Indexing

Indexing is a crucial technique for optimizing query performance. Indexes allow the database to find and retrieve specific rows much faster than without indexing.

Example:

```sql

-- Creating an index on the 'name' column of the 'employees' table

CREATE INDEX idx\_name ON employees (name);

• • • •

# Avoiding SELECT \*

Using SELECT \* retrieves all columns from the table, which can lead to inefficient query performance, especially if not all columns are needed.

Example:

```sql

-- Inefficient query

SELECT * FROM employees;

-- Optimized query

SELECT name, position FROM employees;

• • • •

Use JOINs instead of subqueries

JOINs are generally more efficient than subqueries and can result in better performance.

| Example: |
|---|
| ```sql |
| Using subquery |
| SELECT e.name, (SELECT d.department_name FROM departments d WHERE d.department_id = |
| e.department_id) |
| FROM employees e; |
| |
| Using JOIN |
| SELECT e.name, d.department_name |
| FROM employees e |
| JOIN departments d ON e.department_id = d.department_id; |
| |

Use WHERE clause to filter records

The WHERE clause is used to filter records, which helps in retrieving only the necessary data, thus optimizing the query.

Example:

""sql

SELECT name, position

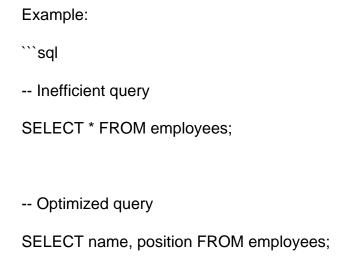
FROM employees

WHERE department_id = 3;

...

Limit the number of columns in SELECT

Only select the columns that are necessary for your query. This reduces the amount of data that needs to be processed and transferred.



Use EXISTS instead of IN

Using EXISTS is often more efficient than using IN, especially for larger datasets.

Example:

```sql

...

-- Using IN

SELECT name FROM employees WHERE department\_id IN (SELECT department\_id FROM departments WHERE location = 'NY');

-- Using EXISTS

SELECT name FROM employees WHERE EXISTS (SELECT 1 FROM departments WHERE departments.department\_id = employees.department\_id AND location = 'NY');

### **Avoid functions in WHERE clause**

Functions in the WHERE clause can slow down query performance because they are evaluated for every row in the result set.

### Example:

```sql

-- Using function

SELECT * FROM employees WHERE YEAR(join_date) = 2020;

-- Optimized query

SELECT * FROM employees WHERE join_date BETWEEN '2020-01-01' AND '2020-12-31';

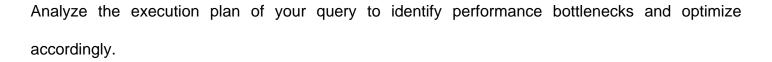
...

Use proper data types

• • • •

| Using appropriate data types can improve performance and reduce storage requirements. |
|---|
| Example: |
| ```sql |
| Inefficient data type |
| CREATE TABLE employees (phone_number VARCHAR(50)); |
| |
| Optimized data type |
| CREATE TABLE employees (phone_number CHAR(10)); |

Use query execution plans



Example:

```sql

-- Viewing execution plan

EXPLAIN SELECT \* FROM employees WHERE department\_id = 3;

• • • •

#### **Partitioning tables**

Partitioning can improve query performance by dividing a large table into smaller, more manageable pieces.

```
Example:

""sql
-- Creating a partitioned table

CREATE TABLE employees_part (
 employee_id INT,
 name VARCHAR(50),
 department_id INT,
 join_date DATE

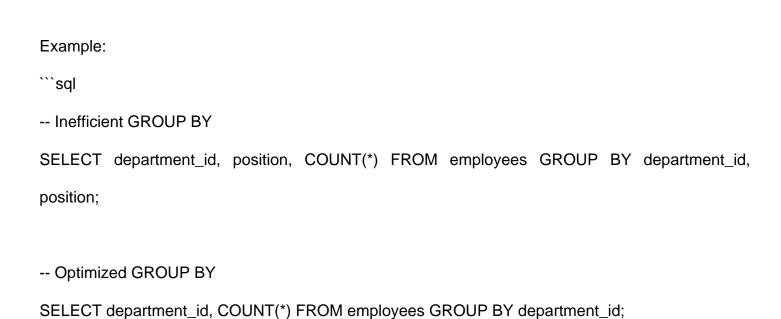
) PARTITION BY RANGE (YEAR(join_date)) (
 PARTITION p0 VALUES LESS THAN (2010),
 PARTITION p1 VALUES LESS THAN (2020),
 PARTITION p2 VALUES LESS THAN (2030)
);
```

# **Optimize JOIN operations**

| Ensure JOIN operations use indexes and are written efficiently.                    |
|------------------------------------------------------------------------------------|
| Example:                                                                           |
| ```sql                                                                             |
| Inefficient JOIN                                                                   |
| SELECT * FROM employees e, departments d WHERE e.department_id = d.department_id;  |
| Optimized JOIN                                                                     |
| SELECT * FROM employees e JOIN departments d ON e.department_id = d.department_id; |

### Avoid unnecessary columns in GROUP BY

Include only the necessary columns in the GROUP BY clause to reduce processing overhead.



#### Use table variables instead of temporary tables

Table variables are often faster and less resource-intensive than temporary tables.

```
Example:

""sql
-- Using temporary table

CREATE TABLE #TempTable (id INT);

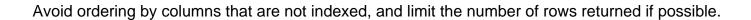
INSERT INTO #TempTable VALUES (1);

-- Using table variable

DECLARE @TempTable TABLE (id INT);

INSERT INTO @TempTable VALUES (1);
```

### **Optimize ORDER BY clause**



Example:

""sql
-- Inefficient ORDER BY

SELECT \* FROM employees ORDER BY name;

-- Optimized ORDER BY

SELECT TOP 10 \* FROM employees ORDER BY name;

### **Use SET NOCOUNT ON**

| Using SET NOCOUNT | ON can reduce network | k traffic between the d | atabase and application. |
|-------------------|-----------------------|-------------------------|--------------------------|

### **Avoid cursors**

SELECT name FROM employees;

| Cursors can be slow and resource-intensive; try to use set-based operations instead. |
|--------------------------------------------------------------------------------------|
| Example:                                                                             |
| ·                                                                                    |
| ```sql                                                                               |
| Using cursor                                                                         |
| DECLARE cursor_example CURSOR FOR SELECT name FROM employees;                        |
| OPEN cursor_example;                                                                 |
| FETCH NEXT FROM cursor_example;                                                      |
| CLOSE cursor_example;                                                                |
|                                                                                      |
| Using set-based operation                                                            |

#### **Use TRY-CATCH for error handling**

TRY-CATCH blocks can help manage errors more efficiently and improve code readability.

```
Example:

"``sql

BEGIN TRY

-- Try block code

SELECT 1 / 0; -- This will cause a divide by zero error

END TRY

BEGIN CATCH

-- Catch block code

SELECT ERROR_MESSAGE() AS ErrorMessage;

END CATCH;
```

### Parameter sniffing

...

Parameter sniffing can improve performance by using the parameter values of the first execution to generate the execution plan.

```
Example:

""sql

-- Example stored procedure

CREATE PROCEDURE GetEmployees

@DepartmentID INT

AS

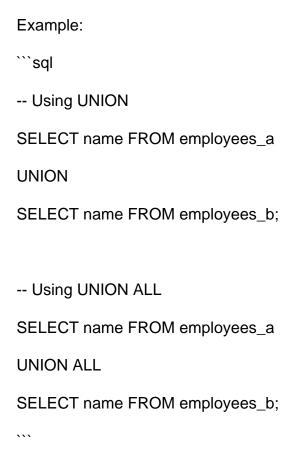
BEGIN

SELECT * FROM employees WHERE department_id = @DepartmentID;

END;
```

### **Use UNION ALL instead of UNION**

UNION ALL is faster than UNION because it does not remove duplicate rows.



# Use stored procedures for repetitive tasks

Stored procedures are precompiled and can improve performance for repetitive tasks.

| Example:                                                     |
|--------------------------------------------------------------|
| ```sql                                                       |
| Creating a stored procedure                                  |
| CREATE PROCEDURE GetEmployeesByDepartment                    |
| @DepartmentID INT                                            |
| AS                                                           |
| BEGIN                                                        |
| SELECT * FROM employees WHERE department_id = @DepartmentID; |
| END;                                                         |
|                                                              |

### Use indexes on foreign keys

Indexing foreign keys can improve the performance of JOIN operations.

Example:

"sql
-- Creating an index on the foreign key column

CREATE INDEX idx\_department\_id ON employees (department\_id);

#### Use derived tables

Derived tables can simplify complex queries and improve readability.

```
Example:

```sql

SELECT department_id, avg_salary

FROM (

SELECT department_id, AVG(salary) AS avg_salary

FROM employees

GROUP BY department_id

) AS derived_table;
```

Minimize locking and blocking

Use appropriate tra	insaction isolation	levels and	design	queries to	minimize	locking and b	ocking.

Example:

```sql

-- Setting transaction isolation level

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

...

### Use pagination for large result sets

...

Pagination can improve performance by retrieving only a subset of rows at a time.

Example:

""sql

SELECT \* FROM employees

ORDER BY employee\_id

OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;

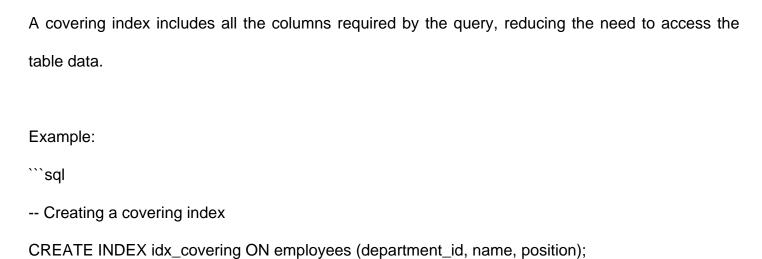
# **Update statistics regularly**

| Keeping statistics up to date helps the query optimizer make better decisions |
|-------------------------------------------------------------------------------|
|-------------------------------------------------------------------------------|

Example:
""sql
-- Updating statistics
UPDATE STATISTICS employees;

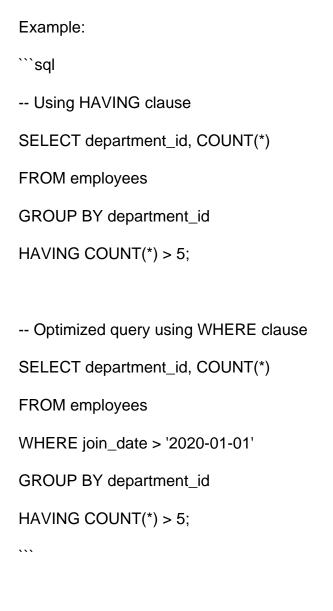
### **Use covering indexes**

...



#### **Optimize HAVING clause**

Use the HAVING clause only when necessary, and try to filter data using the WHERE clause first.



### **Use CROSS APPLY and OUTER APPLY**

CROSS APPLY and OUTER APPLY can be useful for joining table-valued functions.

```
Example:

""sql

SELECT e.name, d.department_name

FROM employees e

CROSS APPLY (

SELECT department_name

FROM departments d

WHERE d.department_id = e.department_id
) d;
```

#### **Use indexed views**

|         |             |           |               |            |          |           |                    | 41        | 1.2         |
|---------|-------------|-----------|---------------|------------|----------|-----------|--------------------|-----------|-------------|
| Indeved | VIEWS Can   | IMPLOVE   | nertormance t | tor comr   | ווט צפור | IPLIES PI | / materializing    | the VIEW  | result set  |
| HIGCACG | VICTVO CALL | IIIIPIOVO | pontonnanto   | ioi ooiiik |          |           | , illatolializilig | LIIC VICT | TOGUIL GOL. |

Example:

"'sql

-- Creating an indexed view

CREATE VIEW EmployeesByDepartment WITH SCHEMABINDING AS

SELECT department\_id, COUNT(\*) AS employee\_count

FROM dbo.employees

GROUP BY department\_id;

GO

CREATE UNIQUE CLUSTERED INDEX idx\_EmployeesByDepartment ON

EmployeesByDepartment (department\_id);

...

#### **Optimize INSERT operations**

Batching multiple INSERT operations into a single statement can improve performance.

```
Example:

""sql

-- Single row insert

INSERT INTO employees (name, department_id, position) VALUES ('John Doe', 1, 'Manager');

-- Batch insert

INSERT INTO employees (name, department_id, position) VALUES

('John Doe', 1, 'Manager'),

('Jane Smith', 2, 'Developer'),

('Jim Brown', 1, 'Analyst');
```

### **Use computed columns**



Example:

"`sql
-- Creating a computed column

ALTER TABLE employees ADD full\_name AS (first\_name + ' ' + last\_name);

### **Use appropriate JOIN types**

Choose the right type of JOIN (INNER, LEFT, RIGHT, FULL) based on your data retrieval needs.

Example:

""sql
-- Using INNER JOIN

SELECT e.name, d.department\_name

FROM employees e

INNER JOIN departments d ON e.department\_id = d.department\_id;

### **Reduce the use of DISTINCT**

| Avoid using DISTINCT unless it is necessary to eliminate duplicate rows. |
|--------------------------------------------------------------------------|
|--------------------------------------------------------------------------|

Example:

""sql
-- Using DISTINCT

SELECT DISTINCT name FROM employees;

-- Optimized query

SELECT name FROM employees GROUP BY name;

#### **Optimize CASE statements**

Simplify CASE statements to reduce complexity and improve readability.

```
Example:
```sql
-- Complex CASE statement
SELECT name, CASE
  WHEN department_id = 1 THEN 'HR'
  WHEN department_id = 2 THEN 'Engineering'
  ELSE 'Other'
END AS department_name
FROM employees;
-- Simplified CASE statement
SELECT name, CASE department_id
  WHEN 1 THEN 'HR'
  WHEN 2 THEN 'Engineering'
  ELSE 'Other'
END AS department_name
FROM employees;
```

Avoid excessive normalization

Over-normalization can lead to complex queries with many JOINs. Consider denormalization where
appropriate.
Example:
```sql
Highly normalized schema
SELECT e.name, d.department_name
FROM employees e
JOIN employee_departments ed ON e.employee_id = ed.employee_id

-- Denormalized schema

SELECT name, department_name

JOIN departments d ON ed.department_id = d.department_id;

FROM employees;

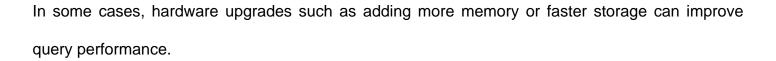
٠.,

### **Use columnstore indexes**

***

Columnstore indexes can significantly improve performance for read-heavy analytical queries.
Example:
```sql
Creating a columnstore index
CREATE COLUMNSTORE INDEX idx_employee_colstore ON employees (name, department_id,
position);

Consider hardware upgrades



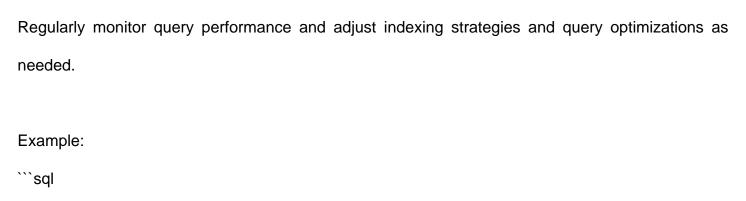
Example:

```sql

-- This technique is more about infrastructure rather than a code example

...

### Regularly monitor performance



-- Using monitoring tools and scripts to identify slow queries

SELECT \* FROM sys.dm\_exec\_query\_stats;

...