SQL Performance Optimization Tips

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

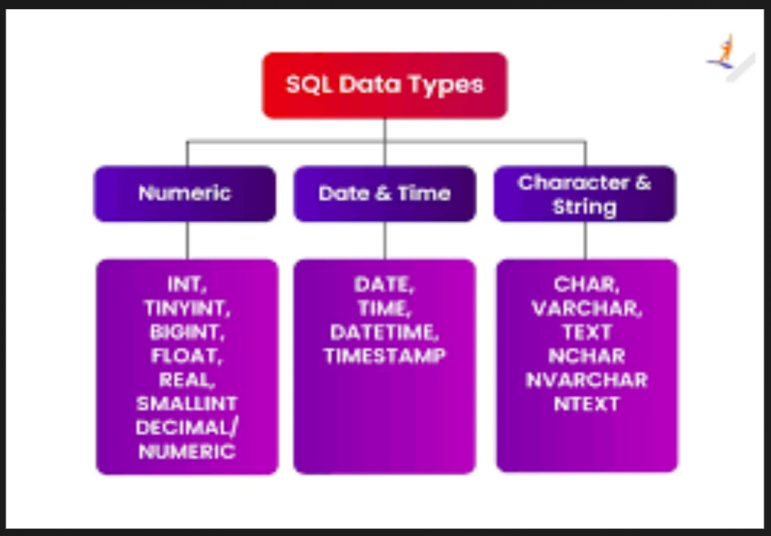
PART 1

Dan Vukota

linkedin.com/in/danvukota/

DDL (Data Definition Language) Optimization

1. Use the Right Data Types



When defining tables, using appropriate data types for your columns can improve query performance.

- Use INT for integers, BIGINT only if needed.
- Use DATE or TIMESTAMP for dates instead of VARCHAR.
- Use fixed-length strings (CHAR) over variable-length strings (VARCHAR) for small fixed-length data.
- Avoid TEXT unless required, as large text is slower to process.

2. Partitioning Large Tables

When creating very large tables, partitioning the table into smaller, more manageable pieces based on specific criteria can improve performance. Each partition is stored as a separate physical entity, and queries can target only the relevant partitions.

Example:

```
CREATE TABLE orders (
    order_id INT,
    order_date DATE,
    customer_id INT,
    quantity INT
)
PARTITION BY RANGE (order_date) (
    PARTITION p1 VALUES LESS THAN ('2023-01-01'),
    PARTITION p2 VALUES LESS THAN ('2024-01-01')
);
```

3. Indexing

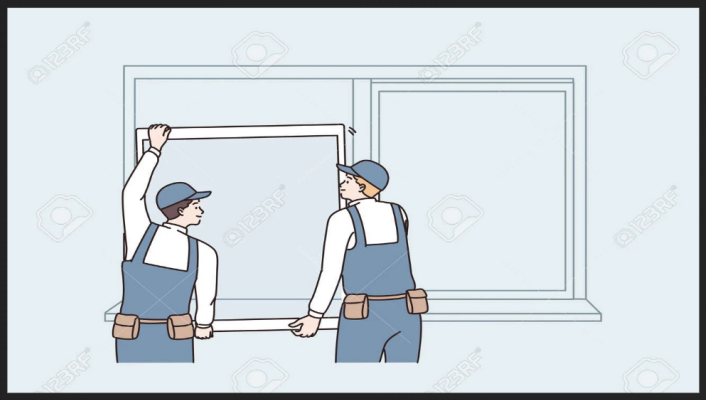
Where possible, make sure that the columns used in WHERE, JOIN, GROUP BY, and ORDER BY clauses are properly indexed. This helps SQL engines quickly locate the relevant rows rather than scanning entire tables.

Example:

```
CREATE INDEX idx_department_id ON employees(department_id);
```

4. Using Materialized Views for Expensive Queries

If a query is particularly expensive and is run frequently, you may benefit from creating a materialized view. This stores the results of the query in a physical table, which is updated periodically, so subsequent queries can be executed much faster.



Example:

```
CREATE MATERIALIZED VIEW avg_salary_per_dept
AS
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id;
```

DQL (Data Query Language) Optimization

5. Using LIKE Operator

The LIKE operator, especially with wildcard characters at the beginning of a string ('%text'), is often very slow because it requires a full scan of the table. Try to use patterns that can take advantage of indexes which are more efficient when the search is looking for exact matches or a pattern starting with a fixed string (e.g., 'text%').

Example:

Use:

```
SELECT employee_id, role
FROM employees
WHERE role LIKE 'IT%';
```

Instead of:

```
SELECT employee_id, role
FROM employees
WHERE role LIKE '%IT%';
```

6. Using BETWEEN Operator

Using BETWEEN for range filters is more efficient than using >= and <= separately because it is a single operation for checking a range, allowing the SQL engine to optimize execution. It reduces the overhead of evaluating two conditions, making the query plan simpler and faster compared to evaluating >= and <= independently.

Example:

Use:

```
SELECT order_id, order_date
FROM orders
WHERE order_date BETWEEN '2023-01-01'
AND '2023-12-31';
```

Instead of:

```
SELECT order_id, order_date
FROM orders
WHERE order_date >= '2023-01-01'
AND order_date <= '2023-12-31';
```

7. Use COALESCE for NULLs

COALESCE() can be more efficient than an IS NULL check, as it resolves the first non-NULL value in a series in a single operation, allowing the database to optimize it as a logical check rather than a separate NULL check.

Example:

Use:

```
SELECT employee_id, name
FROM employees
WHERE COALESCE(department_id, 0) = 0;
```

Instead of:

```
SELECT employee_id, name
FROM employees
WHERE department_id IS NULL;
```

Thank You

Comment

Repost

Save

To be continued in Part 2 ...

linkedin.com/in/danvukota/

Dan Vukota