

MASTERING JOINS, VIEWS, WINDOW FUNCTIONS IN SQL

-by katham shruthi

What are Joins?

In simpler terms, Joins are used to combine the rows from two or more tables, considering that there is at least one common column between the tables. The Joins operations will be performed based on the common column.

Why Joins are Important?



Most of the times, dealing with complex data often requires working with multiple tables, there comes a need where the tables must be joined to retrieve data, this necessitates the need for joining the tables.

Example:

Consider the below table:

Table1: Customers

CustomerID	FirstName	LastName
1	John	Peter
2	Anita	Martin
3	Michael	Shah

Table2: Sales

ProductId	Name	CustomerID
101	Mobile	1
102	Groceries	2
103	Books	3

In the above example, if the customers who purchased any of the products need to be retrieved, joining both the tables becomes essential, the joining operation will be performed on the field Customer_Id as it is the column which is common in both the tables.

Isn't it feasible to put all the data under one single table?

While discussing about joints, there comes a question:

“Why can't we store all the data into single table, which eliminates the operation, joins?”

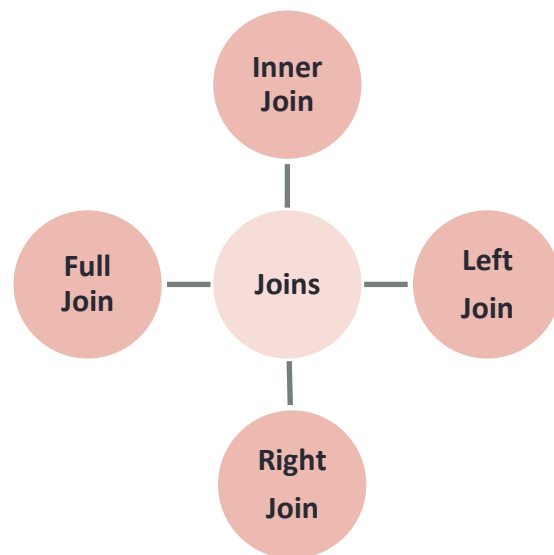
Explanation:

Storing the entire data into a single table leads to data redundancy and duplicate entries. Normalisation of data plays a vital role in analysing the databases, which ensures the data integrity and consistency. Joins optimise the query execution by combining the data from

multiples tables. By organizing the data into normalised tables, one can easily identify the relationships within the database which leads to more meaningful and accurate analysis.

Types of Joins:

There are several types of joins to retrieve data from multiple tables, the most commonly used joins are explained below.



1. Inner Join: Inner Joins are used to retrieve the data which is common between the tables.
2. Left Join: This type of join retrieves complete data from the left most table and the matching records from the right table.
3. Right Join: Gathers all records from the right table and matching records of the left join.
4. Full Join: Used to retrieve the complete data from both the tables.

TIPS FOR EFFECTIVE USE OF JOINS

- ❖ Become familiar with the database, fields, and the relationships among the tables.
- ❖ Understand about the output to be retrieved and identify the tables from which the data needs to be retrieved, this includes identifying the primary and foreign keys to perform the join operation.
- ❖ Know about the differences between joins to have a clear idea of what type of join will be appropriate to achieve the required output from the data base.
- ❖ Usage of ALIAS enhances the readability of the query when comes to dealing with multiple tables.

- ❖ Avoid writing the single query to join large number of tables, instead use simple and multiple queries to reduce the query complexity.

VIEWS IN SQL

What is a view?

A view is a stored SQL query; they are much similar to the tables but exist virtually. The key difference between the view and table is, views cannot store any data in physical form, and they can be invoked dynamically when required.

Why Views are used?

Writing complex queries frequently could become tiresome. Views can be used to store the complex queries. Also where there is a need to execute queries multiple times, they come into picture, saving time by eliminating the need to retype the queries.

Example:

ID	Name	Age	Marks
1	John	18	90
2	Anita	17	69
3	Michael	18	82

Retrieve the data of students whose marks > 80:

```
CREATE VIEW Marks AS
SELECT Name, Marks
FROM Student
WHERE Marks > 80;
```

WINDOW FUNCTIONS IN SQL

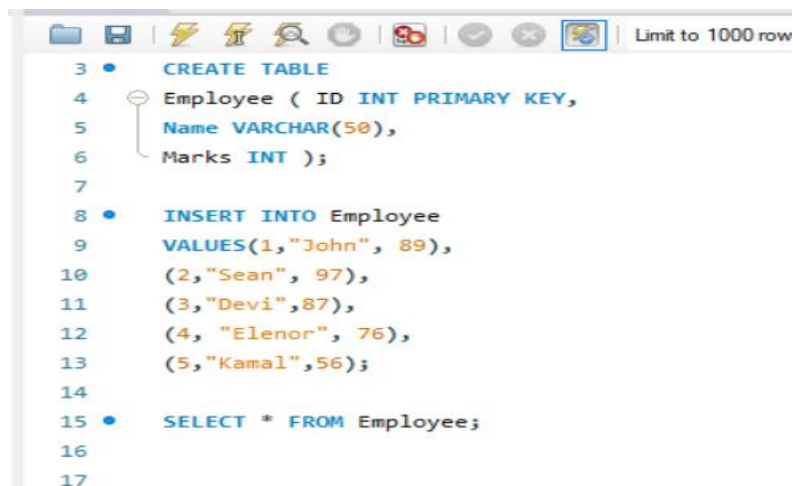
What is Windows function?

Window functions perform calculations similar to the Aggregate Functions, but in window functions each row retains its identity. That is they operate on each row independently, unlike aggregate functions. They can be used to compare one row with another.

EXAMPLE:

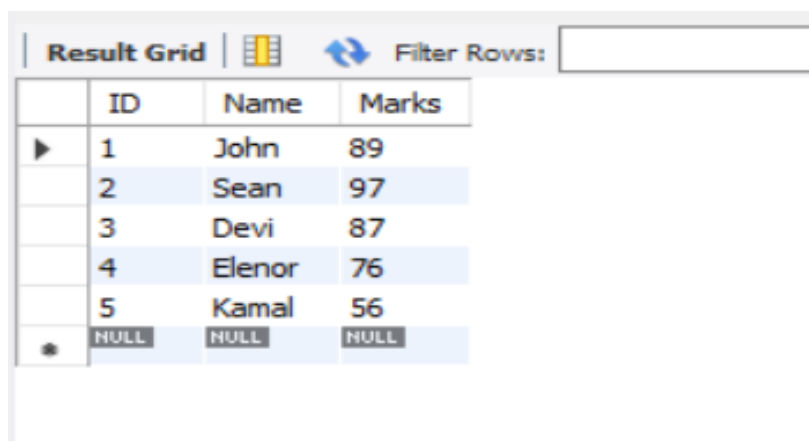
Let us understand the difference between the outputs of the queries given below, which helps to understand the concept of windows effectively.

WITHOUT WINDOW FUNCTIONS:



```
3 • CREATE TABLE
4   Employee ( ID INT PRIMARY KEY,
5   Name VARCHAR(50),
6   Marks INT );
7
8 • INSERT INTO Employee
9   VALUES(1,"John", 89),
10  (2,"Sean", 97),
11  (3,"Devi",87),
12  (4, "Elenor", 76),
13  (5,"Kamal",56);
14
15 • SELECT * FROM Employee;
16
17
```

OUTPUT:

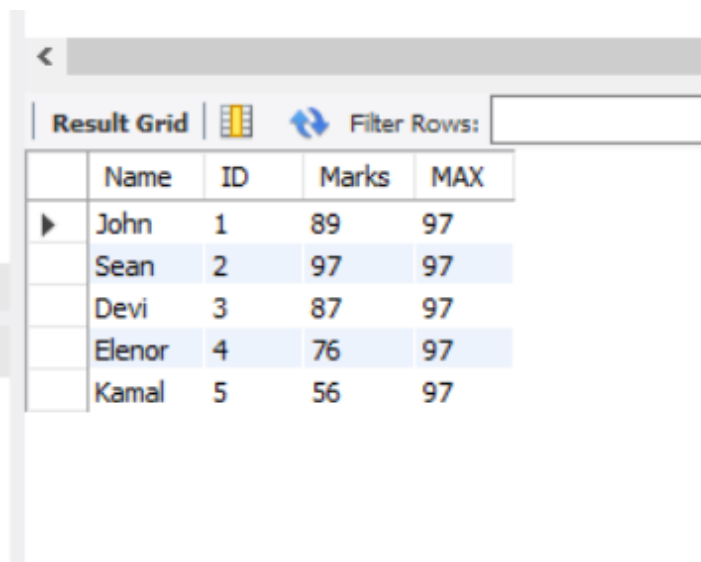


	ID	Name	Marks
▶	1	John	89
	2	Sean	97
	3	Devi	87
	4	Elenor	76
	5	Kamal	56
•	NULL	NULL	NULL

USING WINDOWS FUNCTION:

```
18
19
20 • SELECT Name, ID, Marks,
21      MAX( Marks) OVER() AS MAX
22      FROM Employee
23      GROUP BY ID;
```

OUTPUT:



The screenshot shows a database interface with a 'Result Grid' tab. It displays a table with 5 columns: Name, ID, Marks, and MAX. The data rows are as follows:

	Name	ID	Marks	MAX
▶	John	1	89	97
	Sean	2	97	97
	Devi	3	87	97
	Elenor	4	76	97
	Kamal	5	56	97

In this query, **MAX (Marks) OVER()** calculates the maximum marks over the entire table. Using window functions the Marks of each student can be compared with the maximum marks of total students.

TYPES OF WINDOW FUNCTIONS:

They are primarily classified into 3 types:

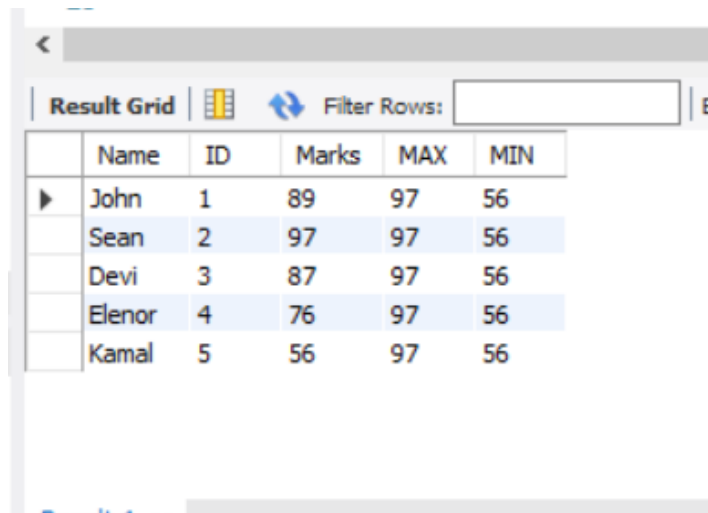
- Aggregate window functions : Performs aggregate operations such as SUM(), MIN(), MAX(), COUNT() ..etc
- Rank window functions : They are used to rank the rows, like RANK(), DENSE RANK() etc
- Value window functions: They allow to access the preceding and following rows, LEAD() LAG() etc.

OVER () CLAUSE : OVER () Clause is used to define the window.

Syntax:

```
SELECT Name, ID, Marks,  
Window_function OVER ()  
FROM Employee;
```

A function is converted to window functions by specifying OVER () clause. Specifying OVER () after the function opens a window. For example MAX () OVER (), MIN () OVER () etc.

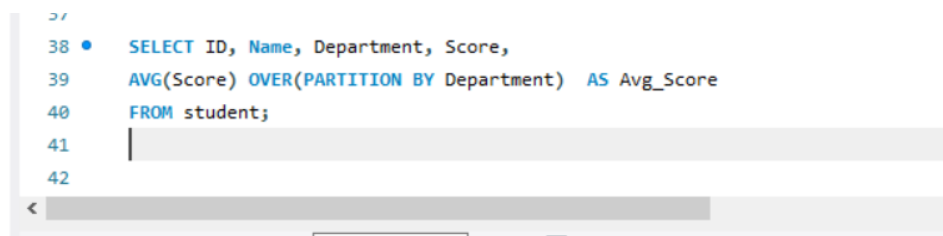


	Name	ID	Marks	MAX	MIN
▶	John	1	89	97	56
	Sean	2	97	97	56
	Devi	3	87	97	56
	Elenor	4	76	97	56
	Kamal	5	56	97	56

Over() clause can be used with parameters, PARTITION BY, ORDER BY, depends on the requirement.

PARTITION BY

PARTITION BY creates set of partitions by segmenting rows into groups mentioned in the OVER() clause.



```
38 • SELECT ID, Name, Department, Score,  
39 AVG(Score) OVER(PARTITION BY Department) AS Avg_Score  
40 FROM student;  
41  
42
```

OUTPUT: The query calculates the average score partitioned into departments .

Result Grid					
Filter Rows: <input type="text"/>					
	ID	Name	Department	Score	Avg_Score
▶	2	Venu	Maths	67	77.3333
	5	Michael	Maths	67	77.3333
	6	Albie	Maths	98	77.3333
	3	Devi	Microbiology	78	78.0000
	1	Sasha	Science	98	93.5000
	4	Kirti	Science	89	93.5000

ORDER BY:

ORDER BY within the OVER () Clause is used to order the rows within the window frame.

```

37
38 • SELECT ID, Name, Department, Score,
39     SUM(Score) OVER(ORDER BY ID) AS Running_Score
40 FROM student;
41
42

```

OUTPUT:

The query arranges the score in the ascending order and calculates the cumulative score.

Result Grid					
Filter Rows: <input type="text"/>					
	ID	Name	Department	Score	Running_Score
▶	1	Sasha	Science	98	98
	2	Venu	Maths	67	165
	3	Devi	Microbiology	78	243
	4	Kirti	Science	89	332
	5	Michael	Maths	67	399
	6	Albie	Maths	98	497

CONCLUSION:

Windows functions can be used when there is a requirement to compare the values, to calculate the cumulative sums or the average of values. They can be used to assign ranks .

References: <https://www.freecodecamp.org/news/window-functions-in-sql/>