How YouTube Supports Billions of Users with MySQL and Vitess

BY TEBY TEGO
JUN 29, 2025

Postgres for Agentic AI—Now in the Cloud (Sponsored)

If you’re building LLM-powered features, you don’t need another black box. pgai on Timescale Cloud gives you full control over your vector data, memory, and retrieval logic—inside PostgreSQL. Everything runs in one place, with SQL and the tools your team already uses. From prototype to production, it's built to scale with you.

Explore pgai on Timescale Cloud

Disclaimer: The details in this post have been derived from presentations and blogs written by the YouTube and Vitess engineering teams. All credit for the technical details goes to the YouTube and Vitess engineering teams. The links to the original articles and videos are present in the references section at the end of the post. We've attempted to analyze the details and provide our input about them. If you find any inaccuracies or omissions, please leave a comment, and we will do our best to fix them.

As YouTube's popularity soared, so did the complexity of its backend. This happens because every video upload, comment, like, and view creates more data.

Initially, a single MySQL database and a few web servers were enough to keep the site running smoothly. But as the platform evolved into a global giant with billions of daily views, this approach began to crumble under the weight of its success.

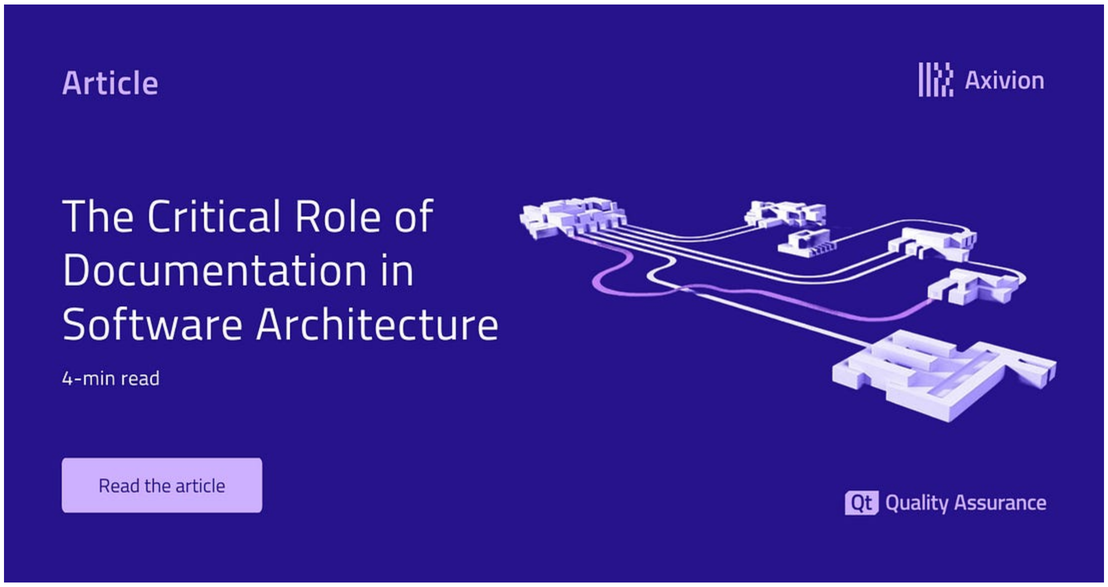
YouTube built a custom solution for managing and scaling MySQL. This was known as Vitess.

Vitess acts like a smart librarian in a massive library. Instead of letting everyone go through the shelves, it organizes requests, routes them efficiently, and ensures that popular books are available at the front.

In other words, Vitess isn't a replacement for MySQL. It's more like a layer on top of it, allowing YouTube to continue using the database system they were already familiar with while enabling horizontal scaling (adding more servers) and graceful traffic handling. By introducing Vitess, YouTube transformed its backend into a more intelligent, resilient, and flexible infrastructure.

In this article, we will look at the various challenges YouTube faced as part of this implementation and the learnings they derived.

The Critical Role of Documentation in Software Architecture (Sponsored)



Good documentation isn't just nice to have, it's essential for building scalable, maintainable software.

From improving team collaboration to reducing technical debt, well-structured architecture documentation delivers long-term value. Investing in documentation today means fewer headaches tomorrow.

In this 4-min read, you'll learn how to create documentation that truly supports your development teams.

Read the article

How do Web Apps Typically Scale?

When developers first build a web or mobile application, setting up the backend is fairly straightforward. Typically, they spin up a MySQL database and connect it to their application via a few web servers. This setup works wonderfully in the beginning. It’s simple, well-documented, and allows the team to move quickly with the features that matter.

In this early phase, the database footprint is small. Users submit data (like account info or posts) and retrieve it as needed. With low traffic, everything runs smoothly. Reads and writes are fast, and backups can be taken by temporarily pausing the application if needed.

However, success introduces complexity. As the application becomes more popular, more users begin to interact with it simultaneously. Each read or write adds to the load on the single MySQL instance, which can lead to slowdowns and an unresponsive database.

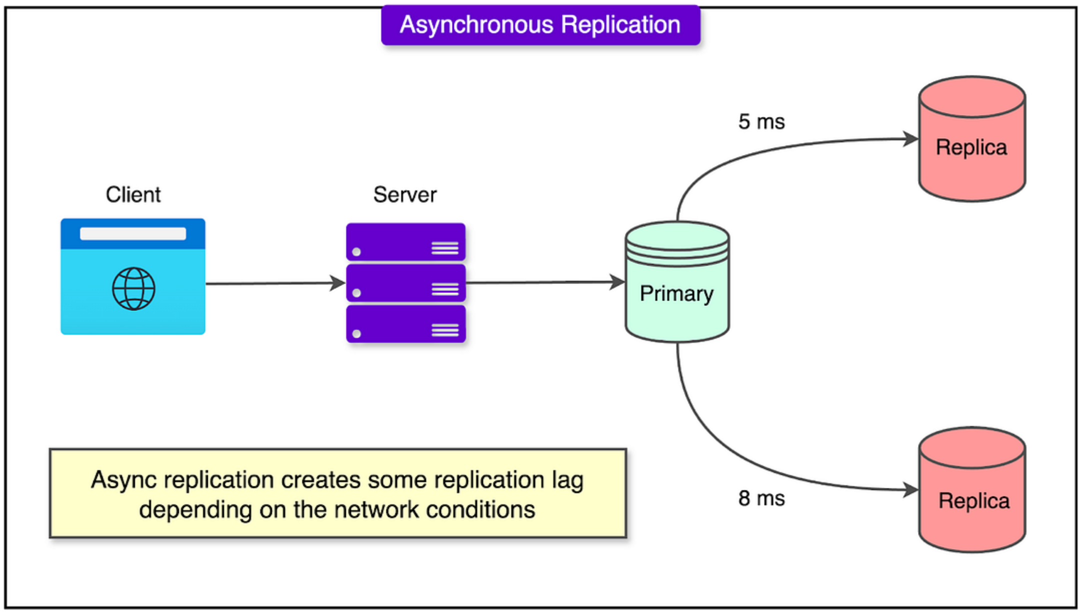
Some of the first problems to appear include:

- Slow queries due to growing data volume.
- Downtime while performing backups or updates.
- Risk of data loss if a single server fails.
- Limited ability to serve users globally due to latency.

Reading from Replicas

As web applications scale, one of the first techniques used to handle increasing load is replication: creating one or more copies (replicas) of the main database (often called the primary). These replicas are kept in sync with the primary by copying its data changes, typically through a process called asynchronous replication.

See the diagram below:


Asynchronous Replication
Client
Server
Primary
5 ms
Replica
8 ms
Replica
Async replication creates some replication lag depending on the network conditions

The main advantage of using replicas is load distribution.

Instead of having every user query hit the primary database, read queries (such as viewing a video, profile page, or browsing a list) are sent to the replicas. This reduces the load on the primary and helps the system handle a much higher volume of requests without degradation in performance.

However, this setup introduces a crucial trade-off: data staleness.

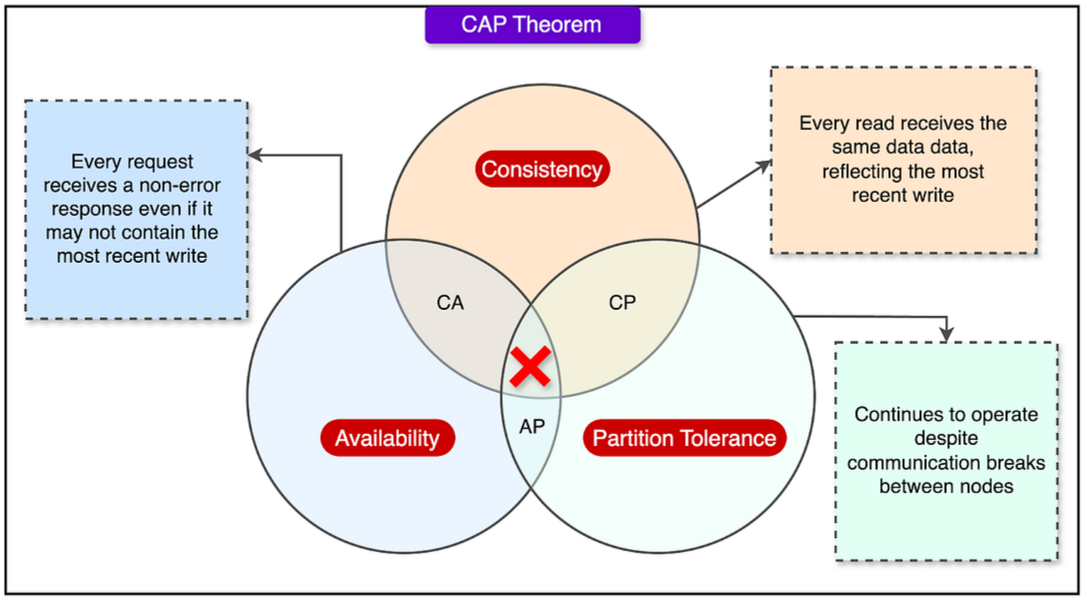
Since replicas receive updates from the master with a delay (even if just a few seconds), they may not always reflect the most current data. For example, if a user updates their profile and immediately refreshes the page, a replica might still contain the old information.

Let's look at how YouTube handled this scenario.

Balancing Consistency and Availability

As YouTube grew, it faced a fundamental challenge in distributed systems: the CAP theorem.

This principle states that in the event of a network issue, a system can only guarantee two of the following three properties: Consistency, Availability, and Partition Tolerance.


CAP Theorem
Every request receives a non-error response even if it may not contain the most recent write
Consistency
Every read receives the same data data, reflecting the most recent write
CA
CP
AP
Availability
Partition Tolerance
Continues to operate despite communication breaks between nodes

Partition tolerance is a must for distributed systems, so engineers are left choosing between consistency and availability.

YouTube, like many large-scale platforms, chose to make trade-offs, sacrificing strict consistency in some areas to maintain high availability.

They did this by classifying different types of read operations with a dual strategy:

- **Replica Reads**: Used when absolute freshness isn't critical. For example, displaying a video or showing view counts doesn't require second-to-second accuracy. A few seconds of delay in updating the view count won't harm user experience.
- **Primary Reads**: Reserved for operations that require up-to-date data. For example, after a user changes their account settings, they expect to see those changes reflected immediately. These read

operations are directed to the primary, which has the most current data.

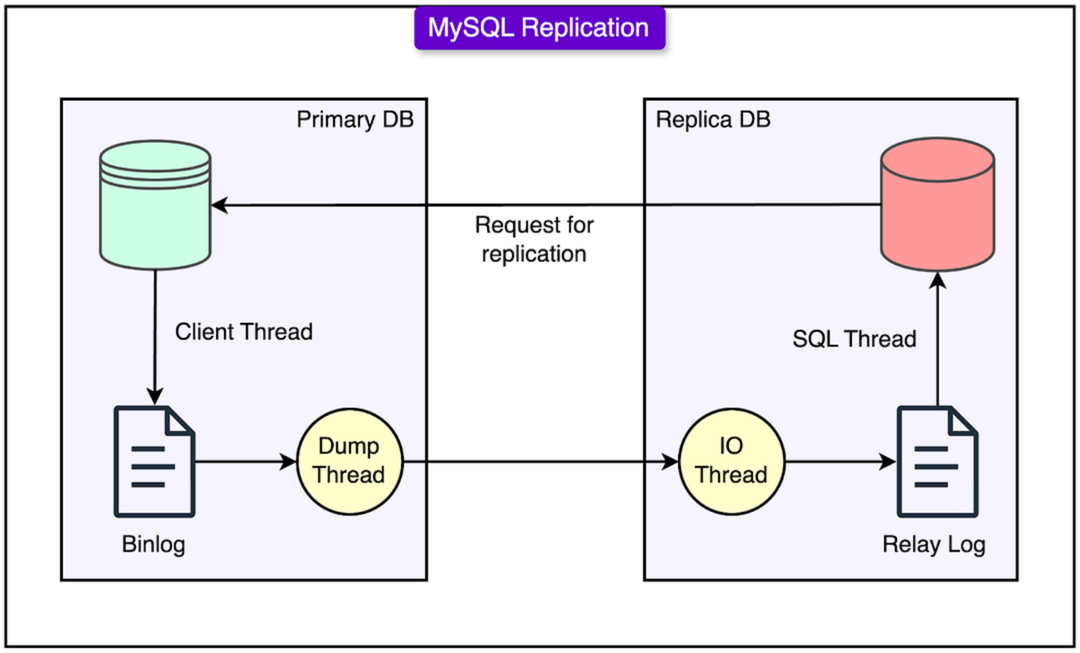
The underlying idea was that not all read requests are equal. Some data must be fresh, but a lot of content can tolerate a slight lag without negatively impacting the user experience.

Write Load Challenges and the Role of Prime Cache

As YouTube's traffic surged, so did the number of write operations to the database: uploads, comments, likes, updates, and more. This increase in write queries per second (QPS) eventually made replication lag a serious problem.

MySQL's replication process, especially in its traditional form, is single-threaded. This means that even if the primary database handles many write operations efficiently, the replicas fall behind because they process changes one at a time. When the volume of writes crosses a certain threshold, the replicas can't keep up. They begin to lag, causing stale data issues and inconsistencies.

See the diagram below that shows the MySQL replication process.



To address this, YouTube engineers introduced a tool called Prime Cache.

Prime Cache reads the relay log (a log of write operations that replicas use to stay in sync with the primary). It inspects the WHERE clauses of upcoming queries and proactively loads the relevant rows into memory (cache) before the replica needs them.

Here’s how it helps:

- Normally, a replica processes a write operation and must fetch data from a disk as needed.
- Prime Cache pre-loads the necessary data into memory, turning what was previously disk-bound work into memory-bound operations.
- As a result, the replication stream becomes faster because memory access is much quicker than disk access.

This optimization significantly improved replication throughput. Replicas were now able to stay more closely in sync with the primary, even under high write loads.

While Prime Cache wasn’t a permanent fix, it allowed YouTube to handle a much higher volume of writes before needing to implement more complex scaling strategies like sharding.

Sharding and Vertical Splitting

As YouTube's backend continued to grow, even optimized replication couldn’t keep up with the sheer scale of the data. The size of the database itself became a bottleneck. It was too large to store efficiently on a single machine, and the load was too heavy for any one server to handle alone.

To address this, YouTube adopted two complementary strategies: vertical splitting and sharding.

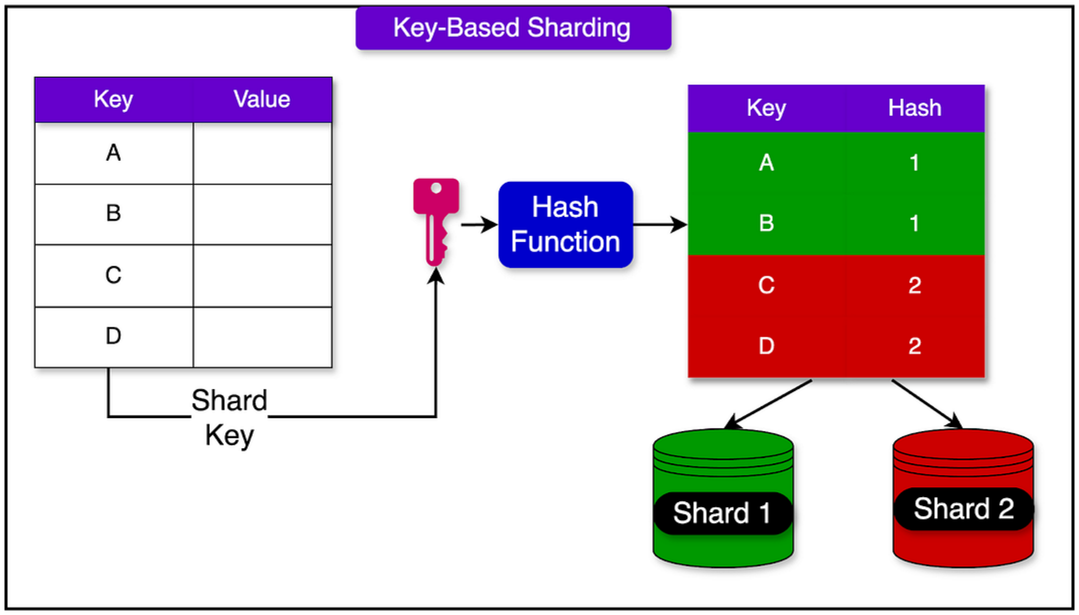
Vertical Splitting

Vertical splitting involves separating groups of related tables into different databases.

For example, user profile data might be stored in one database, while video metadata is stored in another. This reduces the load on any single database and allows independent scaling of different components.

Sharding

Sharding takes this a step further by dividing a single table’s data across multiple databases, based on some key, often a user ID or a data range. Each shard holds only a portion of the overall data, which means that write and read operations are spread across many machines instead of one.



Sharding comes with some trade-offs as well:

- Transactions that span multiple shards are difficult to coordinate, so strong guarantees like atomicity and consistency across shards are often sacrificed.
- Queries that need data from multiple shards or tables may no longer work as expected.
- The application must now know how to route queries to the correct shard and how to handle cross-shard queries when needed.
- For certain types of queries that span shards, special indexing and synchronization logic are required.

To cope with this complexity, YouTube's client logic evolved significantly. It was no longer enough to simply connect to a database and send queries. The client now needed to determine whether a read should go to a replica or the master, decide which shard a query should be routed to, based on the query’s WHERE clause, and maintain and update cross-shard indexes where needed.

This shift placed more responsibility on the application layer, but it also enabled YouTube to scale far beyond the limits of a single MySQL instance.

One of the most powerful features of Vitess is its ability to automate sharding. Here’s how automatic sharding works in Vitess:

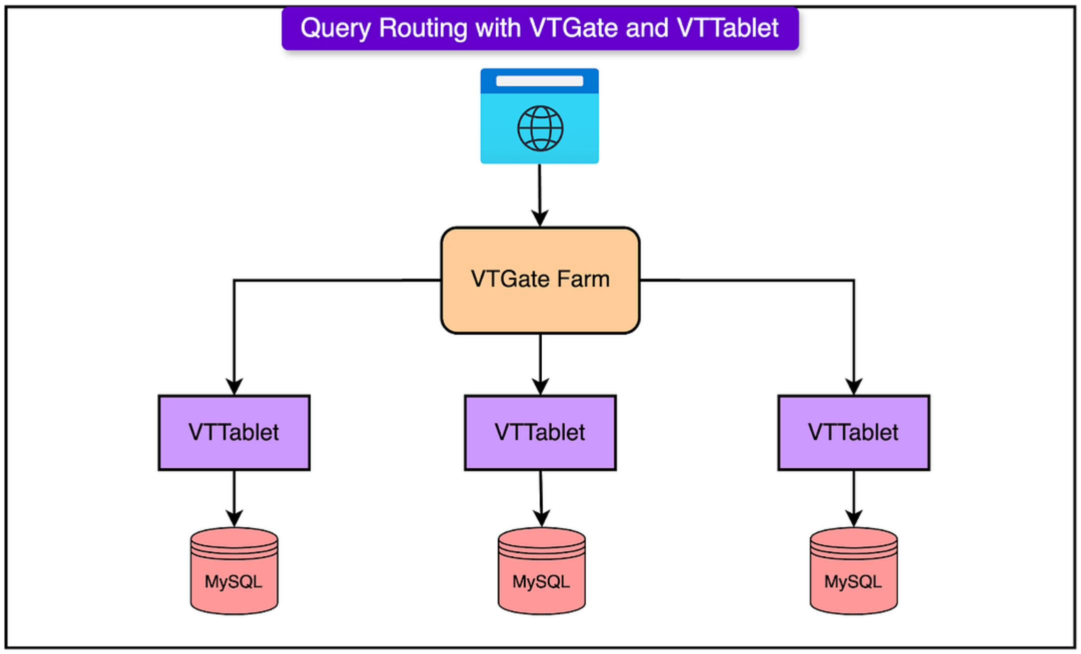
- An engineer specifies that an existing shard (for example, one with too much data or traffic) needs to be split. This might be from 1 shard into 4, 16, or more.
- In the background, Vitess sets up new MySQL instances, initializes them with the schema, and begins copying data.
- Engineers can check the progress using Vitess tools. Once data has been moved and validated, the system signals readiness.
- When everything is in place, the engineer authorizes a traffic switch. Vitess automatically redirects queries to the new shards and updates its metadata.
- Once confirmed, the original overburdened shard is phased out of production.

The process is built to minimize downtime and manual intervention.

Query Routing with VTTablet and VTGate

In a sharded database environment like YouTube's, sending SQL queries to the right database instance becomes a challenge.

Vitess solves tchallenge with two key components: VTTablet and VTGate.



VTGate is a query router. It acts as the main entry point for all application queries. When an application sends an SQL statement, it doesn't need to know which shard to talk to or whether the data lives in one table or across several databases. VTGate handles all of that logic.

VTTablet sits in front of each MySQL instance (each shard). It acts as a lightweight proxy, but with several advanced capabilities:

- **Manages connection pooling**: Rather than allowing thousands of direct connections to MySQL, which would overwhelm it, VTTablet maintains a limited set of pooled connections and serves multiple app queries efficiently.
- **Query safety checks**: VTTablet inspects every incoming query. If a query is missing a LIMIT clause or is likely to return an excessive number of rows, VTTablet may block it or return an error.
- **Performance management**: VTTablet tracks how long queries run and can kill long-running or resource-intensive ones.
- **Data validation and caching**: It interfaces with the row cache, handles invalidations, and ensures data consistency without hitting

MySQL for every request.

Vitess uses its own SQL parsers in both VTGate and VTTablet to understand the structure and intent of each query. While it doesn’t support every edge-case feature of MySQL, it covers nearly all common SQL syntax used in typical applications.

Reparenting and Backups in Vitess

Soon, the YouTube engineering team had to deal with the complexity of managing thousands of database instances.

Manual operations that once took a few minutes became risky and time-consuming. Even small mistakes (like misconfiguring a replica or pointing it to the wrong primary) could cascade into major outages.

Vitess was designed to automate many of these routine but critical database management tasks, such as reparenting and backups.

Reparenting

Reparenting is the process of promoting a replica to become the new primary if the current primary fails or needs to be taken offline.

Without automation, reparenting is a multi-step, manual process that involves the following steps:

- Identify the failure.
- Promote a suitable replica.
- Update all other replicas to follow the new primary.
- Reroute application traffic.

Even if each step takes only a few seconds, the total time can be significant. Worse, human error during this sensitive process can lead to data inconsistencies or prolonged downtime.

Vitess simplifies reparenting through its orchestration layer, powered by a lock server and specialized workflow components.

Backup Management

Vitess also automates backups. Rather than requiring administrators to manually bring down a server and extract data, Vitess tablets can initiate and manage backups on their own. They can perform these tasks without interrupting service, thanks to the separation between primary and replica roles.

This automation is critical at scale. With potentially thousands of database instances across many data centers, manual backups and recovery are not just inefficient, they’re impractical and error-prone.

Core Vitess Features That Helped YouTube Scale

Below is a detailed breakdown of the features that helped Vitess serve the needs of YouTube in terms of scaling.

1 - Connection Pooling

In MySQL, each client connection consumes memory. At YouTube's scale, with tens of thousands of simultaneous user requests, allowing each web server to directly connect to MySQL would quickly exhaust server memory and crash the system.

Vitess solves this with connection pooling, managed by VTTablet:

- All incoming connections are handled by a smaller pool of MySQL connections.
- This prevents memory exhaustion and reduces the load on the MySQL server.

- When a new master comes online (for example, after failover), the system rapidly reconnects and resumes full service without downtime.

2 - Query Safety

In large teams, even well-intentioned developers can accidentally write inefficient queries. Vitess implements multiple safety mechanisms:

- **Row limits**: If a query lacks a LIMIT clause and risks returning an enormous dataset, VTTablet automatically adds a limit or blocks the query.
- **Blacklisting**: Administrators can blacklist problematic queries so they are never executed.
- **Query logging and statistics**: Vitess logs all query behavior, including execution time, error frequency, and resource use. This data is critical for detecting misbehaving queries early.
- **Timeouts**: Long-running queries are automatically killed to prevent resource hogging.
- **Transaction limits**: Vitess enforces a cap on the number of open transactions, avoiding MySQL crashes from transaction overload.

3 - Reusing Results

In high-traffic environments, popular queries can become hotspots. Thousands of users might request the same data at the same time. In vanilla MySQL, each query would be independently executed, increasing CPU and disk usage.

Vitess handles this intelligently. When a request arrives for a query that is already in progress, VTTablet holds the new request until the first one is completed. Once the initial result is ready, it is shared across all waiting requests.

4 - Vitess Row Cache vs MySQL Buffer Cache

MySQL's buffer cache loads 16KB blocks from disk into memory, regardless of how many rows are needed. This works well for sequential reads but performs poorly for random-access patterns, which are common in modern web apps.

Vitess implements its row-level cache, optimized for random access:

- It uses memcached to cache individual rows by primary key.
- When a row is updated, the cache is invalidated.
- If the system is operating in replica mode, Vitess listens to the MySQL replication stream and uses it to invalidate cached rows in real-time.

This means the cache stays accurate without relying on manual expiry times and delivers faster responses for frequently accessed records.

5 - System Fail-Safes To Protect Against Overload

Beyond query and connection management, Vitess implements broader fail-safes:

- Idle or long transactions are terminated, reducing memory leaks and deadlocks.
- Rate limits can be enforced to prevent specific users or services from flooding the database.
- Detailed metrics and dashboards help Site Reliability Engineers (SREs) quickly spot and fix performance regressions.

Conclusion

As YouTube scaled to serve billions of users, its backend faced significant challenges in database performance, reliability, and manageability.

Initially relying on MySQL, the platform encountered limitations with connection capacity, replication lag, and query safety.

To address this, YouTube developed Vitess: a powerful, open-source database clustering system that extends MySQL's scalability while adding critical features. Vitess introduced connection pooling to prevent overload, query safety mechanisms to guard against inefficient operations, and a smart query router to handle sharded data across multiple servers.

Features like result reuse and a custom row cache enhanced efficiency, while real-time cache invalidation ensured consistency. Automation of tasks such as reparenting and backup further reduced operational complexity.

Together, these innovations enabled YouTube to maintain high availability, rapid performance, and data integrity at a massive scale.

References:

- Scaling YouTube's Backend: The Vitess Trade-offs - @Scale 2014
- Vitess VTTablet