Windows Functions
SQL

Window functions operate on a set of rows and return a single value for each row from the underlying query.

The term window describes the set of rows on which the function operates.

A window function uses values from the rows in a window to calculate the returned values.

Basic Syntax :

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

A window function in a query, define the window using the OVER() clause.

The OVER() clause (window definition) differentiates window functions from other analytical and reporting functions.

A query can include multiple window functions with the same or different window definitions.

The OVER() clause has the following capabilities:

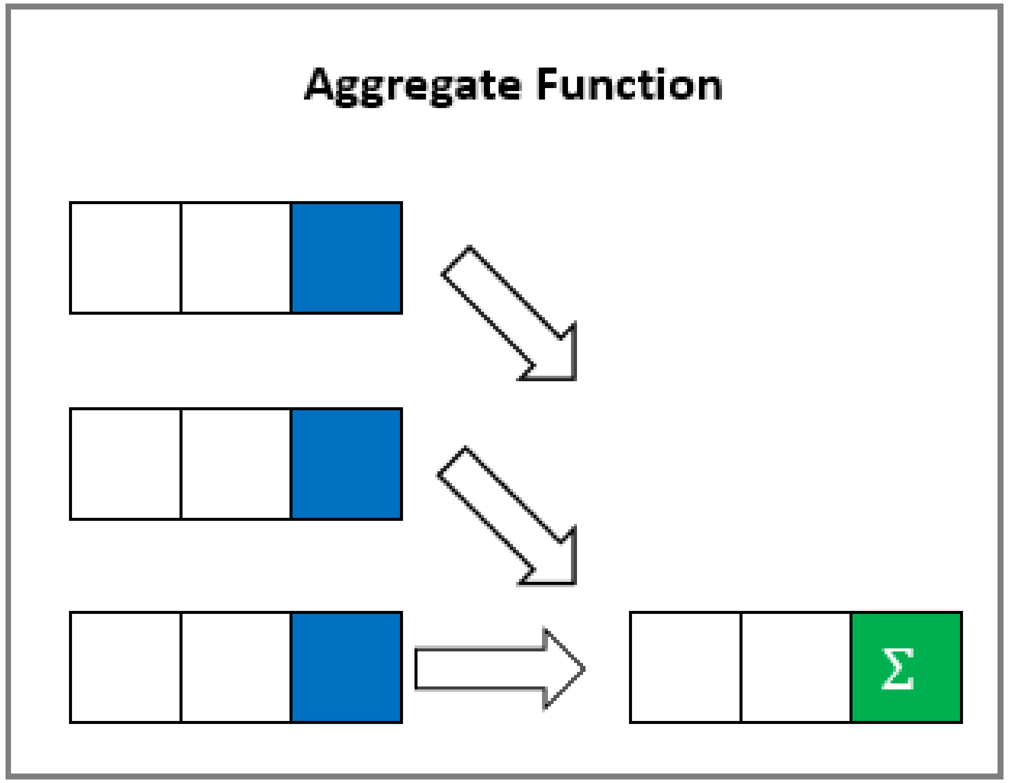
✓Defines window partitions to form groups of rows (PARTITION BY clause).

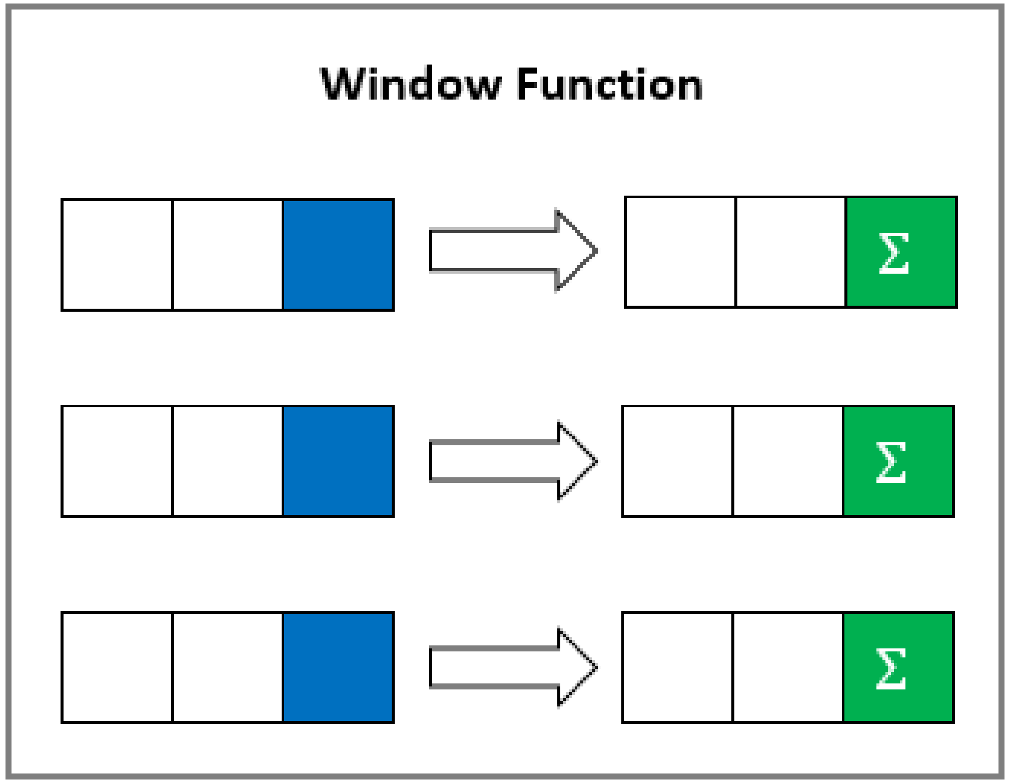
✓Orders rows within a partition (ORDER BY clause).

The sum ()as a window function.It returns the sum salary of all employees along with the salary of each individual employee.

```
SELECT  first_name,   last_name,   salary,   SUM(salary) OVER() sum_salary
FROM    employees;
```

The over()clause signals that the sum()function is used as a window function





SQL window function types

The window functions are divided into three types value window functions, aggregation window functions, and ranking window functions

Value window functions

FIRST_VALUE():-returns the first value in an ordered set of values

Query: **SELECT** first_name, last_name, salary, **FIRST_VALUE** (first_name) **OVER** (**ORDER BY** salary) lowest_salary **FROM** employees e;

FIRST_VALUE()-over partition:returns employee who have lowest salary in each department.

Query:**SELECT** first_name, last_name, department_name, salary, **FIRST_VALUE** (**CONCAT**(first_name,' ',last_name)) **OVER** (**PARTITION BY** department_name **ORDER BY** salary) lowest_salary **FROM** employees e **INNER JOIN** departments d **ON** d.department_id = e.department_id;

LAG():-SQL LAG() is a window function that provides access to a row at a specified physical offset which comes before the current row.

In other words, by using the LAG() function, from the current row, you can access data of the previous row, or from the second row before the current row, or from the third row before current row, and so on.

The LAG() function can be very useful for calculating the difference between the current row and the previous row.

Query:**SELECT** employee_id, fiscal_year, salary, LAG(salary) **OVER** (**PARTITION BY** employee_id **ORDER BY** fiscal_year) previous_salary **FROM** basic_pays;

Note:

Create a table basic pay with emp_id,fiscalyear,salary

LAST_VALUE():

The LAST_VALUE() is a window function that returns the last value in an ordered set of values.

Query: **SELECT** first_name, last_name, salary, **LAST_VALUE** (first_name) **OVER** (**ORDER BY** salary **RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING**) highest_salary **FROM** employees;

Using SQL LAST_VALUE() over partition example

Query: SELECT first_name, last_name,department_name,salary,LAST_VALUE (CONCAT(first_name,' ',last_name)) OVER (PARTITION BY department_name ORDER BY salary RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) highest_salary FROM

employees e   INNER JOIN departments d ON d.department_id = e.department_id;

LEAD()

SQL LEAD() is a window function that provides access to a row at a specified physical offset which follows the current row.

The LEAD() function, from the current row, you can access data of the next row, or the second row that follows the current row, or the third row that follows the current row, and so on

The LEAD() function can be very useful for calculating the difference between the value of the current row and the value of the following row

The following statement returns, for each employee in the company, the hire date of the employee hired just after

Query:

```
SELECT         first_name,  last_name,  hire_date,   LEAD(hire_date, 1) OVER (
               ORDER BY hire_date        ) AS next_hiredFROM      employees;
```

Ranking window functions

CUME_DIST() :The CUME_DIST() is a window function that calculates the cumulative distribution of value within a set of values.

The CUME_DIST() function returns a value that represents the number of rows with values less than or equal to (<=)the current row’s value divided by the total number of rows

The return value of the CUME_DIST() function has a range of the low value greater than 0 and the high value less than or equal to 1.

Query:
```
CREATE VIEW department_headcounts AS SELECT
department_name, COUNT(employee_id) headcount FROM
employees e INNER JOIN departments d ON
d.department_id = e.department_id GROUP BY
e.department_id;
```

```
SELECT department_name, headcount, ROUND( CUME_DIST()
OVER ( ORDER BY headcount ) ,2) cume_dist_val FROM
department_headcounts;
```

SQL DENSE_RANK():

The DENSE_RANK() is a window function that assigns ranks to rows in partitions with no gaps in the ranking values.

If two or more rows in each partition have the same values, they receive the same rank. The next row has the rank increased by one.

Different from the RANK() function, the DENSE_RANK() function always generates consecutive rank values.

Query:

```
CREATE TABLE t ( col CHAR ); INSERT INTO t(col)
VALUES('A'),('B'),('B'),('C'),('D'),('D'),('E');
SELECT * FROM t;
```

SELECT col,DENSE_RANK() OVER (ORDER BY col) my_dense_rank,

RANK() OVER (ORDER BY col) my_rank FROM t;

```
SELECT employee_id, first_name, last_name, salary,
DENSE_RANK() OVER ( ORDER BY salary DESC )
salary_rank FROM employees;
```

SQL DENSE_RANK():

```
SELECT first_name, last_name, department_name,
salary, DENSE_RANK() OVER ( PARTITION BY
department_name ORDER BY salary DESC) salary_rank
FROM employees e INNER JOIN departments d ON
d.department_id = e.department_id;
```

```
SELECT
	*
FROM (
	SELECT
		first_name,
		last_name,
		department_name,
		salary,
		DENSE_RANK() OVER (
			PARTITION BY department_name
			ORDER BY salary DESC) salary_rank
	FROM
		employees e
		INNER JOIN departments d
			ON d.department_id = e.department_id
	) t
WHERE
	salary_rank = 1;
```

SQL NTILE() function:

The SQL NTILE() is a window function that allows you to break the result set into a specified number of approximately equal groups, or buckets. It assigns each group a bucket number starting from one

For each row in a group, the NTILE() function assigns a bucket number representing the group to which the row belongs.

Query:

```
CREATE TABLE t ( col INT NOT NULL ); INSERT INTO
t(col)
VALUES(1),(2),(3),(4),(5),(6),(7),(8),(9),(10);
SELECT * FROM t;
SELECT col, NTILE (3) OVER (ORDER BY col) buckets FROM t;
SELECT
        first_name,
        last_name,
        salary,
        NTILE(5) OVER (
                ORDER BY salary DESC
        ) salary_group
FROM
        employees;
```

SELECT first_name, last_name, salary,NTILE(5) OVER (ORDER BY salary DESC

) salary_group FROM   employees;

PERCENT_RANK()

The PERCENT_RANK() is a window function that calculates the percentile ranking of rows in a result set.

Query:

SELECT first_name,   last_name,   salary, ROUND(PERCENT_RANK() OVER (ORDER BY salary) ,2) percentile_rank FROM employees;

```
SELECT first_name, last_name, salary, department_name,
ROUND( PERCENT_RANK() OVER ( PARTITION BY
e.department_id ORDER BY salary ) ,2) percentile_rank
FROM employees e INNER JOIN departments d ON
d.department_id = e.department_id;
```

RANK() function

The RANK() function is a window function that assigns a rank to each row in the partition of a result set.

```
RANK() OVER (
        PARTITION BY <expr1>[{,<expr2>...}]
        ORDER BY <expr1> [ASC|DESC], [{,<expr2>...}]
)
```

First, the PARTITION BY clause distributes the rows in the result set into partitions by one or more criteria.

Second, the ORDER BY clause sorts the rows in each a partition.

The RANK() function is operated on the rows of each partition and re-initialized when crossing each partition boundary.

CREATE TABLE t (col CHAR);

INSERT INTO t(col)VALUES('A'),('B'),('B'),('C'),('D'),('D'),('E');

SELECT *FROM t;

```
SELECT col, RANK() OVER ( ORDER BY col ) myrank FROM
t;
```

```
SELECT first_name, last_name, salary, RANK() OVER
(ORDER BY salary) salary_rank FROM employees;
```

ROW_NUMBER() Function:

The ROW_NUMBER() is a window function that assigns a sequential integer number to each row in the query's result set.

Query:

```
SELECT ROW_NUMBER() OVER ( ORDER BY salary ) row_num,
first_name, last_name, salary FROM employees;
```

```
SELECT * FROM (
    SELECT
        ROW_NUMBER() OVER (ORDER BY salary) row_num,
        first_name,
        last_name,
        salary
    FROM
        employees
    ) t
WHERE
    row_num > 10 AND row_num <=20;
```