UDC 004.65

Y. Kornaga, Yu. Bazaka, E. Marienko

WAYS TO OPTIMIZE SQL QUERIES TO IMPROVE DATABASE PERFORMANCE IN HIGH-LOAD SYSTEMS

Abstract: SQL statements are used to retrieve information from a database. In most cases, these queries are executed very slowly, the reason for this is the low quality of their writing. For better performance, we need to use faster and more efficient queries. This article shows you how to optimize SQL queries for better performance. The topic of query optimization is very broad, but we will try to cover the most important aspects of this issue. In this paper, I do not focus on in-depth database analysis, but focus on simple hints and tips for setting up queries that can be used to immediately increase productivity.

Keywords: SQL, optimization, database, query, performance, analysis, data processing.

Problem Statement

SQL is an information-logical language designed to describe, modify and extract data stored in relational databases. Over time, the SQL language has become more complex: enriched with new constructs, provided the ability to describe and manage new saved objects, which is why in this article I try to present possible options for optimizing queries. Query customization requires knowledge of techniques such as cost-based and heuristic optimizers, as well as the tools provided by the SQL platform to explain the query execution plan.

The best way to adjust performance is to try to write queries in different ways and compare their ease of reading and execution. The methods presented in this work have been personally tested by me, the justification for their use has been experimentally confirmed.

Proposed solution and comparison with existing solutions

1. Use the column name instead of * in the SELECT statement

If you only need to select a few columns from the table, you do not need to use SELECT *. Although it is easier to write, such a query takes much more time to execute in the database. Selecting only the columns you want can reduce the size of the resulting table, so you can reduce network traffic and, in turn, increase the overall query efficiency[1].

Example:

Initial query: SELECT * FROM Users;

Optimized query: SELECT name, age, surname, location, gender, identificatory FROM Users;

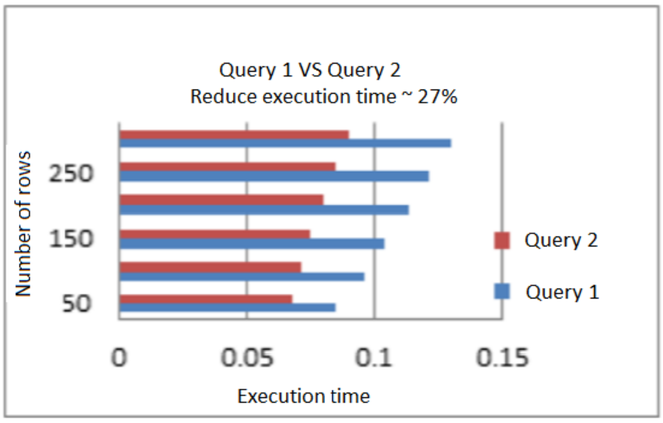

Figure 1. Comparison of query execution speed before and after optimization

2. Avoid using the HAVING keyword in the SELECT statement

The HAVING expression is used to filter rows after grouping data from a table using the GROUP BY expression. Its use is redundant in the SELECT statement. It works as follows: based on the table obtained with SELECT, it compares the filtering condition and the row, and based on the result, it is decided to add a row to the resulting sample or not[2].

Example:

Initial query: SELECT age, count (age), avg (age) FROM Users GROUP BY age HAVING id! = 10 AND id! = 6;

Optimized query: SELECT age, count (age), avg (age) FROM Users WHERE id! = 10 AND id! = 6 GROUP BY age;

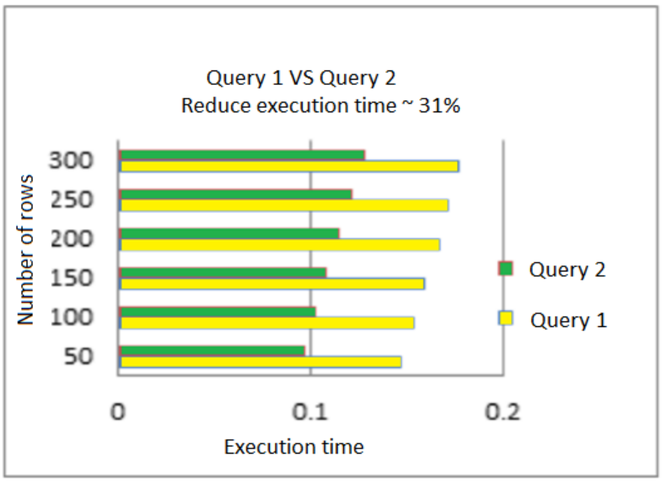

Figure 2. Comparison of query execution speed before and after optimization

3. Get rid of unnecessary DISTINCT operators

For example, in the following query, the keyword DISTINCT is redundant because table_name contains the primary key p.ID, which is part of the result set. Using both DISTINCT and the primary key is a kind of tautology[3].

Example:

Initial query: SELECT DISTINCT * FROM Users JOIN Product ON Users.id = Product.id WHERE Users.id = 1;

Optimized query: SELECT * FROM Users JOIN Product ON Users.id = Product.id WHERE Users.id = 1;

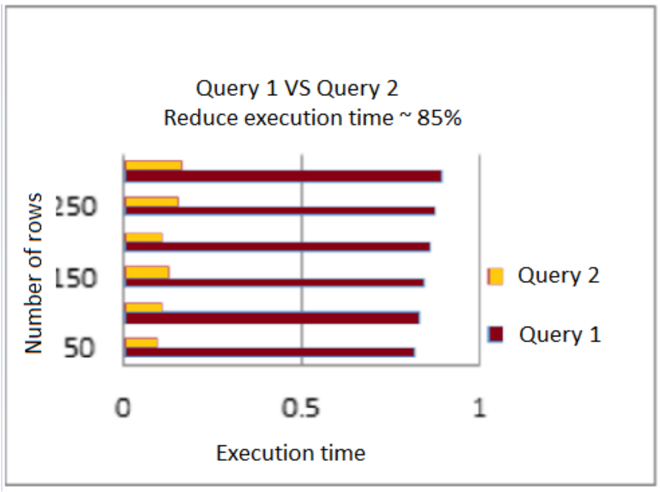

Figure 3. Comparison of query execution speed before and after optimization

4. Avoid subqueries

Using subqueries is very resource intensive, the existing alternative in the form of JOIN is much faster and requires fewer resources. Subqueries are mainly used in ALL, ANY, EXISTS constructions. Unrelated subqueries, or queries with more than one table in FROM, can be executed for each row in the resulting sample, which can lead to an excessive increase in the number of these queries[5].

Example:

Initial query: SELECT * FROM Product WHERE Product.prod_id = (SELECT Users.prod_id FROM Users WHERE Users.id = 356;

Optimized query: SELECT Product. * FROM Product, Users WHERE Product.prod_id = Users.prod_id AND Users.id = 356;

5. Using the IN operator

The IN predicate can be used for indexed search, which will significantly increase the speed of finding the desired data. The list inside IN must contain only constants and values that are constant[7].

Example:

Initial query: SELECT * FROM users WHERE users.id = 14 OR users.id = 17;

Optimized query: SELECT * FROM users WHERE users.id IN (14, 17);

6. Use EXISTS instead of DISTINCT

The DISTINCT keyword selects all columns in the table and then removes duplicates from them. And with EXISTS you can avoid having to return the entire table.

Example:

Initial query: SELECT DISTINCT c.country_id, c.country_name FROM SH.countries c, SH.customers e WHERE e.country_id = c.country_id;

Optimized query: SELECT c.country_id, c.country_name FROM SH.countries c WHERE EXISTS (SELECT 'X' FROM SH.customers e WHERE e.country_id = c.country_id);

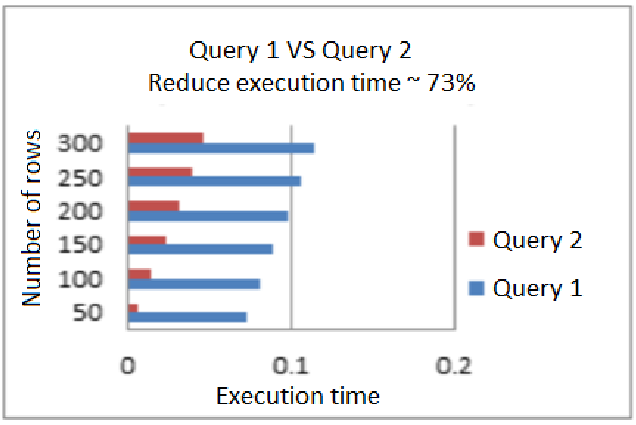

Figure 4. Comparison of query execution speed before and after optimization

7. Using UNION ALL instead of UNION

The UNION ALL expression works faster than UNION because the UNION ALL operator does not consider duplicates, unlike UNION which looks for duplicates when selecting strings, whether they exist or not[8-9].

Exemple:

Initial request: SELECT cust_id FROM SH.sales UNION SELECT cust_id FROM customers;

Optimized query: SELECT cust_id FROM SH.sales UNION ALL SELECT cust_id FROM customers;

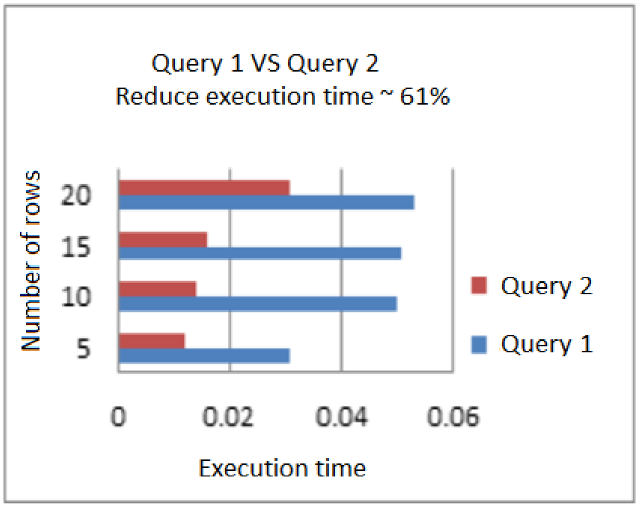

Figure 5. Comparison of query execution speed before and after optimization

Conclusion

Query optimization is a necessary skill that database administrators and application developers must have to increase overall system performance. The purpose of this work is to provide a list of possible SQL scripts, the use of which will significantly reduce the time required to develop and maintain databases. Even with a strong infrastructure, there is a risk that performance may be significantly impaired by inefficient requests. Queries have a very large impact on the speed of the database, and with their optimization you can achieve a significant increase in productivity.

Therefore, I recommend that you carefully follow the general tips for improving the SQL queries that were presented above. These tips are just one of the many ways to optimize your databases, but their use is a must for all developers.

REFERENCES

1. Schwartz B. High Performance MySQL: Optimization, Backups, and Replication / B. Schwartz. - Sebastopol: O'Reilly Media, 2012. - 826 p.

2. Dubois P. MySQL. Collection of recipes / P. Dubois. - Sebastopol: O'Reilly Media, 2015. - 1056 p.

3. Williams H. Learning MySQL: Get a Handle on Your Data / H. Williams. - Sebastopol: O'Reilly Media, 2006. - 618 p.

4. Forta B. MySQL Crash Course / B. Forta. - Indianapolis: Sams Publishing, 2005. - 336 p.

5. Pipes J. Pro MySQL (The Expert's Voice in Open Source) / J. Pipes. - New York: Apress, 2005. - 768 p.

6. Murphy K. MySQL Administrator's Bible / K. Murphy. - Hoboken: Wiley, 2009. - 888 p.

7. Supercharge Your SQL Queries for Production Databases // https://www.sisense.com/. Availabte at: https://www.sisense.com/blog/8-ways-fine-tune-sql-queries-production-databases/.

8. Query optimization techniques in SQL Server: tips and tricks // https://www.sqlshack.com/. Availabte at: https://www.sqlshack.com/query-optimization-techniques-in-sql-server-tips-and-tricks/.

9. Top 10 SQL Query Optimization Tips to Improve Database Performance // https://www.mantralabsglobal.com/. Availabte at: https://www.mantrala bsglobal.com/blog/sql-query-optimization-tips/.

10. SQL Database Performance Tuning for Developers // https://www.toptal.com/. Availabte at: https://www.toptal.com/sql-server/sql-database-tuning-for-developers/.