SQL RANK() function is a window function used in **SQL Server** that calculates the rank of each row of the **result set**.

RANK Function in SQL Server

The RANK function in the SQL server is used to assign a rank to each row based on its value.

The same rank is assigned to the rows which have the same values.

RULES:

- It always works with the OVER() clause.
- It assigns a rank to each row based on the ORDER BY clause.
- It assigns a rank to each row in consecutive order.
- It always assigns a rank to rows, starting with one for each new partition.

SYNTAX:

```
RANK() OVER (
  [PARTITION BY expression, ]
  ORDER BY expression (ASC | DESC) );
```

EXAMPLE:

```
SELECT SALARY,RANK() OVER(ORDER BY SALARY) AS RANKING
FROM EMP
```

OUTPUT:

```
"salary"          "ranking"
300             1
800             2
1100            3
1250            4
1250    4
1300    6
1500    7
1600    8
2450    9
2850    10
2975    11
3000    12
3000    12
```

5000   14

SYNTAX:2

```
1. SELECT column_name
2. RANK() OVER (
3.     PARTITION BY expression
4.     ORDER BY expression [ASC|DESC])
5. AS 'my_rank' FROM table_name;
```

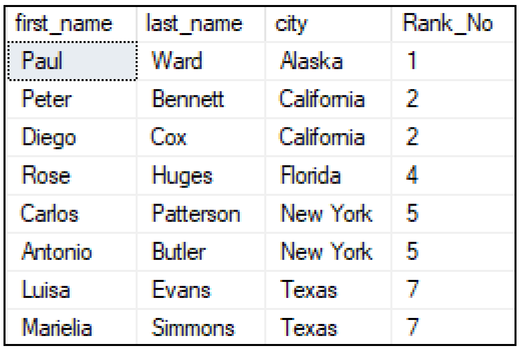
EX:

Example

Let us see how the RANK() function works in SQL Server. The below statement will use the rank function to assign numbering for each row:

```
1. SELECT first_name, last_name, city,
2. RANK () OVER (ORDER BY city) AS Rank_No
3. FROM rank_demo;
```

Since we have not used the **PARTITION BY clause**, the function treated the whole result as a single partition. Executing the statement will display the below output:



first_name	last_name	city	Rank_No
Paul	Ward	Alaska	1
Peter	Bennett	California	2
Diego	Cox	California	2
Rose	Huges	Florida	4
Carlos	Patterson	New York	5
Antonio	Butler	New York	5
Luisa	Evans	Texas	7
Marielia	Simmons	Texas	7

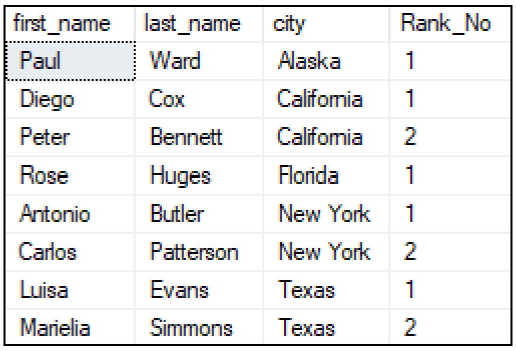
In this output, we can see that some of the rows get the same rank because they have the same value in the **city column**. And the next number in the ranking will be its previous rank plus a number of duplicate numbers.

The following statement is **another example** where we are going to use a partition by clause that will divide the rows based on the **city** column and assign a ranking to each row within a partition. The order of the output is based on the **first_name**:

```
1. SELECT first_name, last_name, city,
```

2. RANK () OVER (PARTITION BY city ORDER BY first_name) AS Rank_No
3. FROM rank_demo;

It will display the below output:



first_name	last_name	city	Rank_No
Paul	Ward	Alaska	1
Diego	Cox	California	1
Peter	Bennett	California	2
Rose	Huges	Florida	1
Antonio	Butler	New York	1
Carlos	Patterson	New York	2
Luisa	Evans	Texas	1
Marielia	Simmons	Texas	2

ROW_NUMBER() Function

This function is used to **return the unique sequential number for each row** within its partition. The row numbering begins at one and increases by one until the partition's total number of rows is reached. It will return the different ranks for the row having similar values that make it different from the RANK() function.

The below syntax illustrates the use of a ROW_NUMBER() function in SQL Server:

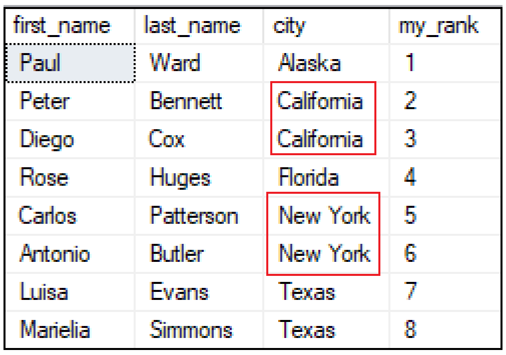
```
1. SELECT column_name
2. ROW_NUMBER() OVER (
3.     PARTITION BY expression
4.     ORDER BY expression [ASC|DESC])
5. AS 'rank_name' FROM table_name;
```

Example

Execute the following query to assign a sequence number for each row:

```
1. SELECT first_name, last_name, city,
2. ROW_NUMBER() OVER(ORDER BY city) AS ROW_NUMBERING
3. FROM rank_demo;
```

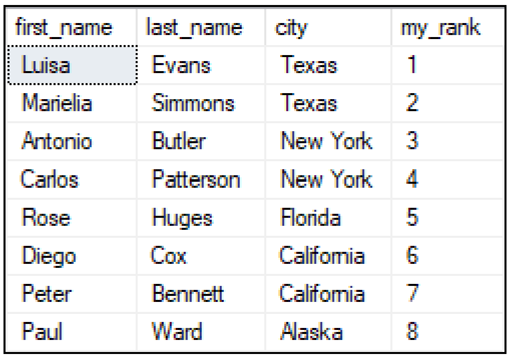
It will assign the ranking for the table as per their **city**. Here we can see that it assigns different ranks for the row which has the same city values.



first_name	last_name	city	my_rank
Paul	Ward	Alaska	1
Peter	Bennett	California	2
Diego	Cox	California	3
Rose	Huges	Florida	4
Carlos	Patterson	New York	5
Antonio	Butler	New York	6
Luisa	Evans	Texas	7
Marielia	Simmons	Texas	8

If we change the sorting order from **ascending to descending** with the ORDER BY clause, this function will also change the RANK accordingly. See the below statement:

1. SELECT first_name, last_name, city,
2. ROW_NUMBER() OVER(ORDER BY city DESC) AS ROW_NUMBERING
3. FROM rank_demo;



first_name	last_name	city	my_rank
Luisa	Evans	Texas	1
Marielia	Simmons	Texas	2
Antonio	Butler	New York	3
Carlos	Patterson	New York	4
Rose	Huges	Florida	5
Diego	Cox	California	6
Peter	Bennett	California	7
Paul	Ward	Alaska	8

DENSE_RANK() Function

This function assigns **a unique rank for each row within a partition** as per the specified column value without any gaps. It always specifies ranking in consecutive order. If we get a **duplicate value**, this function will assign it with the same rank, and the next rank being the next sequential number. This characteristic differs DENSE_RANK() function from the RANK() function.

The below syntax illustrates the use of a DENSE_RANK() function in SQL Server:

1. SELECT column_name
2. DENSE_RANK() OVER (
3.   PARTITION BY expression
4.   ORDER BY expression [ASC|DESC])

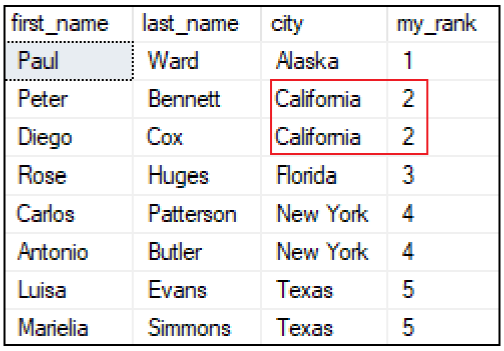
5. AS 'rank_name' FROM table_name;

Example

The following query uses the DENSE_RANK() function to assign a rank number for each row:

1. SELECT first_name, last_name, city,
2. DENSE_RANK() OVER(ORDER BY city) AS my_rank
3. FROM rank_demo;

It will return the below output where we can see that the duplicate values have the same rank, and the following rank will be the next sequential number.

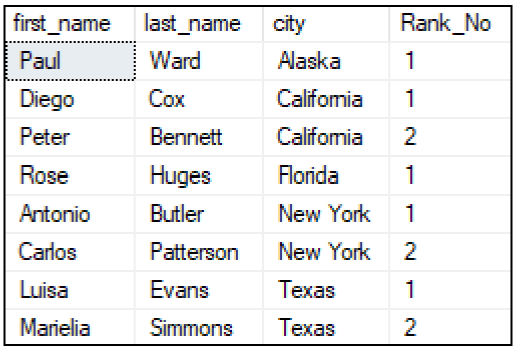


first_name	last_name	city	my_rank
Paul	Ward	Alaska	1
Peter	Bennett	California	2
Diego	Cox	California	2
Rose	Huges	Florida	3
Carlos	Patterson	New York	4
Antonio	Butler	New York	4
Luisa	Evans	Texas	5
Marielia	Simmons	Texas	5

t is another example of the DENSE_RANK() function by using the PARTITION BY clause. This clause will divide the rows based on the city column, and the order of a result set is based on the first_name:

1. SELECT first_name, last_name, city,
2. DENSE_RANK() OVER(PARTITION BY city ORDER BY first_name) AS my_rank
3. FROM rank_demo;

We will get the below output because no two names are the same. In this case, the output is similar to the RANK() function.

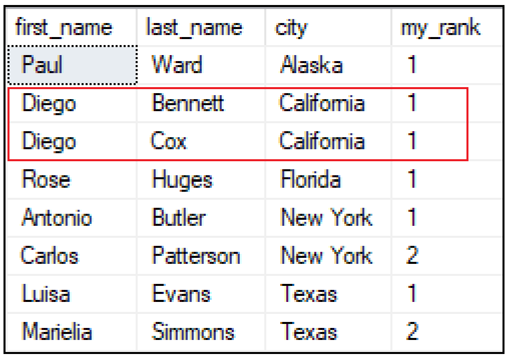


first_name	last_name	city	Rank_No
Paul	Ward	Alaska	1
Diego	Cox	California	1
Peter	Bennett	California	2
Rose	Huges	Florida	1
Antonio	Butler	New York	1
Carlos	Patterson	New York	2
Luisa	Evans	Texas	1
Marielia	Simmons	Texas	2

Let's update the name with the following query:

1. Update rank_demo set first_name = 'Diego' WHERE city = 'California'

Now, execute the same query again. We will see that this table got the same name in **California City**. Therefore, rank is also the same for both names.



first_name	last_name	city	my_rank
Paul	Ward	Alaska	1
Diego	Bennett	California	1
Diego	Cox	California	1
Rose	Huges	Florida	1
Antonio	Butler	New York	1
Carlos	Patterson	New York	2
Luisa	Evans	Texas	1
Marielia	Simmons	Texas	2

NTILE(N) Function

This function is used to **distribute rows of an ordered partition into a pre-defined number** (N) of approximately equal groups. Each row group gets its rank based on the defined condition and starts numbering from one group. It assigns a **bucket number** for every row in a group representing the group to which it belongs.

The following syntax illustrates the use of a NTILE() function in SQL Server:

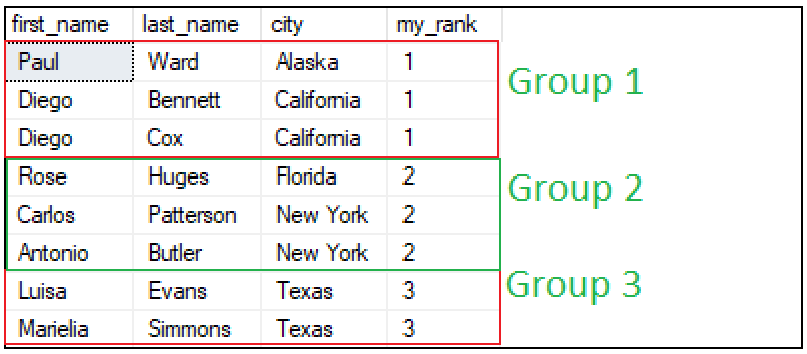
```
1.  SELECT column_name
2.  NTILE(N) OVER (
3.     PARTITION BY expression
4.     ORDER BY expression [ASC|DESC])
5.  AS 'my_rank' FROM table_name;
```

Example

The following query uses the NTILE() function to assign a rank number for each row:

1. SELECT first_name, last_name, city,
2. NTILE(3) OVER(ORDER BY city) AS my_rank
3. FROM rank_demo;

The specified table has **eight records**. Therefore, the **NTILE(3)** tells that the result set must have a **group of three records**. Executing the statement will display the below output:



first_name	last_name	city	my_rank
Paul	Ward	Alaska	1
Diego	Bennett	California	1
Diego	Cox	California	1
Rose	Huges	Florida	2
Carlos	Patterson	New York	2
Antonio	Butler	New York	2
Luisa	Evans	Texas	3
Marielia	Simmons	Texas	3

Group 1
Group 2
Group 3

WINDOWS FUNCTIONS:

Window functions apply to 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 to form a 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 the table.

Syntax:

```
SELECT column_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
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

Aggregate Window Function

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

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

```
SELECT Name, Age, Department, Salary,

 AVG(Salary) OVER( PARTITION BY Department) AS Avg_Salary

 FROM employee
```

Let's consider another case:

```
SELECT Name, Age, Department, Salary,

 AVG(Salary) OVER( PARTITION BY Department ORDER BY Age) AS
Avg_Salary

 FROM employee
```

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;
```