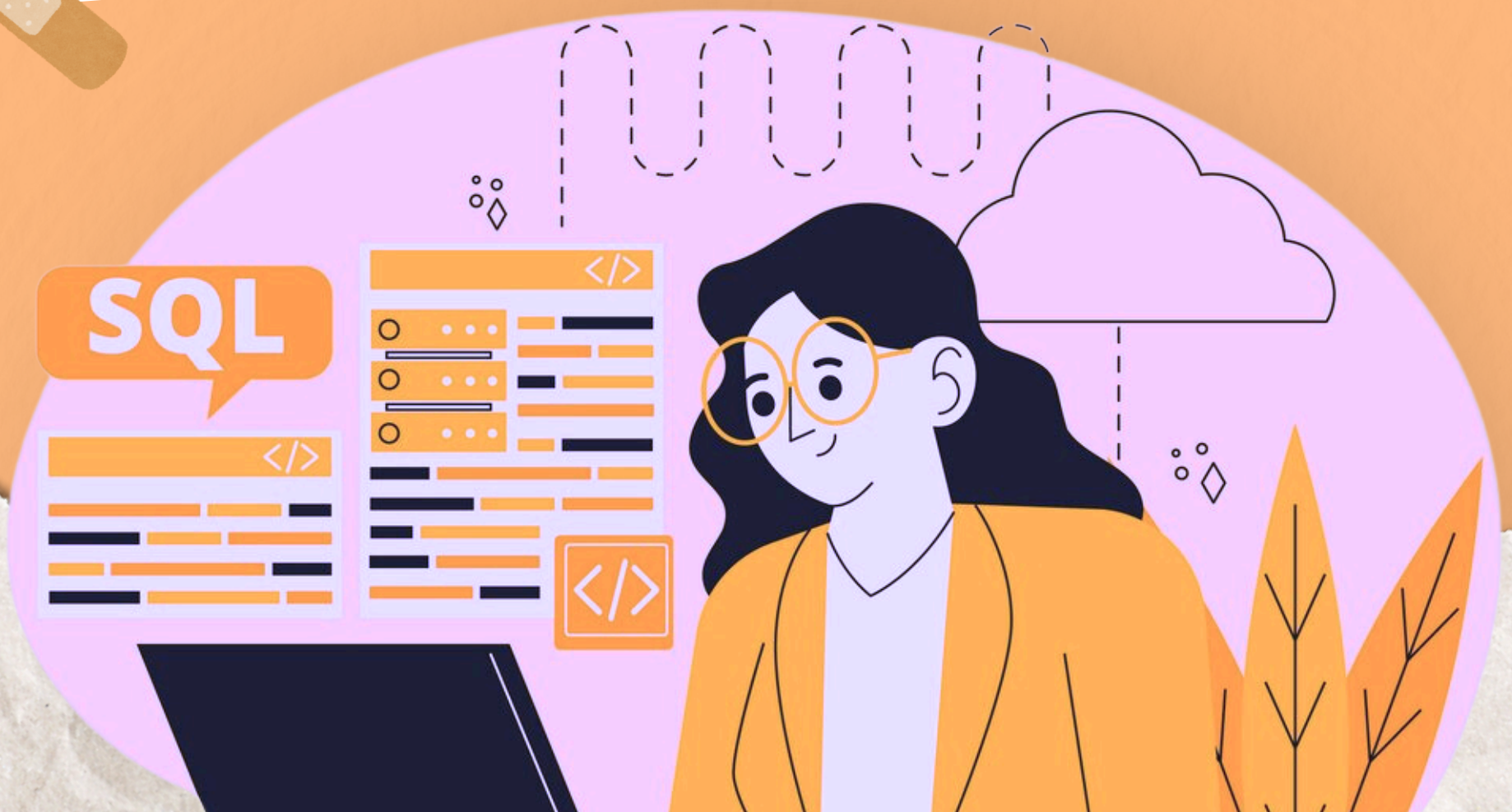
Yogesh Tyagi
@ytyagi782

# SQL Common Table Expression (CTE) Guide

## Simplify Complex Queries




Yogesh Tyagi
@ytyagi782

# "SQL Common Table Expression (CTE): Simplify and Organize Your Queries"

**Learn how to use CTEs for creating modular, readable, and reusable query structures.**

Yogesh Tyagi
@ytyagi782

# What is a Common Table Expression (CTE)?

A CTE is a temporary result set defined within a query for simplifying complex operations and improving query readability.

**Syntax:**

```
WITH cte_name AS (
    SELECT columns FROM table WHERE
condition
)
SELECT * FROM cte_name;
```


Yogesh Tyagi
@ytyagi782

# Why Use CTEs?

| | |
|---|---|
| **Improves Query Readability** | Breaks complex queries into smaller, logical parts. |
| **Reusable Logic** | Use the CTE result multiple times in the same query. |
| **Hierarchical Queries** | Handle recursive queries efficiently. |

Yogesh Tyagi
@ytyagi782

# Basic CTE Example

Simplify a query with a named temporary result set.

**Example**

```
WITH employee_cte AS (
     SELECT name, department_id FROM
employees WHERE salary > 5000
)
SELECT * FROM employee_cte WHERE
department_id = 3;
```

Yogesh Tyagi
@ytyagi782

# Recursive CTEs

**Handle hierarchical data like organizational charts or category trees using recursive CTEs.**

## Syntax

```
WITH RECURSIVE cte_name AS (
    SELECT base_query
    UNION ALL
    SELECT recursive_query
)
SELECT * FROM cte_name;
```

## POINT

```
WITH RECURSIVE org_chart AS (
    SELECT employee_id, name, manager_id
FROM employees WHERE manager_id IS NULL
    UNION ALL
    SELECT e.employee_id, e.name,
e.manager_id
    FROM employees e
    JOIN org_chart o ON e.manager_id =
o.employee_id
)
SELECT * FROM org_chart;
```


Yogesh Tyagi
@ytyagi782

# Using Multiple CTEs

Chain multiple CTEs together to handle complex transformations.

## Example

```sql
WITH sales_cte AS (
    SELECT product_id, SUM(sales) AS total_sales FROM sales GROUP BY product_id
), top_sales_cte AS (
    SELECT product_id FROM sales_cte WHERE total_sales > 1000
)
SELECT * FROM top_sales_cte;
```

Yogesh Tyagi
@ytyagi782

# Combining CTEs with Joins

CTEs can be joined with other tables or CTEs for further analysis.

**Example**

```sql
WITH employee_cte AS (
    SELECT name, department_id FROM employees
), department_cte AS (
    SELECT department_id, department_name FROM departments
)
SELECT e.name, d.department_name
FROM employee_cte e
JOIN department_cte d ON e.department_id = d.department_id;
```


Yogesh Tyagi
@ytyagi782

# CTEs vs Subqueries

| | |
|---|---|
| **CTEs** | Reusable, improve readability, and support recursion. |
| **Subqueries** | Compact but less readable for complex logic. |

## CTE Example

```
WITH high_salary_cte AS (
    SELECT name, salary FROM employees
WHERE salary > 5000
)
SELECT * FROM high_salary_cte;
```

## Subquery Example

```
SELECT * FROM (
    SELECT name, salary FROM employees
WHERE salary > 5000
) AS high_salary;
```


Yogesh Tyagi
@ytyagi782

# Recursive CTE Use Case: Calculating Factorial

## Recursively calculate a factorial using CTEs.

**Example**

```
WITH RECURSIVE factorial_cte AS (
    SELECT 1 AS n, 1 AS factorial
    UNION ALL
    SELECT n + 1, factorial * (n + 1)
    FROM factorial_cte
    WHERE n < 5
)
SELECT * FROM factorial_cte;
```

Yogesh Tyagi
@ytyagi782

# Common Use Cases for CTEs

## Simplifying Complex Joins

```
WITH employee_cte AS (
    SELECT * FROM employees WHERE department_id = 1
)
SELECT * FROM employee_cte;
```

## Simulated with TINYINT

```
WITH running_total_cte AS (
    SELECT product_id, SUM(sales) OVER
(PARTITION BY product_id ORDER BY
sale_date) AS running_total
    FROM sales
)
SELECT * FROM running_total_cte;
```

Yogesh Tyagi
@ytyagi782

# Nesting and Combining CTEs

Use nested or dependent CTEs to break down multi-step operations.

**Example**

```sql
WITH product_cte AS (
    SELECT product_id, category_id FROM products
), sales_cte AS (
    SELECT product_id, SUM(sales) AS total_sales FROM sales GROUP BY product_id
)
SELECT p.category_id, s.total_sales
FROM product_cte p
JOIN sales_cte s ON p.product_id = s.product_id;
```


Yogesh Tyagi
@ytyagi782

# Performance Tips for CTEs

- Avoid overly complex recursive CTEs for large datasets.
- Index columns used in CTEs to improve performance.
- Use CTEs for logical separation but ensure they do not increase query complexity unnecessarily.


Yogesh Tyagi
@ytyagi782

# Limitations of CTEs

**Temporary Scope**

CTEs exist only within the query.

**No Materialization**

Repeated use of the same CTE may lead to re-execution.

**Recursive Depth Limit**

Recursive CTEs have limits on iterations (configurable in some databases).


Yogesh Tyagi
@ytyagi782

# CTE vs Temporary Tables

| Feature | CTE | Temporary Tables |
|---|---|---|
| Scope | Query-specific | Session-specific |
| Performance | Lightweight, no storage overhead | Requires storage space |
| Usage | For query simplification | For intermediate data processing |

Yogesh Tyagi
@ytyagi782

# Wrap-Up

**"SQL CTEs: Simplify, Organize, Optimize!"**

**Master SQL Common Table Expressions to break down complex queries, improve readability, and handle recursive operations efficiently.**

**Yogesh Tyagi**

**@ytyagi782**

**Follow for More**