Common Table Expressions (CTEs) :
Syntax of CTE:

A CTE is defined using the `WITH` clause, followed by a query that specifies the CTE name and its columns. The CTE is then referenced in the main query.

Example 1: Basic CTE Syntax

```sql
WITH cte_name (column1, column2, ...) AS (
    -- CTE query
    SELECT column1, column2, ...
    FROM your_table
    WHERE condition
)
-- Main query referencing the CTE
SELECT *
FROM cte_name;
```

Recursive CTEs:

Recursive CTEs are a special type of CTE that can reference their own output. This is useful for working with hierarchical data, such as organizational charts or tree structures.

Example 2: Recursive CTE Syntax

```sql
WITH recursive_cte (id, name, manager_id) AS (
    -- Anchor member (non-recursive part)
    SELECT id, name, manager_id
    FROM employees
    WHERE manager_id IS NULL  -- Assuming NULL indicates top-level managers

    UNION ALL

    -- Recursive member
    SELECT e.id, e.name, e.manager_id
    FROM employees e
    JOIN recursive_cte r ON e.manager_id = r.id
)
-- Main query referencing the recursive CTE
SELECT *
FROM recursive_cte;
```

In this example, the CTE `recursive_cte` selects top-level managers in the anchor member (non-recursive part) and then recursively selects employees who report to those managers.

Certainly! Let's delve into more details about Common Table Expressions (CTEs) and Recursive CTEs with additional examples.

More on CTE Syntax:

CTEs can also be used to simplify and organize complex queries by allowing you to break them into modular, named blocks.

Example 3: Multiple CTEs in a Query

```sql
WITH cte1 AS (
    SELECT column1, column2
    FROM table1
),
cte2 AS (
    SELECT column3, column4
    FROM table2
    WHERE condition
)
-- Main query referencing multiple CTEs
SELECT cte1.column1, cte1.column2, cte2.column3, cte2.column4
FROM cte1
JOIN cte2 ON cte1.column1 = cte2.column3;
```

In this example, two CTEs (`cte1` and `cte2`) are defined, each encapsulating a part of the logic. The main query then joins these CTEs to retrieve the desired result.

More on Recursive CTEs:

Recursive CTEs are often used for traversing hierarchical data, like an organizational hierarchy. Let's explore a more elaborate example.

Example 4: Recursive CTE for Hierarchical Data

```sql
WITH recursive employee_hierarchy AS (
    SELECT employee_id, employee_name, manager_id
    FROM employees
    WHERE manager_id IS NULL -- Top-level managers

    UNION ALL

    SELECT e.employee_id, e.employee_name, e.manager_id
    FROM employees e
    JOIN employee_hierarchy r ON e.manager_id = r.employee_id
)
-- Main query to display the hierarchy
SELECT *
FROM employee_hierarchy
ORDER BY manager_id, employee_id;
```

In this example, the recursive CTE `employee_hierarchy` is used to build a hierarchical structure of employees and their managers. The main query then retrieves the hierarchical data, ordered by the manager's ID and employee's ID.

These examples showcase the flexibility and readability that CTEs bring to SQL queries, allowing you to break down complex logic into more manageable parts.