SQL Performance Optimization Tips

20 Best Practices for Optimizing SQL DDL, DQL, and DML Operations

PART 2

Dan Vukota

linkedin.com/in/danvukota/

DQL (Data Query Language) Optimization

8. OR vs IN in Filtering

Most modern database engines optimize the IN operator more efficiently than OR. Internally, IN (x, y, ...) is often transformed into a range or hash lookup, which is typically faster, whereas OR may require multiple evaluations of the condition for each row, leading to less efficient query execution.

Example:

Use:

```
SELECT employee_id, name
FROM employees
WHERE department_id IN (3, 4);
```

Instead of:

```
SELECT employee_id, name
FROM employees
WHERE department_id = 3
OR department_id = 4;
```

9. Limit the Columns in the SELECT Clause

Instead of using SELECT *, specify only the necessary columns to reduce data transfer, minimize I/O operations, and lower memory usage, especially for large tables or columns with large data types (e.g., BLOB or TEXT).

Example:

Use:

```
SELECT order_id, order_date, quantity
FROM orders WHERE customer_id = 123;
```

Instead of:

```
SELECT *
FROM orders WHERE customer_id = 123;
```

10. Limit the Rows Returned

Limiting the rows returned using LIMIT (MySQL, PostgreSQL), TOP (SQL Server), or ROWNUM (Oracle) reduces the amount of data transferred and processed, which speeds up the query.

Example:

```
SELECT employee_id, salary FROM employees
ORDER BY salary DESC LIMIT 10;
```

11. Avoid Functions on Indexed Columns

When you apply a function (like UPPER(), DATE(), or CAST()) on an indexed column, it can prevent the database from using the index effectively. Try to avoid using functions on columns in WHERE or JOIN clauses if possible.

Example:

$f(x)$

If *name* column is indexed, try avoiding using the UPPER function if possiblel:

```
SELECT employee_id, name
FROM employees
WHERE UPPER(name) = 'JOHN DOE';
```

12. Avoid Redundant GROUP BY Columns

Avoid grouping by columns already used in the same type of aggregate functions, as it increases computation. However, there may be a use case for aggregating all rows and also including non-null values.

Example:

```
SELECT department_id, COUNT(*), COUNT(department_id)
FROM employees
GROUP BY department_id;
```

13. Using WHERE before HAVING

If you're filtering rows before aggregation (using WHERE), you're reducing the data set before performing any grouping and aggregation filtering which is generally more efficient.

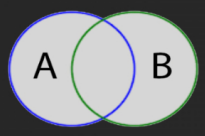
Example:

```
SELECT department_id, COUNT(employee_id)
FROM employees
WHERE department_id IN (3, 4)
GROUP BY department_id
HAVING COUNT(employee_id) > 10;
```

14. Use UNION ALL Instead of UNION

When combining multiple result sets, use UNION ALL instead of UNION if you don't need to remove duplicates. UNION sorts and removes duplicates, which can be resource-intensive, while UNION ALL simply appends the result sets.

Example:




```
SELECT employee_id FROM employees
WHERE department_id = 1
UNION ALL
SELECT employee_id FROM employees
WHERE department_id = 2;
```

15. Use Window Functions Instead of Subqueries for Ranking or Aggregation

Window functions like ROW_NUMBER(), RANK(), and SUM() OVER() can replace complex subqueries, improving efficiency by calculating rankings or aggregations in a single pass without requiring a correlated subquery.

Example:

Use:

```
SELECT employee_id, salary
FROM (
    SELECT employee_id, salary, AVG(salary) OVER () AS avg_salary
    FROM employees
) AS subquery
WHERE salary > avg_salary;
```

Instead of:

```
SELECT employee_id, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

16. Use WITH Wisely

While CTEs (Common Table Expressions) like WITH can make queries more readable, excessive or nested CTEs can harm performance. Be careful when using them for complex queries, as some database engines may not optimize CTEs well, and they might materialize intermediate results unnecessarily.

(CTE)

Example:

Use:

```
SELECT e.employee_id, e.name, COUNT(*)
OVER (PARTITION BY e.department_id) AS employee_count
FROM employees e;
```

Instead of:

```
WITH temp AS (
SELECT department_id, COUNT(*) AS employee_count
FROM employees
GROUP BY department_id
 )
 SELECT e.employee_id, e.name, t.employee_count
 FROM employees e
 INNER JOIN temp t ON e.department_id = t.department_id;
```

Thank You

Comment

Repost

Save

To be continued in Part 3 ...

linkedin.com/in/danvukota/

Dan Vukota