Window functions applies aggregate and ranking functions over a particular window (set of rows). OVER clause is used with window functions to define that window. OVER clause does two things :

- Partitions rows into form set of rows. (PARTITION BY clause is used)
- Orders rows within those partitions into a particular order. (ORDER BY clause is used)

Note -

If partitions aren't done, then ORDER BY orders all rows of table.

Basic Syntax :

```
SELECT coulmn_name1,
 window_function(cloumn_name2),
 OVER([PARTITION BY column_name1] [ORDER BY column_name3]) AS new_column
FROM table_name;
```

window_function= any aggregate or ranking function
column_name1= column to be selected
coulmn_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

Aggregate Window Function :

Various aggregate functions such as SUM(), COUNT(), AVERAGE(), MAX(), MIN() applied over a particular window (set of rows) are called aggregate window functions.

Consider the following **employee** table :

Name	Age	Department	Salary

Ramesh	**20**	**Finance**	**50, 000**
Deep	25	Sales	30, 000
Suresh	22	Finance	50000
Ram	28	Finance	20, 000
Pradeep	22	Sales	20, 000

Example -

Find average salary of employees for each department and order employees within a department by age.

```
SELECT Name, Age, Department, Salary,
 AVERAGE(Salary) OVER( PARTITION BY Department ORDER BY Age) AS Avg_Salary
 FROM employee
```

The output of above query will be :

Name	**Age**	**Department**	**Salary**	**Avg_Salary**
Ramesh	20	Finance	50, 000	40, 000
Suresh	22	Finance	50000	40, 000
Ram	28	Finance	20, 000	40, 000
Pradeep	22	Sales	20, 000	25, 000
Deep	25	Sales	30, 000	25, 0000

As we can see in above example, the average salary within each department is calculated and displayed in column Avg_Salary. Also, employees within particular column are ordered by their age.

Ranking Window Functions :

Ranking functions are, RANK(), DENSE_RANK(), ROW_NUMBER()

- **RANK() -**
  As the name suggests, the rank function assigns rank to all the rows within every partition. Rank is assigned such that rank 1 given to the first row and rows having same value are assigned same rank. For the next rank after two same rank values, one rank value will be skipped.

- **DENSE_RANK() -**
  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(), for the next rank after two same rank, consecutive integer is used, no rank is skipped.

- **ROW_NUMBER() -**
  It assigns consecutive integers to all the rows within partition. Within a partition, no two rows can have same row number.

Note -

ORDER BY() should be specified compulsorily while using rank window functions.

Example -

Calculate row no., rank, dense rank of employees is employee table according to salary within each department.

```
SELECT
ROW_NUMBER() OVER (PARTITION BY Department ORDER BY Salary DESC)
AS emp_row_no, Name,  Department, Salary,
RANK() OVER(PARTITION BY Department
ORDER BY Salary DESC) AS emp_rank,
DENSE_RANK() OVER(PARTITION BY Department
        ORDER BY Salary DESC)
        AS emp_dense_rank,
FROM employee
```

The output of above query will be :

emp_row_no	Name	Department	Salary	emp_rank	emp_dense_rank
1	Suresh	Finance	50, 000	1	1
2	Ramesh	Finance	50, 000	1	1
3	Ram	Finance	20, 000	3	2
1	Deep	Sales	30, 000	1	1
2	Pradeep	Sales	20, 000	2	2

So, we can see that as mentioned in the definition of ROW_NUMBER() the row numbers are consecutive integers within each partition. Also, we can see difference between rank and dense rank that in dense rank there is no gap between rank values while there is gap in rank values after repeated rank.