Mastering Common Table Expressions (CTEs) in MySQL

1. What are Common Table Expressions (CTEs)?

A CTE (Common Table Expression) is a temporary result set that you can reference within a SQL SELECT, INSERT, UPDATE, or DELETE statement.

Key Points:

- Defined using the WITH clause.
- Enhances query readability and reusability.
- Can be recursive or non-recursive.

Swipe →

2. Why Use CTEs?

- Simplifies complex queries by breaking them into logical components.
- Makes SQL code more modular and easier to understand.
- Improves maintainability by eliminating the need for subqueries or derived tables.

Swipe →

3. CTE Syntax in MySQL

The basic syntax for a CTE in MySQL:

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

The WITH clause creates the CTE, which is used in the SELECT query that follows.

Swipe →

4. Example of a Non-Recursive CTE

Using a CTE to calculate average salaries per department:

```
WITH DepartmentSalaries AS (
SELECT department_id, AVG(salary) AS
avg_salary
FROM employees GROUP BY department_id)
SELECT department_id, avg_salary
FROM DepartmentSalaries;
```

The CTE simplifies the query by breaking the salary calculation into a separate logical step.

Swipe

5.

Recursive CTEs: What Are They?

A recursive CTE refers to itself in the query and is used to solve hierarchical or tree-like data structures, such as finding a path in organizational structures.

Syntax of Recursive CTEs

Recursive CTEs have two parts:

1. Anchor member: The base query.
2. Recursive member: The part that refers to the CTE itself.

Swipe →

6. Example of a Non-Recursive CTE

```
WITH RECURSIVE EmployeeHierarchy AS (
SELECT employee_id, manager_id, 1 AS level
FROM employees
WHERE manager_id IS NULL
UNION ALL
SELECT e.employee_id, e.manager_id, eh.level + 1
FROM employees e
INNER JOIN EmployeeHierarchy eh ON
e.manager_id = eh.employee_id)
SELECT * FROM EmployeeHierarchy;
```

Swipe

5. Practical Applications of CTEs

- Simplifying Complex Queries: Break down long queries into manageable parts. Hierarchical
- Data: Recursive CTEs are perfect for organizational or tree structures. Reusability: CTEs can
- be referenced multiple times in the same query.

Swipe

6. CTE vs. Subquery:

CTE:

- Better readability.
- Can reference itself (recursive).
- Great for breaking down complex queries.

Subquery:

- Works well for simple tasks.
- Use when the query is small and does not need modularity.

Swipe →

5. When to Use CTEs?

Use CTEs when:

- The query is complex and needs modularity.
- You need to reference the same temporary result multiple times.
- Recursive queries are required, such as hierarchical data.

Swipe

→

Challenge yourself with real-world datasets. Write complex queries, apply CTEs, and explore recursive CTEs.