Get unlimited access to the best of Medium for less than $1/week. Become a member

Optimizing SQL Query Performance: A Comprehensive Guide

Discover the secrets to SQL query optimization and supercharge your database's performance.

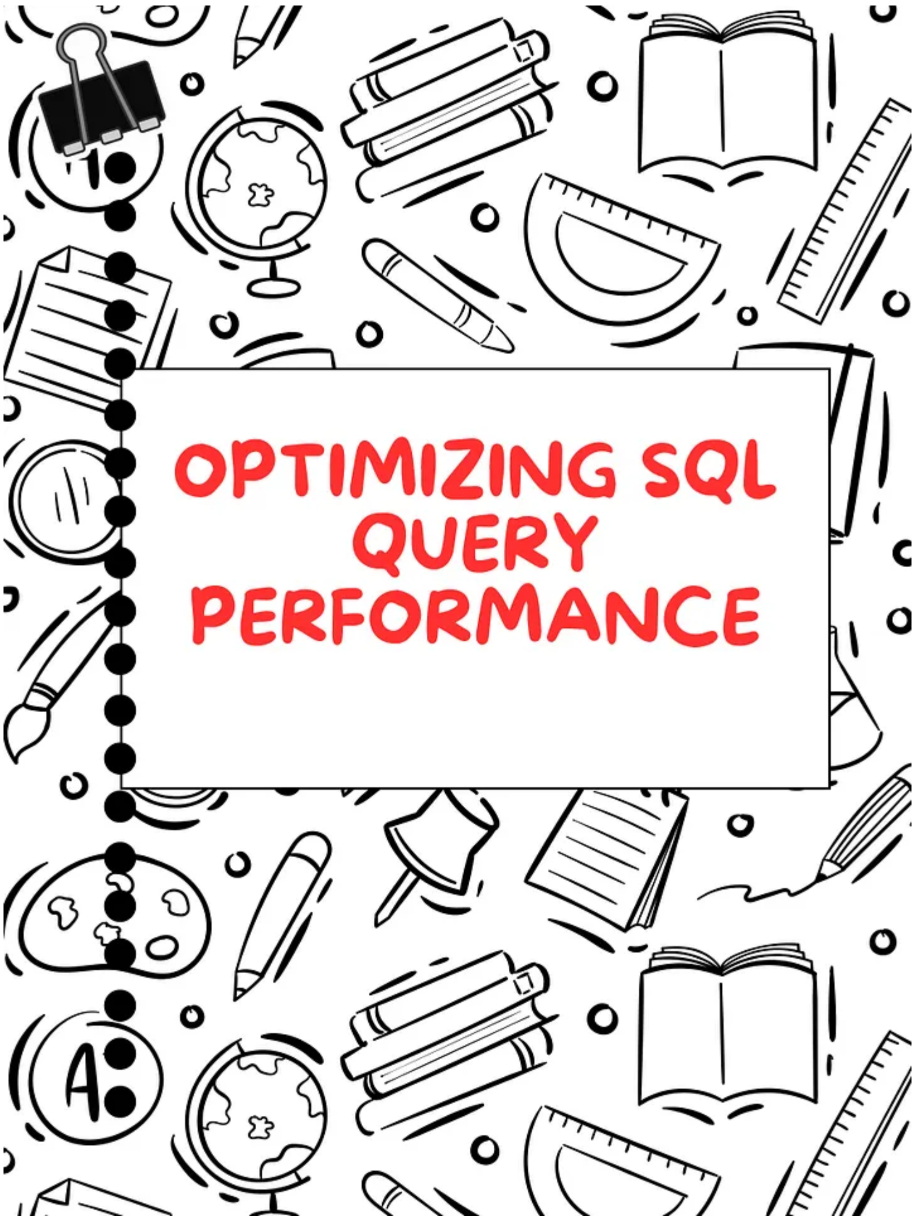
Taran Kaur Follow 10 min read · Mar 7, 2024

264 3

Key Takeaways:

- Analyze Execution Plans
- Use Indexes Effectively
- Optimize Data Retrieval
- Refine JOIN Operations
- Use LIMIT and OFFSET
- Avoid SELECT DISTINCT
- Limit Use of Subqueries
- Choose EXISTS Over IN
- Implement Efficient Pagination
- Minimize OR Conditions
- Avoid Unnecessary Calculations
- Regular Review and Refinement
- Understand Data and Requirements
- Optimize Use of Temporary Tables

. . .



SQL query optimization

In the world of managing databases, making SQL queries run efficiently is incredibly important. By optimizing these queries, we can cut down on how long they take to execute, make apps run smoother, and in the end, leave users happier. This article explores different ways and tips to make SQL queries perform better.

💡 **Enjoyed this piece? Stay connected with FactMedics for more data-driven health insights.**
👉 **Subscribe to our LinkedIn Newsletter**
📊 **Looking for clinician-led analytics or scientific writing support? Explore our services**

. . .

Indexing

One key technique for speeding up queries is indexing. Essentially, indexing helps speed up the process of finding and accessing data in a database. It's like a quick reference guide that allows for faster searches. But it's **important to use indexing wisely because while it can make data retrieval faster, it also comes with extra work, especially when data changes often.**

Using indexing smartly can significantly boost the speed of queries, as it helps the database system to find and fetch data much quicker.

Example:

```
CREATE INDEX idx_customer_name ON customers(name);
```

This creates an index on the `name` column of the `customers` table, speeding up queries searching by customer name.

. . .

Use EXISTS Instead of IN

When checking if a value is part of a smaller group, using EXISTS is usually faster than IN. This is because EXISTS lets the database stop looking the moment it finds a matching record, while IN may require going through the whole dataset to gather the smaller group for comparison.

In simpler terms, EXISTS speeds things up by ending the search as soon as it finds what it's looking for.

Example:

```
SELECT * FROM orders o WHERE EXISTS (SELECT 1 FROM customers c WHERE c.id = o.cu
```

This query retrieves orders only for active customers, using `EXISTS` for efficient existence checking.

. . .

Use Appropriate Datatype

The selection of data types greatly affects the efficiency of SQL queries. Opting for the **most appropriate data type for each column helps to optimize data storage, speed up comparisons, and eliminate needless data conversions.**

By picking the correct data type, you can reduce storage needs and enhance the performance of queries.

Example:

For a column storing dates, use the `DATE` datatype instead of `VARCHAR`.

```
ALTER TABLE events MODIFY column event_date DATE;
```

. . .

Use LIMIT and OFFSET

When an application needs to organize data into pages, using the LIMIT and OFFSET commands can improve its performance. These commands help by only pulling a specific chunk of records at a time, which is especially useful for speeding up the loading of big datasets.

In short, LIMIT and OFFSET help with data pagination by only loading a select portion of records, making things run smoother

Example:

```
SELECT * FROM products ORDER BY name LIMIT 10 OFFSET 20;
```

This retrieves 10 products, skipping the first 20, useful for displaying the third page of a paginated list.

. . .

Avoid SELECT DISTINCT

Applying SELECT DISTINCT in your queries can slow things down significantly, especially when working with large amounts of data. This is because the database has to sift through the data to remove any duplicates. A **smarter approach is to modify your queries or use indexing to ensure data is unique right from the get-go, avoiding reliance on SELECT DISTINCT.**

In essence, SELECT DISTINCT can be quite demanding on resources. It's smarter to craft your queries in such a manner that they naturally prevent duplicates without needing to resort to SELECT DISTINCT.

Example:

Instead of using `SELECT DISTINCT`, ensure your joins do not produce duplicate rows.

```
SELECT name FROM customers JOIN orders ON customers.id = orders.customer_id WHER
```

. . .

Avoid Subqueries

Subqueries provide a versatile method for organizing your queries, but they might slow things down. **Whenever possible, it's better to use joins or temporary tables instead, as this can make your queries run faster.**

Switching from subqueries to joins or temporary tables is a good strategy to boost performance.

Example:

Instead of a subquery:

```
SELECT * FROM employees WHERE department_id IN (SELECT id FROM departments WHERE
```

Use a join:

```
SELECT e.* FROM employees e JOIN departments d ON e.department_id = d.id WHERE d
```

. . .

Optimize Database Design

Having a well-organized database design is crucial for fast query performance. This means using normalization to get rid of unnecessary data repetition, wisely applying denormalization to avoid complicated joins, and implementing the right indexing strategies.

In short, a neatly arranged database is key to running efficient queries.

Example:

Normalization reduces redundancy, but strategic denormalization can reduce complex joins:

```sql
-- Adding frequently joined data as a column in a table
ALTER TABLE orders ADD COLUMN customer_name VARCHAR(255);
```

. . .

Prefer UNION ALL to UNION

You should opt for UNION ALL instead of UNION if you don't need to get rid of duplicate entries. **UNION ALL skips the step of sorting data to eliminate duplicates, making it a quicker option.**

Essentially, UNION ALL is speedier because it doesn't bother removing duplicates, unlike UNION.

Example:

```sql
SELECT name FROM employees WHERE status = 'active'
UNION ALL
SELECT name FROM contractors WHERE status = 'active';
```

. . .

Selecting Specific Fields

Choosing specific fields in the SELECT clause instead of using SELECT * reduces the data that needs to be handled and moved, which in turn boosts the performance of your query.

In simpler terms, only pick the fields you really need for your task.

Example:

```
SELECT id, name FROM customers;
```

Instead of `SELECT * FROM customers;`.

. . .

Use Stored Procedures

Stored procedures wrap up complex tasks and improve performance by being pre-compiled and reusing execution plans efficiently. They also boost security and cut down on the amount of data sent over the network.

In essence, stored procedures keep SQL operations on the server side, leading to better performance.

Example:

```
CREATE PROCEDURE GetActiveCustomers()
BEGIN
    SELECT * FROM customers WHERE active = TRUE;
END;
```

. . .

Avoid Cursors

Cursors, which handle data one row at a time, can use up a lot of resources because of their sequential processing approach. **It's better to use operations that deal with data in sets whenever you can**, as these are more efficient and can handle larger amounts of data more effectively.

In simple terms, working with data in batches is generally more efficient than processing it row by row, as cursors do.

Example:

Avoid:

```
DECLARE cursor_name CURSOR FOR SELECT * FROM table_name;
```

Prefer set-based operations wherever possible.

. . .

Minimize the Use of Wildcard Characters

Using wildcard characters, particularly at the start of a string (like in the case of LIKE '%term'), can slow down performance because it stops indexes from being used effectively. It's best to limit and think carefully about using them.

Example:

Avoid:

```
SELECT * FROM customers WHERE name LIKE '%Smith';
```

Prefer:

Top highlight

```
SELECT * FROM customers WHERE name LIKE 'Smith%';
```

. . .

Monitor Query Performance

Keeping an eye on how your queries perform regularly can help spot areas that need improvement and chances to make them run better. Using things like query execution plans and database profiling tools can really help with this process.

In simpler terms, **regularly check and tweak your queries using the right tools and techniques.**

Example:

```
EXPLAIN SELECT * FROM customers WHERE name = 'Smith';
```

`EXPLAIN` shows how MySQL executes a query, helping identify potential inefficiencies.

. . .

Use Appropriate Join Types

Picking the correct join type (INNER, LEFT, RIGHT, FULL) according to how tables are related and what data you need to get can greatly impact how well your query performs.

In essence, **choosing the appropriate join type is key to making your queries run more efficiently.**

Example:

```
SELECT orders.id, customers.name FROM orders
INNER JOIN customers ON orders.customer_id = customers.id;
```

Use `INNER JOIN` to retrieve orders with corresponding customer information.

. . .

Use Temp Table Instead of Subqueries

Using temporary tables can make complicated queries easier to handle and run faster. They work by dividing operations that involve many subqueries or intricate calculations into simpler, more manageable parts.

In short, **temporary tables help streamline complex queries.**

Example:

```
CREATE TEMPORARY TABLE IF NOT EXISTS temp_orders AS (SELECT * FROM orders WHERE
SELECT * FROM temp_orders JOIN customers ON temp_orders.customer_id = customers.
```

. . .

Avoid SELECT

The recommendation to "avoid SELECT" probably means you should skip selecting columns you don't need in the SELECT clause. It's important to **ask only for the columns required** for your particular task to cut down on how much data is processed and the time it takes to send this data over the network

. . .

Avoid Using Too Many Joins

Although joins are a strong feature, using them too much can make queries run slowly. Making the database layout more efficient or trying different methods like subselects or temporary tables could help reduce the reliance on too many joins.

. . .

Define Your Requirements

Having a clear understanding of what data you need to fetch can lead to creating more efficient queries. This means getting to know the data, how it's organized, and what you aim to achieve with the query.

In simple terms, **knowing exactly what data you're after makes it easier to write effective queries.**

Example:

Define the exact information needed before writing your query to avoid over-fetching data.

. . .

Missing Indexes

Finding and setting up indexes that are missing is a key part of making queries run faster. By looking at how queries are executed, you can figure out which tables could benefit from indexes to cut down on the amount of data they need to go through.

In simpler terms, **make sure to add indexes to columns** you use a lot to speed things up.

Example:

```
CREATE INDEX idx_order_status ON orders(status);
```

. . .

Optimize Your SQL Code

Writing efficient SQL code involves practices such as avoiding unnecessary

Medium
Search
Write
I
All your favorite parts of Medium are now in one sidebar for easy access.
Okay, got it
Profile
Stories
Stats
Following
Gaurav Goel
The Medium Blog
Reshma Bidikar
Find writers and publications to follow.
See suggestions

Review and refine your SQL queries regularly.

Example:

Optimize by combining operations, removing unnecessary conditions, and using efficient functions.

. . .

Simplify Joins

Simplifying join conditions and reducing the number of tables involved in a join can enhance query performance. This might involve restructuring the database or rethinking the approach to data retrieval.

Use the simplest possible joins that get the job done.

Example:

```
SELECT * FROM orders o JOIN customers c ON o.customer_id = c.id;
```

Use WHERE Instead of HAVING

WHERE clauses filter data before it's grouped together, and HAVING clauses do so afterward. This makes WHERE clauses more efficient for narrowing down data. Whenever you can, use WHERE to start filtering rows early in the process of running a query.

In simple terms, it's best to use WHERE to filter rows as soon as possible in your query.

Example:

```
SELECT * FROM orders WHERE order_date > '2020-01-01';
```

. . .

Avoid Negative Searches

Queries that include conditions such as NOT IN, NOT EXISTS, or start with wildcards (like using LIKE '%text') tend to run slower. This is because such conditions make it harder to use indexes effectively. By avoiding or limiting the use of these patterns, you can make your queries run faster.

In simpler terms, **using negative conditions in your queries can make them less efficient.**

Example:

Avoid:

```
SELECT * FROM customers WHERE NOT (age < 18);
```

Prefer positive conditions.

. . .

Avoiding Loops

In SQL, iterative operations (loops) can usually be replaced with set-based operations, which are more efficient and better optimized by the database engine.

Minimize or eliminate loops in SQL procedures.

Example:

Avoid procedural loops and opt for set-based operations.

. . .

Conclusion

Optimizing SQL queries is a critical skill for developers and DBAs alike. By applying the strategies outlined above, including appropriate indexing, avoiding costly operations, and leveraging the database's features, one can significantly enhance the performance of their database applications. Continuous learning and performance monitoring are key to maintaining and improving query efficiency.

. . .

FAQ

How can I speed up a slow-running SQL query?

Start by analyzing the query execution plan to identify bottlenecks. Use indexes effectively, avoid using SELECT *, limit the use of JOINs by only fetching necessary data, and consider using WHERE clauses to filter rows early.

Why is my query with JOIN operations slow, and how can I improve it?

Queries with JOINs can be slow if they're not using indexes efficiently or if they're joining large datasets. To improve performance, ensure that the columns used for joining are indexed. Also, consider whether you can limit the datasets being joined with WHERE clauses before the JOIN.

What is the impact of using SELECT * in my queries?

Using SELECT * can negatively impact performance because it retrieves all columns from the table, including those not needed for your specific operation. This increases the amount of data processed and transferred. Specify only the necessary columns in your SELECT clause.

How do indexes improve SQL query performance?

Indexes improve query performance by allowing the database engine to find data without scanning the entire table. They are especially beneficial for queries with WHERE clauses, JOIN operations, and ORDER BY statements. However, keep in mind that excessive indexing can slow down write operations.

Can using temporary tables improve query performance?

Yes, in some cases, using temporary tables to store intermediate results can simplify complex queries and improve performance. This is particularly useful for breaking down complex calculations or when working with multiple subqueries.

What is the difference between using IN and EXISTS, and which is faster?

EXISTS can be faster than IN in many cases because EXISTS stops processing as soon as it finds a match, whereas IN might scan the entire dataset. Use EXISTS for subquery conditions where you're checking for the existence of a record.

How can I optimize a query that uses a lot of subqueries?

Consider replacing some subqueries with JOINs or using temporary tables to hold intermediate results. This can reduce the complexity of the query and potentially improve performance.

What are some best practices for writing efficient SQL queries?

Some best practices include using indexes wisely, avoiding unnecessary columns in the SELECT clause, filtering data early with WHERE clauses, and preferring set-based operations over loops. Regularly review and refactor your queries for performance improvements.

How does pagination affect SQL query performance, and how can I

optimize it?

Pagination can affect performance by requiring the database to process large amounts of data for each page request. To optimize, use LIMIT and OFFSET clauses wisely, and consider caching page results for frequently accessed pages.

Why is my query with a lot of OR conditions slow, and how can I optimize it?

Queries with many OR conditions can be slow because they require the database to evaluate multiple conditions, which can be inefficient. Consider restructuring your query to minimize the use of OR or using UNION to combine the results of simpler queries.

. . .

Related Article:

Are SQL Queries Case Sensitive?
Exploring Case Sensitivity in SQL Queries.
code.likeagirl.io

Can SQL Store Images?
Exploring Image Storage in SQL.
code.likeagirl.io

What is the difference between SQL and Spark operators : A Review
A Guide to SQL and Spark Operators.
code.likeagirl.io

. . .

💡 Enjoyed this piece? Stay connected with FactMedics for more data-driven health insights.
👉 Subscribe to our LinkedIn Newsletter
📊 Looking for clinician-led analytics or scientific writing support? Explore our services

Sql Programming Coding Information Technology Data

Published in Women in Technology

3.5K followers · Last published 4 days ago

Follow

Women in Tech is a publication to highlight women in STEM, their accomplishments, career lessons, and stories. We feature the unique voices of our writers. Their opinions are their own and don't necessarily reflect our editorial stance.

Written by Taran Kaur

328 followers · 189 following

Follow

I help health orgs and researchers communicate insights clearly, analyse data, data storytelling—reach out for collaboration ! www.linkedin.com/in/tarancanada |

Responses (3)

Ishu

What are your thoughts?

EllaLiu
Sep 23, 2024

Very helpful!

12 1 reply Reply

Vijenderakuladotnet
Jul 17

Excellent !

Reply

Melroy van den Berg
Apr 27

What are your thoughts about denormalization vs multiple JOINs?

Lets say you have a MySQL/MariaDB tables where you need to do 2 or 3 JOINs in order to retrieve the column (eg. administration_id for example). Meaning you would need to constantly need... more

1 reply Reply

More from Taran Kaur and Women in Technology

Taran Kaur

Understanding Little's MCAR Test: A Key Tool in Missing Data Analysis

Ensuring Data Integrity: How Little's MCAR Test Detects Random Missingness.

Feb 10 4

In Women in Technology by Weiwei Hu

The Freedom of Choosing Beyond Yourself

Leadership is the courage to choose beyond yourself and to see those choices as the ste...

Oct 9 201 17

In Women in Technology by Tanushree

Part1: 20 Essential C# Interview Questions From a Decade of .NET...

This is the Part 1 of 3 part series which will test your C# knowledge. I have been workin...

Jan 13 77 1

Taran Kaur

A Comprehensive Guide to Data Imputation: Techniques,...

Filling the Gaps: How to Effectively Handle Missing Data for Accurate Analysis and...

Feb 12 17

See all from Taran Kaur

See all from Women in Technology

Recommended from Medium

Saumya Patel

Major Data Modeling Mistakes That Kill Scalability (and How to F...

Data modeling is like creating a blueprint for your data. Just as a well-designed building...

Oct 11 25

Code With Sunil | Code Smarter, not harder

Advanced SQL Interview Questions: 8 You Must Master in...

If you are not a Member — Read for free here

Oct 5 57

Dinesh Arney

Scaling the Data Layer with MySQL: A Detailed Exploration

1. SQL Data Layer Scalability Challenges

Apr 26 1

The Analytics Edge 🌟

Top 10 Ways to Use Indexing Effectively in SQL

Master SQL indexing to supercharge query speed and build scalable data systems.

Jun 28

Rohan Dutt

Ranking, Partitioning, and Deduplication—FAANG SQL...

Advanced SQL Techniques for Real-World FAANG Scenarios

6d ago 55 2

Adith - The Data Guy

MERGE in SQL

The UPSERT You Should Be Using

Aug 21

See more recommendations