Starficient Technical Brief

MySQL 5.7 to 8.0 Upgrades

K.Hunt

STARFICIENT

2024-04-04

Technical Brief: MySQL 5.7 to 8.0 Upgrades

Introduction

MySQL 8.0, a new version of the popular database management system, has made a big change by removing the query cache feature. This change affects how we can make databases run quickly and smoothly. It's especially important for people who are upgrading from MySQL 5.7 to 8.0, like developers and database administrators.

In this document, we'll talk about why the query cache was removed and how it might impact database performance. We'll also suggest some strategies and tools to help deal with these challenges and make databases work better with the new version.

The main goal is to help readers understand the changes and give them practical solutions so they can confidently start using MySQL 8.0. We'll explain everything in a way that's easy to understand, so people can make informed decisions about their databases.

Background and Context

MySQL is a popular database management system that many websites and applications use to store and retrieve data. As technology has changed over time, MySQL has also evolved to keep up with the needs of modern applications.

Before MySQL 8.0, there was a feature called the query cache. It was designed to improve the speed of reading data from the database. When a SELECT query was run, the query cache would store the results. If the same query was run again later, the cached results could be quickly retrieved instead of running the query all over again. This made data retrieval faster, especially for frequently used queries.

However, as web applications and the environments in which databases operate have changed, the query cache has become less effective. In situations where many users are accessing the database at the same time (high concurrency), the query cache can actually slow things down. The work required to keep the cache up-to-date and valid often cancels out the benefits it provides. Additionally, modern applications often need to handle data in more dynamic ways, which the query cache wasn't designed for.

As a result, MySQL 8.0 has removed the query cache feature. This change reflects a shift towards finding other ways to optimize database performance that are better suited to today's applications and environments. It's part of a larger trend in technology to find solutions that are more scalable, efficient, and adaptable.

For developers and database administrators, this change means they need to find new ways to ensure their databases are running quickly and efficiently. Understanding why the query cache was removed and exploring alternative strategies is important for successfully adapting to this new landscape of database performance optimization.

Issue Description

The main problem talked about in this technical brief is that MySQL 8.0 no longer has a feature called the query cache[1]. The query cache was a tool that made databases faster at reading data by storing the results of SELECT queries. When the same query was run again, the stored results could be quickly pulled from the cache. This reduced the workload on the database and made things run faster. Now that it's gone, it could cause performance problems, especially for applications that relied heavily on this feature for speed and efficiency.

This issue mainly affects developers, database administrators, and businesses who have built their systems using MySQL and are planning to upgrade from version 5.7 to 8.0. Without the query cache, they might see a drop in performance, particularly when there are a lot of read operations happening at once. This can impact not only how fast things run but also how well the applications can handle increased usage and traffic.

Additionally, this problem also affects the operational and financial sides of things, especially for services hosted on platforms like Amazon RDS[2]. Users of MySQL 5.7 may have to pay more in extended support fees as they figure out when and how to upgrade, considering the removal of the query cache.

Overall, this change means that people will need to rethink their strategies for making databases run efficiently to work well with the new version of MySQL without sacrificing performance or operational effectiveness.

[1]The MySQL 5.7 Reference Manual - 8.10.3 The Query Cache.
[2]Automating SQL Caching for Amazon ElastiCache and Amazon RDS

Impact Analysis

The decision to remove the query cache in MySQL 8.0 has some big effects on how well software runs, how users experience it, and how businesses operate overall. Right away, we might see increased delays and possibly more strain on the database server. This is because the query cache used to help optimize things, but now it's gone. Applications that do a lot of reading from the database and relied on the query cache for quick data retrieval may now feel slower and less responsive.

For businesses, this change could lead to slower applications and websites. Users might get frustrated if things take longer to load or respond. This could even cause some users to stop using these applications altogether. Companies may need to invest in more powerful hardware for their databases or find other ways to speed things up to keep their users happy.

Behind the scenes, IT teams and database administrators will be busy. They'll need to rework their database setups and find new ways to optimize performance without the query cache. This might involve trying out different database designs, using new tools, or even switching to a different database management system entirely.

The costs of running databases may also go up. Without the performance boost from the query cache, companies might need to pay for more computing power and storage. They may also need to train their staff on new optimization techniques, which takes time and money.

In short, removing the query cache in MySQL 8.0 has ripple effects that touch on software performance, user satisfaction, business operations, and IT budgets. Companies and their tech teams will need to adapt and find new solutions to keep things running smoothly in this post-query-cache world.

Immediate Impacts:

- **Performance Degradation**: Without the query cache, every SELECT query hits the database, which can increase response times and load, particularly for applications not optimized for such changes.
- **Increased Server Load**: The removal necessitates additional computation for each request, potentially leading to increased CPU and memory usage, which can strain resources in high-concurrency environments.

Long-Term Impacts If Unresolved:

- **Scalability Issues**: As businesses grow and data demands increase, the absence of a built-in caching mechanism may exacerbate challenges in maintaining performance and managing load efficiently.

- **Operational Costs**: Higher resource consumption might necessitate scaling up infrastructure sooner than anticipated, leading to increased operational costs. For users on managed services like Amazon RDS, lingering on MySQL 5.7 due to concerns over the removal of the query cache could also accrue additional extended support fees.
- **User Experience Deterioration**: The cumulative effect of performance challenges can significantly affect the end-user experience, leading to dissatisfaction and potentially impacting the business reputation and customer retention.

Potential Business Process Impacts:

- **Maintenance and Development Overheads**: The need to implement alternative caching mechanisms or optimize applications for the new database environment may require considerable development effort and resources.
- **Adaptation to Technological Evolution**: The shift reflects broader trends in database and application design towards more dynamic data handling. Failing to adapt could risk falling behind technologically, affecting competitiveness.

In the end, getting rid of the query cache in MySQL 8.0 causes some technical problems right away, but over time, it could have a big impact on how happy users are, how well businesses operate, and how competitive they are.

If companies don’t deal with these issues quickly, they could see some serious negative consequences. Users might get frustrated and leave if the software they use feels slower or less reliable. This could lead to a bad reputation and loss of business.

Operations could also take a hit. If databases aren’t running as efficiently, it could slow down important processes and make it harder for employees to do their jobs. This could cost businesses time and money.

In a competitive market, companies that can’t adapt to this change and keep their software running smoothly might fall behind. Customers may choose to go with faster, more reliable options from other businesses.

That’s why it’s so important for companies to jump on this issue right away. They need to find new ways to optimize their databases and adapt to a world without the query cache. This might take some work upfront, but it’s essential for keeping users happy, operations running smoothly, and staying competitive in the long run.

The smartest companies will see this as an opportunity. By getting creative and finding even better ways to handle database performance, they could come out ahead. But they need to act fast and be ready to make some changes.

So while losing the query cache might seem like a small technical detail, it could have a big impact on businesses if they don't handle it well. Quick action and smart optimization strategies are the keys to turning this challenge into an opportunity.

Proposed Solution(s)

To effectively navigate the challenges presented by the removal of the query cache in MySQL 8.0, a combination of optimization strategies, adoption of external caching mechanisms, and careful planning is proposed. The following solutions are designed to mitigate the immediate and long-term impacts on performance, scalability, and operational efficiency.

1. Optimization of Database Queries and Indexes

- **Technical Approach**: Analyze current database queries and schemas to identify inefficiencies. Optimize queries by reducing complexity, eliminating unnecessary operations, and ensuring proper use of indexes.
- **Tools and Technologies**: Utilize MySQL Workbench for query optimization, and consider tools like Percona's pt-query-digest[3] for analyzing query performance.

2. Implementation of External Caching Solutions

- **Technical Approach**: Adopt external caching systems such as Redis or Memcached to offload read operations from the database. This approach seeks to emulate the benefits of the deprecated query cache by storing frequently accessed data in memory.
- **Tools and Technologies**: Implement Redis or Memcached as an in-memory data store. Use client libraries compatible with your application's development framework to interact with the cache.

3. Server and Database Configuration Tuning

- **Technical Approach**: Adjust MySQL server configurations to optimize performance for the specific characteristics of your workload. Focus on parameters that affect the InnoDB storage engine, memory allocation, and query execution plans.
- **Tools and Technologies**: Leverage MySQL's built-in configuration recommendations and third-party tools like MySQLTuner[4] or Percona Toolkit to identify optimal settings.

[3] pt-query-digest
[4] MySQLTuner

4. Use of Database Proxy for Connection Pooling and Query Routing

- **Technical Approach**: Deploy a database proxy such as ProxySQL to manage connection pooling, reduce latency, and intelligently route queries. This can also help in implementing read/write splits to distribute load more effectively.
- **Tools and Technologies**: ProxySQL[5], or even HAProxycan be used for connection pooling, query routing, and optionally automatic query caching.

5. Continuous Performance Monitoring

- **Technical Approach**: Establish a comprehensive monitoring framework to continuously assess database performance and identify areas for improvement. This proactive approach allows for timely adjustments to configuration, schema, or indexing strategies.
- **Tools and Technologies**: Use monitoring solutions like Prometheus paired with Grafana for visualization, along with MySQL's Performance Schema[6] for in-depth database metrics.

By implementing these proposed solutions, organizations can address the challenges brought about by the removal of the query cache in MySQL 8.0, ensuring their applications remain performant, scalable, and cost-effective. It is critical to adopt a holistic approach, combining technical optimizations with strategic planning and continuous monitoring, to adapt successfully to these changes.

[5] ProxySQL
[6] MySQL's Performance Schema

Alternatives Considered

In addressing the removal of the query cache feature in MySQL 8.0, several alternative solutions were explored but ultimately not chosen as primary recommendations. Below are the alternatives considered, along with the reasons for their exclusion from the final proposed solutions.

1. Sticking with MySQL 5.7

- **Consideration**: One straightforward approach could have been to delay the upgrade to MySQL 8.0 and stick with MySQL 5.7 to continue leveraging the query cache feature.
- **Reasoning Against**: While delaying the upgrade avoids immediate challenges, it postpones inevitable migration problems and does not take advantage of the improved features and security enhancements in MySQL 8.0. Additionally, sticking with older versions may eventually lead to compatibility and support issues. For customers on Amazon AWS RDS and Aurora, there are additional charges after March 1st 2024 to continue using MySQL 5.7.[7]

2. Database Splitting/Sharding

- **Consideration**: Dividing the database into smaller distinct parts (sharding) or splitting database functions (such as separating read and write operations) across different database instances could potentially distribute load more effectively.
- **Reasoning Against**: Sharding and splitting are complex strategies that can significantly increase operational complexity and overhead. They require substantial changes to the application architecture and database management practices, making them less favorable for immediate performance optimization needs.

3. Over-provisioning Database Resources

- **Consideration**: Scaling up the database resources (CPU, RAM) might serve as a direct approach to compensate for the increased load due to the absence of query caching.
- **Reasoning Against**: While this can offer a temporary buffer against performance degradation, it is a costly and inefficient solution in the long term. Over-provisioning does not address the root causes of performance issues and can lead to wasteful resource utilization.

[7]Amazon RDS Extended Support for MySQL databases on Amazon Aurora and Amazon RDS

4. Application-level Caching

- **Consideration**: Implement caching mechanisms directly within the application's codebase, rather than relying on external or database-level caching.
- **Reasoning Against**: While application-level caching offers fine-grained control over what gets cached and when, it introduces additional complexity into the application code. It also places the onus of cache invalidation and management on the application developers, potentially leading to maintenance challenges and inefficiencies. Consider a No-Code SQL Query cache such as ProxySQL[8] or Heimdall Data Proxy[9].

Each of these alternatives presents possible avenues to address the impacts of the query cache removal, but they were deemed less favorable due to the trade-offs in complexity, cost, and scalability they introduce. The selected solutions aim to balance efficiency, manageability, and future-proofing as organizations transition to MySQL 8.0.

[8] ProxySQL
[9] Heimdall Data

Implementation Plan

Implementing the chosen solution to address the removal of the query cache in MySQL 8.0 involves a series of strategic steps. This plan outlines the process, timelines, resource requirements, and dependencies necessary for a smooth transition.

Step 1: Performance Assessment and Baseline Establishment

- **Activity**: Conduct a comprehensive performance assessment of the current MySQL 5.7 setup. Identify queries and operations heavily reliant on the query cache.
- **Timeline**: 1-2 weeks
- **Resources**: Database analysts, access to current MySQL setups
- **Dependencies**: Performance monitoring tools

Step 2: External Caching Solution Selection and Setup

- **Activity**: Choose an external caching solution (Redis or Memcached). Install and configure the chosen cache system.
- **Timeline**: 2-3 weeks
- **Resources**: Infrastructure team for setup, budget for additional hardware/software if applicable
- **Dependencies**: Decision on caching solution, infrastructure availability

Step 3: Database Queries and Indexes Optimization

- **Activity**: Review and optimize existing database queries and indexes. Run tests to ensure optimizations do not negatively impact application functionality.
- **Timeline**: 3-4 weeks
- **Resources**: Database developers, comprehensive test suite for application functionalities
- **Dependencies**: Performance assessment data, access to development and testing environments

Step 4: MySQL Configuration Tuning

- **Activity**: Adjust MySQL server settings for optimal performance in the absence of the query cache. Focus on parameters affecting the InnoDB engine and memory allocation.
- **Timeline**: 1 week
- **Resources**: Database administrators, documentation on MySQL 8.0 settings
- **Dependencies**: Completion of steps 2 and 3, access to MySQL configuration files

Step 5: Implementation of Database Proxy (Optional)

- **Activity**: Deploy a Database Proxy for connection pooling and smart query routing. Configure caching and load balancing rules.
- **Timeline**: 2 weeks
- **Resources**: Systems administrators, budget for proxy servers
- **Dependencies**: External caching setup, optimized database queries

Step 6: Continuous Monitoring and Performance Tuning

- **Activity**: Establish a monitoring framework using tools like Prometheus and Grafana. Continuously assess database performance and adjust configurations as necessary.
- **Timeline**: Ongoing
- **Resources**: Monitoring tools, dedicated personnel for performance analysis
- **Dependencies**: Implementation of previous steps, access to performance data

Step 7: Documentation and Team Training

- **Activity**: Document the changes and configurations made. Train the development, database, and operations teams on the new setup and best practices.
- **Timeline**: 2 weeks
- **Resources**: Technical writers, training materials, personnel for conducting training sessions
- **Dependencies**: Completion of previous steps, availability of personnel for training

Step 8: Final Testing and Go-live

- **Activity**: Conduct thorough testing of the MySQL 8.0 environment with the new optimizations and caching solutions in place. Once confirmed stable, proceed with full migration.
- **Timeline**: 2 weeks
- **Resources**: Testing team, rollback plans in case of issues
- **Dependencies**: Successful completion and stability of previous steps

This phased approach ensures a structured and manageable transition to MySQL 8.0 without the query cache, focusing on performance, scalability, and operational continuity.

Risks and Mitigations

The transition to MySQL 8.0 and adaptation to the absence of the query cache entail several potential risks. Identifying these risks and implementing targeted mitigation strategies is crucial for a smooth implementation. Below are identified risks and their respective mitigations:

Risk 1: Performance Degradation Post-Transition

- **Mitigation**: Conduct extensive performance testing in a staged environment that mirrors the production setup as closely as possible before making the transition. Use the results to fine-tune configurations and optimizations further.

Risk 2: Increased Complexity with External Caching Solutions

- **Mitigation**: Select caching solutions with strong community support and comprehensive documentation. Provide training for team members on managing and operating the chosen caching tools effectively.

Risk 3: Inadequate Resource Allocation

- **Mitigation**: Perform a resource usage assessment as part of the performance testing phase. Based on the findings, ensure that the infrastructure is sufficiently scaled to handle anticipated loads without the query cache.

Risk 4: Operational Disruptions During Migration

- **Mitigation**: Develop a detailed migration plan that includes rollback procedures. Conduct the migration during off-peak hours to minimize the impact on operations. Ensure thorough testing in the staging environment to reduce the likelihood of unforeseen issues.

Risk 5: Increased Operational Costs

- **Mitigation**: Monitor and analyze costs associated with the new setup continuously. Implement cost-optimization strategies, such as rightsizing instances based on actual usage patterns and exploring reserved instances for long-term savings.

Risk 6: Skill Gaps Among Team Members

- **Mitigation**: Identify training needs early in the planning phase. Organize workshops and training sessions focusing on MySQL 8.0 features, external caching mechanisms, and performance optimization techniques.

Risk 7: Compliance and Security Concerns

- **Mitigation**: Review the security features and compliance certifications of the chosen external caching solutions. Ensure that data handling and storage practices align with regulatory requirements and industry best practices.

Adopting these mitigations can help manage the risks effectively, ensuring that the transition to MySQL 8.0 and the adoption of alternative performance optimization strategies proceed as smoothly as possible.

Next Steps

To implement the proposed solution, the following steps should be taken:

1. Review the technical brief and ensure understanding of the issue and proposed solution.
2. Identify resources and personnel required to implement the solution.
3. Schedule a meeting to discuss the implementation plan and obtain buy-in from relevant stakeholders.
4. Begin implementation of the solution, following the step-by-step process outlined in the technical brief.
5. Monitor progress and adjust as needed to ensure successful implementation.

We look forward to working with you to resolve this issue and improve the software's performance.

Conclusion

The transition from MySQL 5.7 to 8.0 marks a significant evolutionary step in the MySQL database's development, introducing a range of enhancements designed to improve performance, security, and usability. One of the most notable changes— the removal of the query cache—presents both a challenge and an opportunity for organizations relying on MySQL for their database needs. This brief has outlined the rationale behind the removal of the query cache, its potential impacts on database performance, and detailed a comprehensive strategy for adapting to this change.

Key points discussed include: - The removal of the query cache is driven by its inefficiencies in high-concurrency scenarios and the need for MySQL to support more scalable, dynamic data management strategies. - Proposed solutions to mitigate the impact of this change focus on optimizing database queries and indexes, adopting external caching solutions like Redis or Memcached, fine-tuning MySQL configuration, and implementing a robust monitoring framework to ensure continued performance optimization. - Implementing these solutions requires careful planning, resource allocation, and ongoing performance monitoring but promises significant benefits in terms of improved database performance, scalability, and operational efficiency.

Addressing the challenge presented by the removal of the query cache in MySQL 8.0 is crucial for maintaining the high performance and reliability that users have come to expect from their database systems. By adopting the recommended strategies, organizations can turn this challenge into an opportunity to modernize their data management practices, enhance database performance, and ensure their infrastructure remains robust and scalable for the future.

This technical brief provides a comprehensive overview of the issue and a detailed proposal for resolving it. We encourage readers to review the document and consider the proposed solution. If you have any questions or concerns, please reach out to Starficient; https://www.starficient.com/contact/ for any questions you may have.

Thank you for your attention to this matter.

References

The following references provide additional information and context for the topics discussed in this technical brief:

1. MySQL Documentation - MySQL 8.0 Reference Manual. Available at: https://dev.mysql.com/doc/refman/8.0/en/
2. Redis - An open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker. Official website: https://redis.io/
3. Memcached - A high-performance, distributed memory object caching system. Official website: http://memcached.org/
4. Percona Toolkit - A collection of advanced open source command-line tools that are used by Percona support staff to perform a variety of MySQL and MongoDB server and system tasks. Available at: https://www.percona.com/software/mysql-tools/percona-toolkit
5. Prometheus - An open-source systems monitoring and alerting toolkit originally built at SoundCloud. Official website: https://prometheus.io/
6. ProxSQL - A high-performance MySQL proxy for MySQL, MariaDB, Amazon Aurora / RDS and other forks. https://proxysql.com/
7. Grafana - The open platform for beautiful analytics and monitoring. Official website: https://grafana.com/
8. MySQL Workbench - A unified visual tool for database architects, developers, and DBAs. Available at: https://www.mysql.com/products/workbench/
9. Heimdall Data - SQL database proxy that dynamically caches, balances, and routes for better SQL query performance. Available at: https://www.heimdalldata.com/
10. MySQL Performance Schema - A feature for monitoring MySQL Server execution at a low level. Available in the MySQL documentation: https://dev.mysql.com/doc/refman/8.0/en/performance-schema.html
11. MySQLTuner - A script written in Perl that allows you to review a MySQL installation quickly and make adjustments to increase performance and stability. Available at: http://mysqltuner.com/
12. Amazon RDS - Amazon Relational Database Service. Available at: https://aws.amazon.com/rds/
13. Automating SQL Caching for Amazon ElastiCache and Amazon RDS. Amazon Web Services, Inc.: https://aws.amazon.com/blogs/database/automating-sql-caching-for-amazon-elasticache-and-amazon-rds/.
14. The MySQL 5.7 Reference Manual - 8.10.3 The Query Cache." Oracle: https://dev.mysql.com/doc/refman/5.7/en/query-cache.html

Glossary

- ACID Compliance; Stands for Atomicity, Consistency, Isolation, Durability. A set of properties that guarantee database transactions are processed reliably.
- Amazon ElastiCache; A web service by Amazon Web Services that makes it easy to set up, manage, and scale a distributed cache environment in the cloud.
- Amazon RDS; Amazon Relational Database Service, a distributed relational database service by Amazon Web Services (AWS).
- Buffer Pool; A memory area within the database engine where data is cached to speed up data retrieval operations by reducing the need to access the disk.
- Cache Invalidation; The process of removing entries from a cache because they are no longer considered to be valid, ensuring that stale or incorrect data is not served to users.
- Concurrency; The capability of a database management system to manage multiple transactions being executed simultaneously, not necessarily simultaneously but within the same timeframe, without leading to conflicts or data integrity issues.
- Configuration Wizard; Tool(s) that guide users through setting up and configuring software, often based on the specific needs or expected workload of the system.
- Data Routing; In the context of databases, it refers to the decisions made regarding which database server or node should handle specific queries or transactions, often used in distributed databases or replication setups.
- Database Proxy; A server that acts as an intermediary for requests from clients seeking resources from databases. It can provide additional features like load balancing, caching, or SQL query manipulation.
- Extended Support Fees; Costs incurred for continuing to receive support for a software product after its standard support period has ended.
- Heimdall Data; A database proxy that provides query caching, load balancing, and database failover solutions to improve SQL database performance.
- High-Concurrency; Referring to environments where a database or application handles many transactions or operations simultaneously, often leading to increased complexity in data management.
- Inefficiency; The reduced effectiveness or productivity within a system, often marked by increased resource consumption or slower performance.
- Load Balancing; The process of distributing workloads across multiple computing resources, such as databases or servers, to optimize resource use, maximize throughput, reduce response time, and ensure redundancy.
- Memcached; A high-performance, distributed memory object caching system designed to speed up dynamic web applications by alleviating database load.
- MySQL Workbench; An integrated development environment (IDE) for MySQL, which allows devel-

opers and database administrators to visually design, model, generate, and manage databases.

- MySQL; An open-source relational database management system based on SQL (Structured Query Language).
- Percona Toolkit; A collection of advanced command-line tools to perform a variety of MySQL, MongoDB, and PostgreSQL server and system tasks for optimization, maintenance, and scaling.
- Query Cache; A feature in MySQL (up to version 5.7) that stores the result set of SELECT queries for faster retrieval on subsequent identical queries.
- Query Execution Plan; A set of steps used by the database to retrieve data. It outlines the method chosen by the database's query planner to execute SQL queries, including how tables are joined and in which order.
- Query Optimization; The process of modifying a SQL query to improve its efficiency, reducing the resources it requires or the time it takes to run.
- Redis; An open-source, in-memory data structure store, used as a database, cache, and message broker.
- Scalability; The ability to handle a growing amount of work, or the potential to be enlarged to accommodate that growth.
- Server Optimization; Adjusting the configurations and settings of a server to improve its performance and efficiency.
- Thread Cache; A cache implemented by MySQL to manage threads. When a client disconnects, its thread is placed in the cache if there is room, thereby avoiding thread cleanup and creation overhead for future connections.
- sys schema; A set of views, functions, and procedures in MySQL that helps database administrators to get insight into database performance, health, and internals.