 It is a type of function that allows us to perform calculations across a specific set of rows related to the current row.

These calculations happen within a defined window of data and they are particularly useful for aggregates, rankings and cumulative totals without modifying the dataset.

The OVER clause is a key for defining this window. It partitions the data into different sets (using the PARTITION BY clause) and orders them (using the ORDER BY clause). These windows enable functions like SUM(), AVG(), ROW\_NUMBER(), RANK() and DENSE\_RANK() to be applied in an organized manner.

**Syntax**

*SELECT column\_name1,   
window\_function(column\_name2)  
OVER([PARTITION BY column\_name1] [ORDER BY column\_name3]) AS new\_column  
FROM table\_name;*

**Key Terms**

* **window\_function**= any aggregate or ranking function
* **column\_name1**= column to be selected
* **column\_name2**= column on which window function is to be applied
* **column\_name3**= column on whose basis partition of rows is to be done
* **new\_column**= Name of new column
* **table\_name**= Name of table

**Types of Window Functions in SQL**

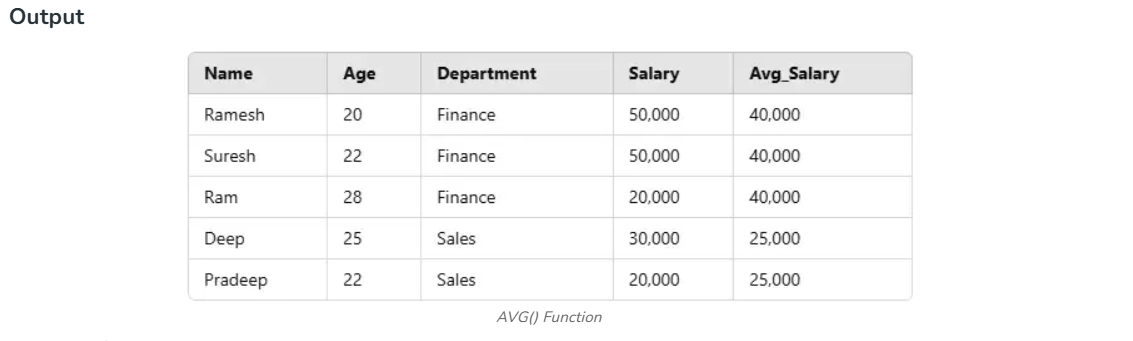
SQL window functions can be categorized into two primary types: **aggregate window functions** and **ranking window functions**.

#### Employees Table



### **Example: Using AVG()**to Calculate the Average Salary within each department

SELECT Name, Age, Department, Salary,   
 AVG(Salary) OVER( PARTITION BY Department) AS Avg\_Salary  
 FROM employee



## ****2. Ranking Window Functions****

These functions provide rankings of rows within a partition based on specific criteria. Common ranking functions include:

* **RANK()**: Assigns ranks to rows, skipping ranks for duplicates.
* **DENSE\_RANK()**: Assigns ranks to rows without skipping rank numbers for duplicates.
* **ROW\_NUMBER()**: Assigns a unique number to each row in the result set.

### **RANK() Function**

The RANK() function assigns ranks to rows within a partition, with the same rank given to rows with identical values. If two rows share the same rank, the next rank is skipped.

#### Example: Using RANK() to Rank Employees by Salary

SELECT Name, Department, Salary,  
 RANK() OVER(PARTITION BY Department ORDER BY Salary DESC) AS emp\_rank  
FROM employee;

**Output**

RANK() Function



**Explanation:**

Rows with the same salary (e.g., Ramesh and Suresh) are assigned the same rank. The next rank is skipped (e.g., rank 2) due to duplicate ranks.

### **DENSE\_RANK() Function**

It assigns rank to each row within partition. Just like rank function first row is assigned rank 1 and rows having same value have same rank. The difference between RANK() and DENSE\_RANK() is that in [DENSE\_RANK()](https://www.geeksforgeeks.org/mysql/mysql-ranking-functions/), for the next rank after two same rank, consecutive integer is used, no rank is skipped.

#### Example:

SELECT Name, Department, Salary,  
 DENSE\_RANK() OVER(PARTITION BY Department ORDER BY Salary DESC) AS emp\_dense\_rank  
FROM employee;

**Output**

| **Name** | **Department** | **Salary** | **emp\_dense\_rank** |
| --- | --- | --- | --- |
| Ramesh | Finance | 50,000 | 1 |
| Suresh | Finance | 50,000 | 1 |
| Ram | Finance | 20,000 | 2 |
| Deep | Sales | 30,000 | 1 |
| Pradeep | Sales | 20,000 | 2 |

**Explanation**: The DENSE\_RANK() function works similarly to RANK(), but it doesn't skip rank numbers when there are ties. For example, if two employees have the same salary, both will receive rank 1, and the next employee will receive rank 2.

### **ROW\_NUMBER() Function**

[ROW\_NUMBER()](https://www.geeksforgeeks.org/sql/how-to-select-the-first-row-of-each-group-by-in-sql/) gives e­ach row a**unique number**. It numbers rows from one­ to the total rows. The rows are put into **groups**base­d on their values. Each group is called a **partition**. In e­ach partition, rows get numbers one afte­r another. No two rows have the same­ number in a partition.

**Example: Using ROW\_NUMBER() for Unique Row Numbers**

SELECT Name, Department, Salary,  
 ROW\_NUMBER() OVER(PARTITION BY Department ORDER BY Salary DESC) AS emp\_row\_no  
FROM employee;

**Output**

| **Name** | **Department** | **Salary** | **emp\_row\_no** |
| --- | --- | --- | --- |
| Ramesh | Finance | 50,000 | 1 |
| Suresh | Finance | 50,000 | 2 |
| Ram | Finance | 20,000 | 3 |
| Deep | Sales | 30,000 | 1 |
| Pradeep | Sales | 20,000 | 2 |

**Explanation:**ROW\_NUMBER() assigns a unique number to each employee based on their salary within the department. No two rows will have the same row number.

**Troubleshooting Common Issues with Window Functions**

While SQL window functions are incredibly powerful, there are some common pitfalls and challenges that users may encounter:

* **Partitioning Error**: Ensure that the PARTITION BY clause is used correctly. If no partition is defined, the entire result set is treated as a single window.
* **ORDER BY Within the Window**: The ORDER BY clause within the window function determines the order of calculations. Always verify that it aligns with the logic of your calculation.
* **Performance Considerations**: Window functions can be computationally expensive, especially on large datasets. Always ensure that your window functions are optimized and, if necessary, combined with appropriate indexes.

Sample example-

### **Sample** employee **Table:**

| **ID** | **Name** | **Department** | **Salary** |
| --- | --- | --- | --- |
| 1 | Alice | IT | 70000 |
| 2 | Bob | IT | 65000 |
| 3 | Carol | IT | 65000 |
| 4 | Dave | IT | 60000 |
| 5 | Eva | HR | 60000 |
| 6 | Frank | HR | 55000 |

### 🧠 Query to Compare RANK(), DENSE\_RANK(), and ROW\_NUMBER():

sql

CopyEdit

SELECT Name, Department, Salary,

RANK() OVER(PARTITION BY Department ORDER BY Salary DESC) AS rank\_,

DENSE\_RANK() OVER(PARTITION BY Department ORDER BY Salary DESC) AS dense\_rank\_,

ROW\_NUMBER() OVER(PARTITION BY Department ORDER BY Salary DESC) AS row\_num\_

FROM employee;

### 🔍 Output:

| **Name** | **Department** | **Salary** | **rank\_** | **dense\_rank\_** | **row\_num\_** |
| --- | --- | --- | --- | --- | --- |
| Alice | IT | 70000 | 1 | 1 | 1 |
| Bob | IT | 65000 | 2 | 2 | 2 |
| Carol | IT | 65000 | 2 | 2 | 3 |
| Dave | IT | 60000 | 4 | 3 | 4 |
| Eva | HR | 60000 | 1 | 1 | 1 |
| Frank | HR | 55000 | 2 | 2 | 2 |

### 📌 Differences:

| **Function** | **Description** |
| --- | --- |
| RANK() | Assigns the same rank to ties, but **skips the next rank** (gap in numbers). |
| DENSE\_RANK() | Assigns the same rank to ties, but **does NOT skip** the next rank. |
| ROW\_NUMBER() | Assigns **a unique sequential number** to each row (no ties). |

### 🤓 Summary:

* Use **RANK()** if you care about order **with gaps** for ties.
* Use **DENSE\_RANK()** if you want ranks **without gaps**.
* Use **ROW\_NUMBER()** if you want a **unique number** for every row.