Database Using SQL Server & Raw Data available over my GitHub Profile

MySQL

PostgreSQL

12 Most Commonly Used CTE in the Industry

AnshLibrary

What is a CTE in SQL?

A **CTE (Common Table Expression)** is a **temporary result set** in SQL that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement.
It improves query readability, modularity, and is particularly useful for recursive queries and complex joins.

Syntax of a CTE

```
WITH CTE_Name AS (
 SELECT column1, column2
 FROM TableName
 WHERE condition )

SELECT * FROM CTE_Name;
```

Types of CTEs in SQL Server

1. Non-Recursive CTE
•Used to simplify complex joins, subqueries, or aggregations.
•Does **not** call itself.

2. Recursive CTE
A CTE that **calls itself** to return hierarchical or tree-structured data.
Has two parts: **Anchor Member** (initial result) and **Recursive Member** (repeated execution).

Non-Recursive CTE

```
WITH HighValueCustomers AS (
    SELECT CustomerID, SUM(OrderTotal) AS TotalSpent
    FROM Orders
    GROUP BY CustomerID
    HAVING SUM(OrderTotal) > 10000
)
SELECT * FROM HighValueCustomers;
```

Recursive CTE

```
WITH EmployeeHierarchy AS (
    SELECT EmployeeID, ManagerID, Name, 0 AS Level
    FROM Employees
    WHERE ManagerID IS NULL

 UNION ALL

    SELECT e.EmployeeID, e.ManagerID, e.Name, h.Level + 1
    FROM Employees e
    JOIN EmployeeHierarchy h ON e.ManagerID = h.EmployeeID
)
SELECT * FROM EmployeeHierarchy;
```

Anchor Member
(initial result)

Recursive Member
(repeated execution)

AnshLibrary

01

Recursive Hierarchy – Employee Manager Tree

```
WITH EmployeeCTE AS (
  SELECT EmployeeID, ManagerID, Name, 0 AS Level
  FROM Employees
  WHERE ManagerID IS NULL

  UNION ALL

  SELECT e.EmployeeID, e.ManagerID, e.Name,
c.Level + 1
  FROM Employees e
  INNER JOIN EmployeeCTE c ON e.ManagerID =
c.EmployeeID
)
SELECT * FROM EmployeeCTE ORDER BY Level;
```

Use Case : Organization hierarchy, reporting chain

AnshLibrary

02

Running Total Calculation

```
WITH SalesCTE AS (
  SELECT OrderID, OrderDate, Sales,
         SUM(Sales) OVER (ORDER BY OrderDate)
AS RunningTotal
  FROM Orders
)
SELECT * FROM SalesCTE;
```

Use Case : Trend analysis, cumulative reports

03

Get Duplicate Records

```
WITH DuplicateCTE AS (
  SELECT Name, COUNT(*) AS Cnt
  FROM Customers
  GROUP BY Name
  HAVING COUNT(*) > 1
)
SELECT * FROM DuplicateCTE;
```

Use Case : Data cleansing, quality checks.

AnshLibrary

04

Find the Nth Highest Salary

```
WITH RankedSalaries AS (
SELECT
  EmployeeID,
  Name,
  Salary,
  DENSE_RANK() OVER (ORDER BY  Salary DESC) AS Rank
FROM Employees
)

SELECT * FROM RankedSalaries WHERE Rank = 3;

--Use Rank = Nth number
```

Use Case : Compensation analysis, leaderboard ranking

AnshLibrary

05

Find First Order for Each Customer

```
WITH FirstOrders AS (
  SELECT *,
        ROW_NUMBER() OVER (PARTITION BY CustomerID
ORDER BY OrderDate) AS rn
  FROM Orders
)

SELECT * FROM FirstOrders WHERE rn = 1;
```

Use Case : Customer behavior analysis.

AnshLibrary

06

Latest Status Per Entity

```
WITH LatestStatus AS (

 SELECT *,
  ROW_NUMBER() OVER (PARTITION BY TicketID
  ORDER BY StatusDate DESC) AS rn
 FROM TicketStatus
)

SELECT * FROM LatestStatus WHERE rn = 1;
```

Use Case : Latest ticket or transaction state.

AnshLibrary

07

CTE with UPDATE Statement

```
WITH UpdatedSalaries AS (
  SELECT
     EmployeeID,
     Salary * 1.10 AS NewSalary
  FROM Employees
  WHERE Department = 'Sales'
)

UPDATE e
SET e.Salary = u.NewSalary
FROM Employees e
JOIN UpdatedSalaries u
ON e.EmployeeID = u.EmployeeID;
```

Use Case : Mass salary revision or adjustment.

AnshLibrary

08

Pagination Using CTE

```
WITH PagedOrders AS (
  SELECT *,
ROW_NUMBER() OVER (ORDER BY OrderDate
                        DESC) AS RowNum
  FROM Orders
)

SELECT * FROM PagedOrders
WHERE RowNum BETWEEN 11 AND 20;
```

Use Case: Backend pagination in web apps.

AnshLibrary

09

CTE for Monthly Sales Summary

```
WITH MonthlySales AS (
  SELECT
    YEAR(OrderDate) AS SalesYear,
    MONTH(OrderDate) AS SalesMonth,
    SUM(SalesAmount) AS TotalSales
  FROM Orders
  GROUP BY YEAR(OrderDate),
MONTH(OrderDate)
)

SELECT * FROM MonthlySales ORDER BY
SalesYear, SalesMonth;
```

Use Case: Generating month-wise sales reports.

AnshLibrary

Multiple CTEs with JOIN

```
WITH TopCustomers AS (
  SELECT
   CustomerID,
   SUM(OrderTotal) AS TotalSpent
  FROM Orders
  GROUP BY CustomerID
  HAVING SUM(OrderTotal) > 50000
),

RecentOrders AS (
  SELECT
   OrderID,
   CustomerID,
   OrderDate
  FROM Orders
  WHERE OrderDate >= DATEADD(MONTH, -6, GETDATE())
)

SELECT c.CustomerID, c.TotalSpent, r.OrderID,
r.OrderDate
FROM TopCustomers c
JOIN RecentOrders r ON c.CustomerID = r.CustomerID;
```

Use Case: Combine high spenders with their recent activities.

11

CTE for Ranking Products by Sales

```
WITH RankedProducts AS (
  SELECT
     ProductID,
     SUM(Quantity) AS TotalSold,
        RANK() OVER (ORDER BY SUM(Quantity)
        DESC) AS ProductRank
  FROM OrderItems
  GROUP BY ProductID
)

SELECT * FROM RankedProducts
WHERE ProductRank <= 5;
```

Use Case: Top 5 best-selling products..

AnshLibrary

12

CTE for Average vs Individual Salary

```
WITH DepartmentAvg AS (
  SELECT
    DepartmentID,
    AVG(Salary) AS AvgSalary
  FROM Employees
  GROUP BY DepartmentID
)

SELECT
   e.EmployeeID,
   e.FullName,
   e.Salary,
   d.AvgSalary
FROM Employees e
JOIN DepartmentAvg d
ON e.DepartmentID = d.DepartmentID
WHERE e.Salary > d.AvgSalary;
```

Use Case: Identify high-performing or overpaid employees

AnshLibrary