2010: Sun acquired by Oracle
2010: SkySQL founded
2013: Monty Program acquired by SkySQL

2011

MySQL declared a
“fate worse than death”
by Mike Stonebraker

2013

MySQL still running most of the web,
including Twitter and Facebook
and Google and ...

MySQL

MySQL wins

Pretty fast, usually (<500µs for typical reasonable queries)
Very robust data storage layer (InnoDB)
Replication that usually works (or is at least well understood)
Easy to use and easy to run

Hmm!

A random server we came across at Twitter:

Uptime: 212d 11h 16m
Questions: 127481750624
(127 billion, or 6,943 per second)
Innodb_rows_read: 24989035721780
(24.9 trillion, or 1.36M per second)

MySQL loses

Really bad for ID generation at scale (meh auto-increment)
Not good by itself for graph data -- need software on top
Replication inefficiency sucks for busy OLTP (meh lag)
I value stability and performance over fancy new features. Oracle doesn’t always feel the same way.

MySQL’s happy place

Use it as-is for smaller datasets (<= 1.5TB)
Use as a permanent backing store for larger datasets
Build on top of it to add the features that are broken or missing
Happy place is expanding a bit with 5.5, 5.6

The MySQL ecosystem

Oracle MySQL

Official and “most upstream” version of MySQL.
Continuing to do good development, but often without much public visibility until release.
Ignores bugs, feedback, communication from community.

5.5 is stable and in wide usage.
5.6 is newly GA and not widely used yet.
5.7 is in active development.

Percona Server

Strictly downstream from Oracle MySQL.
Series of patches applied on top of a given MySQL release.
Many changes *eventually* end up in Oracle MySQL, but it can take several years.
Always innovating on MySQL, but some changes and features can be pretty risky and/or dangerous.
Quick to fix their mistakes. :)

MariaDB

Started by Monty as a non-Oracle-owned alternative.
Lots of original MySQL developers working on it.
Initially a new storage engine (Maria/Aria).
Later, a full fork with active development of most aspects.
Aiming to be compatible with Oracle MySQL wherever possible.
5.5 is downstream from Oracle MySQL 5.5.
10.0 is a full fork, generationally equivalent to Oracle MySQL 5.6.

In-house development forks

Not really true “forks” -- branches for internal use by each company, not intended for external consumption in whole.
Published as a robust communication mechanism for working code and discussion of features and directions.
Some features make it upstream.

Google was perhaps the first with MySQL 4.0 fork, but now:
Google (4.0-5.1, MariaDB 10.0)
Facebook (5.1, 5.6)
Twitter (5.5)

Why do in-house MySQL development?

Absolute control over development of minor features and especially bug fixes. Get a fix made and out in days, not months.

Roadmap planning for major features required for future business requirements.

Ability to be make internal bug fix releases, with exactly one bug fix, and being able to deploy it very quickly to production with very low risk (e.g. Twitter’s 5.5.23.t6.1 to fix a deadlock issue).

Usage scenarios

Small: One master, many slaves

Typical configuration for many companies
Read traffic can scale with slave count
Write traffic is limited to a single master
Modern machines with SSDs, the limits are not low anymore

Bad: Divide and conquer with master-slave

Typically when limits of single master are reached.
Naive approach moves some entire tables to other master-slave clusters on separate hardware.
Very labor intensive and limited success.
No transactions across (arbitrary) boundaries.
A mess of code to maintain.

Enter:
Partitioning of data

aka “Sharding”

Bad: Fixed range or hash partitioning

“Users 1-100 in DB A, 101-200 in DB B, …”
“Users id % 8 == 0 in DB A, id % 8 == 1 in DB B, …”

Often the next “brilliant” idea when dividing tables fails. Scalability is very good for fixed data sets, but growth is challenging and generally not in-place.

Good: Dynamic directory-based partitioning

An additional database stores metadata about data location.
Often hash-based partitioning with many shards (thousands).
Typically uses “virtual shard” or “bucket”, but may track location of individual user/key.

Implementations:
Twitter: Gizzard
Google/YouTube: Vitess
Many many others...

Sharding library availability

Companies are mostly building their own internal sharding systems.
Sharing this code is difficult: it is very critical to the business and often written to use internal-only libraries and features.
But, usually not really necessarily proprietary.
It may be impenetrable to others due to complexity or domain-specific problems. (See: Gizzard and Vitess and ...)
It may be re-architected to meet needs of the company without consulting any community.

MySQL is not magic

Some (especially commercial) RDBMSes claim to be magic, but are they really? Really?

MySQL is not free

Real work is often required in the real world.

Questions?