



SQL 5 Data Analyst Fellowship



Saksham Arora

**Software
Engineer**



[Saksham Arora - Microsoft | LinkedIn](#)

Common Table Expression (CTE)

- A common table expression, or CTE, is a temporary named result set created from a simple SELECT statement that can be used in a subsequent SELECT statement

We define CTEs by adding a WITH clause directly before SELECT, UPDATE or DELETE statement.

The WITH Clause can include one or more CTEs with the help of comma (,)

Common Table Expression (CTE)

- Syntax

```
WITH my_cte AS (
    SELECT a,b,c
    FROM Table1 )
SELECT a,c
FROM my_cte
```

The diagram illustrates the structure of a Common Table Expression (CTE). It shows two main components: the CTE query and the main query. The CTE query is defined by the `WITH` clause and its associated parentheses, containing a `SELECT` statement. The main query follows the CTE definition and contains another `SELECT` statement. Orange curly braces on the right side group the CTE definition and the main query together, while a separate orange brace groups just the main query.

Example Table for CTEs -

	employee_id [PK] integer	name character varying (100)	department character varying (100)	salary integer
1	1	John Doe	HR	50000
2	2	Jane Smith	IT	60000
3	3	Alice Johnson	Finance	55000
4	4	Bob Brown	IT	70000
5	5	Charlie White	HR	48000

Basic CTE Example

```
18 WITH it_employees AS (
19     SELECT employee_id, name, salary, department
20     FROM employees
21     WHERE department LIKE 'IT'
22 )
23     SELECT * FROM it_employees;
```

Data Output Messages Notifications



	employee_id [PK] integer	name character varying (100)	salary integer	department character varying (100)
1	2	Jane Smith	60000	IT
2	4	Bob Brown	70000	IT

Multiple CTE

Query Query History

```
25  WITH hr_employees AS (
26      SELECT employee_id, name, salary
27      FROM employees
28      WHERE department LIKE 'HR'
29  ),
30  it_employees AS (
31      SELECT employee_id, name, salary
32      FROM employees
33      WHERE department LIKE 'IT'
34  )
35  SELECT * FROM hr_employees
36  UNION ALL
37  SELECT * FROM it_employees;
```

Data Output Messages Notifications

≡+ ⌂ ↻ ⌂ ↻ ⌂ ⌂ ⌂ ⌂ SQL

	employee_id integer	name character varying (100)	salary integer
1	1	John Doe	50000
2	5	Charlie White	48000
3	2	Jane Smith	60000
4	4	Bob Brown	70000

Recursive CTE

Table -

```
CREATE TABLE employee_hierarchy (
    employee_id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    manager_id INT
);
```

```
INSERT INTO employee_hierarchy (name, manager_id)
VALUES
('CEO', NULL),
('CTO', 1),
('HR Head', 1),
('IT Manager', 2),
('HR Specialist', 3),
('Developer', 4);
```

55
56 ▾ **SELECT * FROM employee_hierarchy**
57

Data Output Messages Notifications

≡+ ↻ ⌂ ↻ 🗑️ 📁 ↻ 🗑️ 📡 ↻ 📈 ↻ SQL

	employee_id [PK] integer	name character varying (100)	manager_id integer
1	1	CEO	[null]
2	2	CTO	1
3	3	HR Head	1
4	4	IT Manager	2
5	5	HR Specialist	3
6	6	Developer	4

```
WITH RECURSIVE employee_tree AS (
    -- Base case: start with the CEO (the top level)
    SELECT employee_id, name, manager_id, 1 AS level
    FROM employee_hierarchy
    WHERE manager_id IS NULL

    UNION ALL

    -- Recursive case: find employees who report to the employees in the previous
    result
    SELECT e.employee_id, e.name, e.manager_id, et.level + 1
    FROM employee_hierarchy e
    INNER JOIN employee_tree et ON e.manager_id = et.employee_id
)
SELECT employee_id, name, level
FROM employee_tree
ORDER BY level;
```

Base Case:

- Starts with the `CEO` (i.e., `manager_id IS NULL`), returning the CEO's `employee_id`, `name`, and level `1`.

sql

 Copy code

employee_id	name	manager_id	level
1	CEO	NULL	1

First Recursive Call:

- The first recursive step joins the `CEO`'s `employee_id` with the `manager_id` in the `employee_hierarchy` table to find direct reports of the `CEO`.
- The query returns employees reporting directly to the CEO (like `CTO` and `HR Head`), with their level now being `2`.

diff

 Copy code

<code>employee_id</code>	<code>name</code>	<code>manager_id</code>	<code>level</code>
2	CTO	1	2
3	HR Head	1	2

Second Recursive Call:

- Now the recursive part looks for employees reporting to CTO (employee_id 2) and HR Head (employee_id 3).
- It finds IT Manager reporting to the CTO and HR Specialist reporting to HR Head, and adds them to the result set with level = 3.

diff

 Copy code

employee_id	name	manager_id	level
4	IT Manager	2	3
5	HR Specialist	3	3

Assign Category to all employees based on salary

- salary ≥ 70000 then display category as HIGH
- salary ≥ 55000 and salary < 70000 then display category as MEDIUM
- salary < 55000 then display category as LOW

The screenshot shows a database interface with a toolbar at the top containing various icons for data management. Below the toolbar is a table with four columns: employee_id, salary, and category. The category column contains values HIGH, MEDIUM, and LOW, corresponding to the salary ranges defined in the assignment.

	employee_id integer	salary integer	category text
1	4	70000	HIGH
2	2	60000	MEDIUM
3	3	55000	MEDIUM
4	1	50000	LOW
5	5	48000	LOW

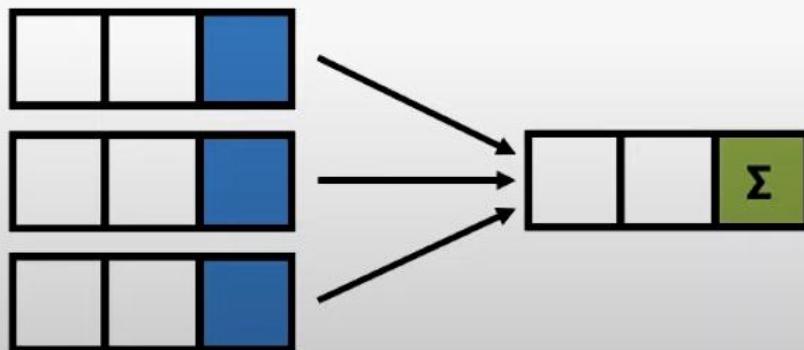
When to Use CTEs:

- When breaking down **complex** queries.
- When needing **intermediate results** that are reused.
- When dealing with **recursive or hierarchical data**.
- When improving the **readability** of long queries.
- When using **window functions** or **ranking**.

WINDOW FUNCTION

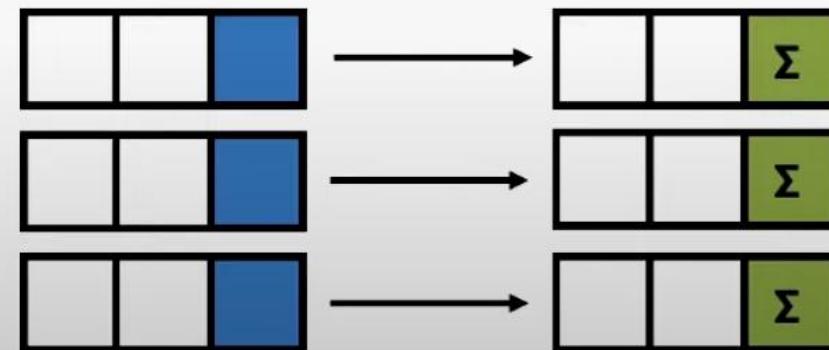
- **Window functions** applies aggregate, ranking and analytic functions over a particular window (set of rows).
- And **OVER** clause is used with window functions to define that window.

Aggregate Functions (SUM, AVG, etc.)



Give output one row per aggregation

Window Functions



The rows maintain their separate identities

WINDOW FUNCTION SYNTAX

```
SELECT column_name(s),  
      fun( ) OVER ( [ <PARTITION BY Clause> ]  
                  [ <ORDER BY Clause> ]  
                  [ <ROW or RANGE Clause> ] )  
FROM table_name
```

Select a function

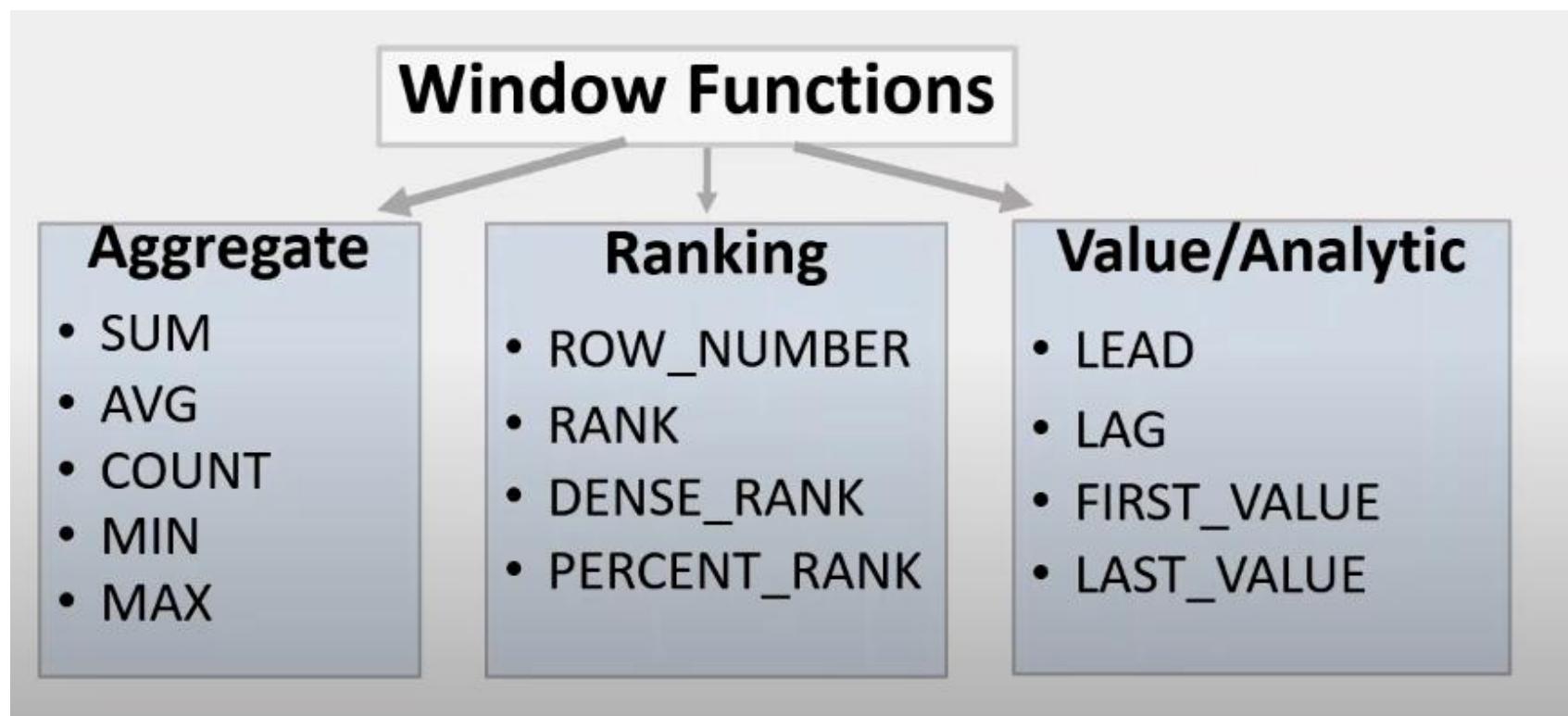
- Aggregate functions
- Ranking functions
- Analytic functions

Define a Window

- PARTITION BY
- ORDER BY
- ROWS

WINDOW FUNCTION TYPES

There is no official division of the SQL window functions into categories but high level we can divide into three types



Example Table for Window Functions -

Data Output Messages Notifications

	employee_id [PK] integer	name character varying (100)	department character varying (100)	salary integer
1	1	John Doe	IT	60000
2	2	Jane Smith	HR	50000
3	3	Alice Johnson	Finance	55000
4	4	Bob Brown	IT	70000
5	5	Charlie White	HR	48000
6	6	Diana Green	Finance	62000
7	7	Eve Black	IT	62000

```

26 SELECT employee_id, department, salary,
27   SUM(salary) OVER(PARTITION BY department ORDER BY salary) as "CUMULATIVE_SUM",
28   AVG(salary) OVER(PARTITION BY department ORDER BY salary) as "WINDOW_AVG",
29   COUNT(employee_id) OVER(PARTITION BY department ORDER BY salary) as "ORDER_GROUPWISE",
30   MAX(salary) OVER(PARTITION BY department ORDER BY salary) as "PREFIX_MAX",
31   MIN(salary) OVER(PARTITION BY department ORDER BY salary) as "PREFIX_MIN"
32 FROM employees

```

Data Output Messages Notifications 

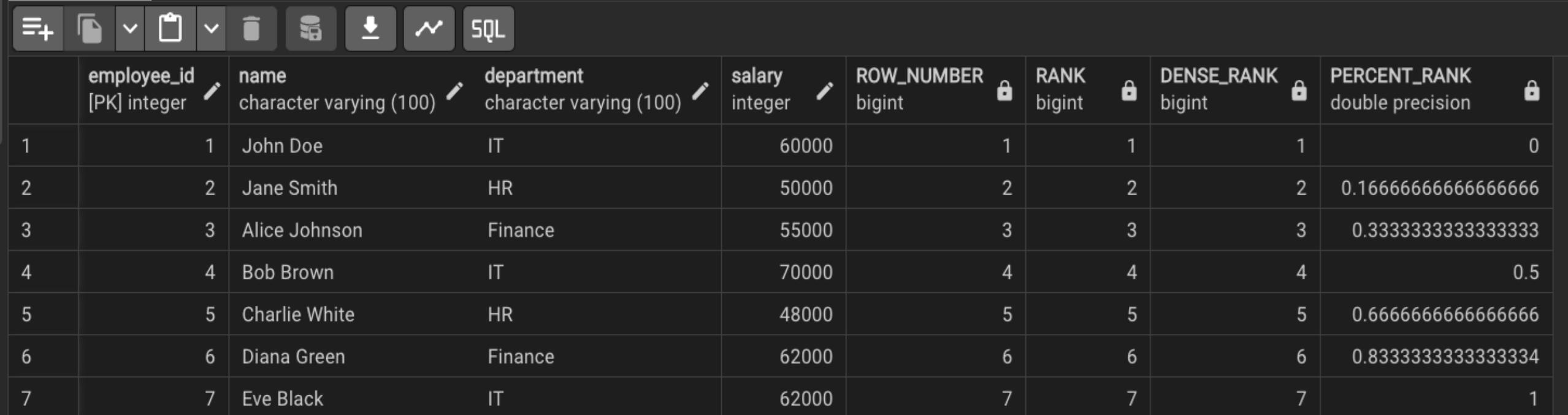


	employee_id [PK] integer	department character varying (100)	salary integer	CUMULATIVE_SUM bigint	WINDOW_AVG numeric	ORDER_GROUPWISE bigint	PREFIX_MAX integer	PREFIX_MIN integer
1	3	Finance	55000	55000	55000.000000000000	1	55000	55000
2	6	Finance	62000	117000	58500.000000000000	2	62000	55000
3	5	HR	48000	48000	48000.000000000000	1	48000	48000
4	2	HR	50000	98000	49000.000000000000	2	50000	48000
5	1	IT	60000	60000	60000.000000000000	1	60000	60000
6	7	IT	62000	122000	61000.000000000000	2	62000	60000
7	4	IT	70000	192000	64000.000000000000	3	70000	60000

RANK functions -

```
17 ▾ SELECT *,  
18   ROW_NUMBER() OVER(ORDER BY employee_id) as "ROW_NUMBER",  
19   RANK() OVER(ORDER BY employee_id) as "RANK",  
20   DENSE_RANK() OVER(ORDER BY employee_id) as "DENSE_RANK",  
21   PERCENT_RANK() OVER(ORDER BY employee_id) as "PERCENT_RANK"  
22   FROM employees  
23
```

Data Output Messages Notifications



The screenshot shows a database interface with a toolbar at the top and a table below it. The table displays data from the 'employees' table, including calculated rank-related columns.

	employee_id [PK] integer	name character varying (100)	department character varying (100)	salary integer	ROW_NUMBER bigint	RANK bigint	DENSE_RANK bigint	PERCENT_RANK double precision
1	1	John Doe	IT	60000	1	1	1	0
2	2	Jane Smith	HR	50000	2	2	2	0.1666666666666666
3	3	Alice Johnson	Finance	55000	3	3	3	0.3333333333333333
4	4	Bob Brown	IT	70000	4	4	4	0.5
5	5	Charlie White	HR	48000	5	5	5	0.6666666666666666
6	6	Diana Green	Finance	62000	6	6	6	0.8333333333333334
7	7	Eve Black	IT	62000	7	7	7	1

RANK functions -

```
SELECT new_id,  
       ROW_NUMBER() OVER( ORDER BY new_id) AS "ROW_NUMBER",  
       RANK() OVER( ORDER BY new_id) AS "RANK",  
       DENSE_RANK() OVER( ORDER BY new_id) AS "DENSE_RANK",  
       PERCENT_RANK() OVER( ORDER BY new_id) AS "PERCENT_RANK"  
FROM test_data
```

new_id	ROW_NUMBER	RANK	DENSE_RANK	PERCENT_RANK
100	1	1	1	0
200	2	2	2	0.166
200	3	2	2	0.166
300	4	4	3	0.5
500	5	5	4	0.666
500	6	5	4	0.666
700	7	7	5	1

```

34  SELECT employee_id, department, salary,
35    FIRST_VALUE(employee_id) OVER(ORDER BY employee_id) as "FIRST_VALUE",
36    LAST_VALUE(employee_id) OVER(ORDER BY employee_id) as "LAST_VALUE",
37    LEAD(employee_id) OVER(ORDER BY employee_id) as "LEAD",
38    LAG(employee_id) OVER(ORDER BY employee_id) as "LAG"
39  FROM employees

```

Data Output Messages Notifications



	employee_id [PK] integer	department character varying (100)	salary integer	FIRST_VALUE integer	LAST_VALUE integer	LEAD integer	LAG integer
1	1	IT	60000	1	1	2	[null]
2	2	HR	50000	1	2	3	1
3	3	Finance	55000	1	3	4	2
4	4	IT	70000	1	4	5	3
5	5	HR	48000	1	5	6	4
6	6	Finance	62000	1	6	7	5
7	7	IT	62000	1	7	[null]	6

Questions to practice -

<https://leetcode.com/problems/department-highest-salary/description/>

<https://leetcode.com/problems/rank-scores/description/>

<https://nextleap.app/interview-preparation/sql/questions>



Q&A

What's on your mind?

A man with dark hair and a beard is pointing his index finger towards the camera. He is wearing a black t-shirt with a red and green plaid pattern. In the background, there is a large, glowing purple eye with a yellow ring, set against a red and orange gradient. To the right, there are several colorful, pixelated characters from a video game, including a yellow creature and a brown bear-like character. The overall style is a meme or a reaction image.

I have several questions.

Feed us back!