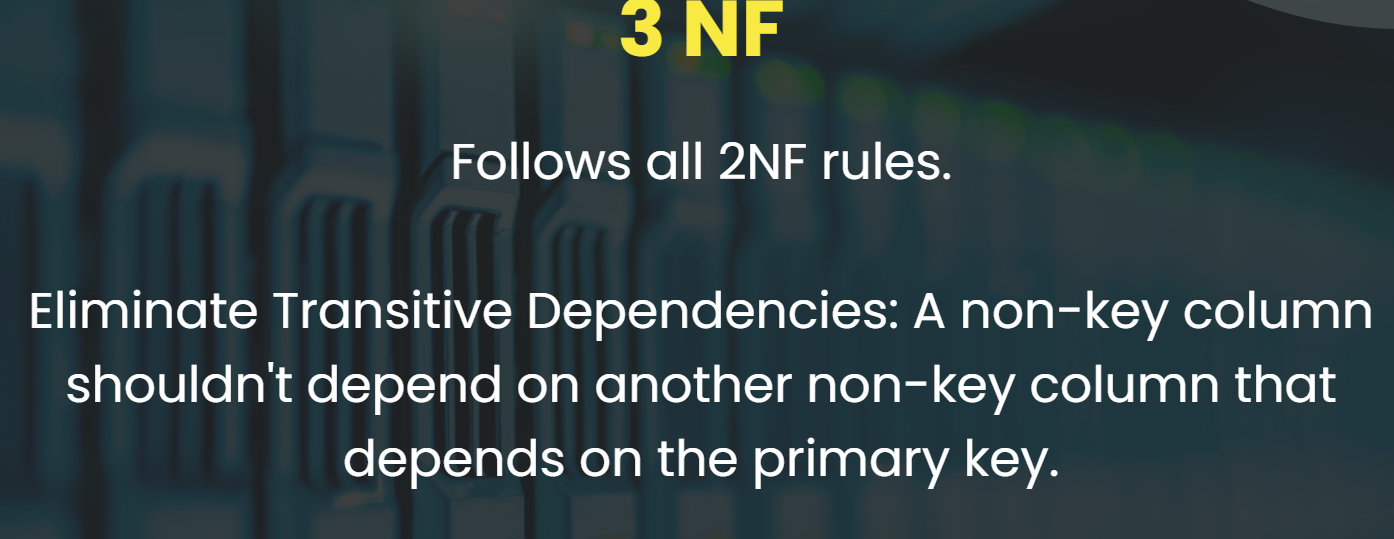
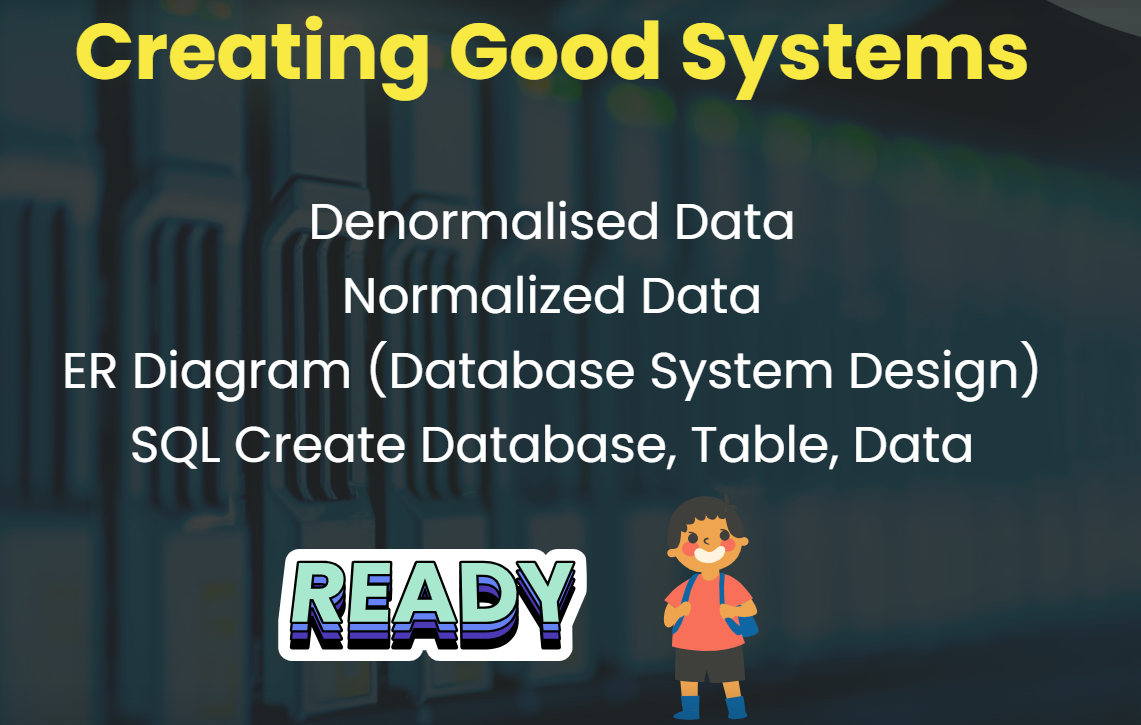


<https://docs.google.com/spreadsheets/d/1yuToth9W27z1GOFRSJpMNfbUMiOQKkWiaVPwFX-gWAQ/edit?gid=0#gid=0>









# **Window Function in SQL :**

### **What are Window Functions?**

* Window functions are powerful tools that allow you to perform calculations on a set of rows within a result set, defined by a window.
* They differ from aggregate functions (SUM, COUNT, etc.) in that they return a value for each row, not just a single value for the entire group.

### **Syntax of Window Functions:**

SELECT column\_name, window\_function(expression) OVER (window\_definition)

FROM table\_name;

* column\_name: The column you want to apply the window function to.
* window\_function: The specific window function you want to use (SUM, COUNT, ROW\_NUMBER, etc.).
* expression: The expression to be evaluated within the window.
* window\_definition: (Optional) Defines the window frame (PARTITION BY and ORDER BY clauses) to specify which rows are considered for the calculation.

### **Window Functions with Aggregate Functions (SUM, COUNT, MIN, MAX, AVG):**

These functions operate on a group of rows within the window frame, returning a value for each row based on the specified aggregation:

* SUM(expression): Calculates the running total of the expression within the window.
* COUNT(expression): Counts the number of rows within the window (excluding NULLs by default).
* MIN(expression): Returns the minimum value of the expression within the window.
* MAX(expression): Returns the maximum value of the expression within the window.
* AVG(expression): Calculates the running average of the expression within the window.

### **Window Functions with Ranking Functions (ROW\_NUMBER, RANK, DENSE\_RANK):**

These functions assign a rank or order to each row within the window:

* ROW\_NUMBER(): Assigns a unique sequential number to each row within the window, regardless of duplicates.
* RANK(): Assigns a rank to each row within the window, but skips ranks for ties (e.g., two rows with the same value might get the same rank).
* DENSE\_RANK(): Assigns a rank to each row within the window, but unlike RANK, it does not skip ranks for ties (e.g., consecutive rows with the same value will have the same rank).

### **Window Functions with Value/Analytic Functions (FIRST\_VALUE, LAST\_VALUE, LEAD, LAG):**

* FIRST\_VALUE(expression): Returns the first value of the expression encountered within the window frame.
* LAST\_VALUE(expression): Returns the last value of the expression encountered within the window frame.
* LEAD(expression, offset): Returns the value of the expression offset rows ahead in the window frame (offset can be negative).
* LAG(expression, offset): Returns the value of the expression offset rows behind in the window frame (offset can be negative).

### **Example :**

#### **calculates the average salary within each department while still showing the individual employee records**

SELECT

id,

name,

email,

gender,

salary,

department,

AVG(salary) OVER(PARTITION BY department) AS AvgDepartmentSalary

FROM

emp;

#### **This example finds the minimum salary within each department**

SELECT

id,

name,

email,

gender,

salary,

department,

MIN(salary) OVER(PARTITION BY department) AS MinDepartmentSalary

FROM

emp;

#### **This example calculates the total salary expenditure for each department:**

SELECT

id,

name,

email,

gender,

salary,

department,

SUM(salary) OVER(PARTITION BY department) AS TotalDepartmentSalary

FROM

emp;

#### **Assign a row number to each employee in each department, ordered by their salary in descending order (highest salary gets the row number 1):**

SELECT

id,

name,

email,

gender,

salary,

department,

ROW\_NUMBER() OVER(PARTITION BY department ORDER BY salary DESC) AS SalaryRank

FROM

emp;

#### **Rank employees in each department based on their salary, with ties resulting in the same rank and a gap for the subsequent ranks:**

SELECT

id,

name,

email,

gender,

salary,

department,

RANK() OVER(PARTITION BY department ORDER BY salary DESC) AS SalaryRank

FROM

emp;

#### **Dense rank employees in each department based on their salary, with ties resulting in the same rank but no gaps for subsequent ranks:**

SELECT

id,

name,

email,

gender,

salary,

department,

DENSE\_RANK() OVER(PARTITION BY department ORDER BY salary DESC) AS SalaryRank

FROM

emp;

#### **Get the highest salary in each department alongside each employee's salary :**

SELECT

id,

name,

email,

gender,

salary,

department,

FIRST\_VALUE(salary) OVER(PARTITION BY department ORDER BY salary DESC) AS HighestSalary

FROM

emp;

#### **Display each employee's salary and the salary of the employee who follows them in the same department**

SELECT

id,

name,

email,

gender,

salary,

department,

LEAD(salary) OVER(PARTITION BY department ORDER BY salary) AS NextSalary

FROM

emp;

# **Normalization in Database**

### **Description :**

* Normalization is a systematic approach to organizing data in a relational database to minimize redundancy (data duplication) and improve data integrity, efficiency, and maintainability.
* It involves structuring data into tables with well-defined relationships, adhering to a set of rules called normal forms.
* This process involves dividing large tables into smaller, interrelated tables and defining relationships between them to minimize redundancy and improve data integrity.

### **Normalized vs. Denormalized Data**

#### **Normalised Data**

* Follows normalization principles to minimize redundancy.
* Ensures data consistency and reduces storage space.
* Requires more complex queries when retrieving related data across tables.

#### **Denormalised Data**

* Introduced repetition in some cases to optimize query performance for specific use cases.
* Denormalization is sometimes used deliberately for performance reasons, such as reducing the number of joins needed to query a database, which can speed up read operations at the cost of making write operations more complex and potentially slower.
* May be more efficient for frequent reads but requires careful management to avoid inconsistencies.

### **Levels of Normalization**

Normalization is categorized into several normal forms, each with its own set of rules. The most commonly discussed are:

* First Normal Form (1NF)
* Second Normal Form (2NF)
* Third Normal Form (3NF)

### **1NF Level**

* Atomic Values: Each cell should contain a single, indivisible value.
* Primary Key: Every table must have a primary key that uniquely identifies each row. It can be a single column or a combination of columns.

### **2NF Level**

* Follows all 1NF rules.
* Eliminate Partial Dependencies: A non-key column should depend only on the entire primary key, not on a part of it.
* Relationship between tables should be stablished by using Foreign Keys.

### **3NF Level**

* Follows all 2NF rules.
* Eliminate Transitive Dependencies: A non-key column shouldn't depend on another non-key column that depends on the primary key.

### **Database Anomalies**

* Insertion Anomaly : Occurs when certain attributes cannot be inserted into the database without the presence of other attributes. Normalization allows for the independent insertion of data.
* Deletion Anomaly : Happens when the deletion of data about one entity inadvertently leads to the loss of data about another entity. Proper normalization ensures that deletions affect only the intended data.
* Update Anomaly: Arises when multiple instances of the same data are kept across the database; updating some but not all instances leads to inconsistent data. Normalization, by storing each piece of information only once, helps prevent such inconsistencies.

## **Resources - Official Documentation and Other Resources**

* Window Function - <https://mode.com/sql-tutorial/sql-window-functions>
* Database Normalization - <https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description>
* Database Normalization 2 - <https://www.simplilearn.com/tutorials/sql-tutorial/what-is-normalization-in-sql>

Window Function - 1

-- Switch to the desired database

USE db101;

-- Show all tables in the database

SHOW TABLES;

-- Describe the 'emp' table structure

DESCRIBE emp;

-- Select all data from the 'emp' table

SELECT \* FROM emp;

-- Group data by department and calculate the count of employees in each department

SELECT department, COUNT(\*) AS emp\_count

FROM emp

GROUP BY department;

-- Group data by department and calculate the average salary in each department

SELECT department, AVG(salary) AS avg\_salary

FROM emp

GROUP BY department;

-- Count the total number of employees

SELECT COUNT(id) FROM emp;

-- Group data by department and calculate the count of employees along with their details

SELECT department, COUNT(\*) AS emp\_count, name, id, salary

FROM emp

GROUP BY department, name, id, salary;

-- Window functions are useful when you need to retain the individual row details while also performing some calculation over a set of rows related to the current row

-- Select all data from the 'emp' table

SELECT \* FROM emp;

-- For each employee, give all details along with the total number of employees

SELECT id, name, department, COUNT(id) OVER() AS total\_emp

FROM emp;

-- For each employee, give all details along with the minimum salary in the whole organization

SELECT id, name, department, MIN(salary) OVER() AS min\_salary

FROM emp;

-- For each employee, give all details along with the department's average salary

SELECT id, name, department, AVG(salary) OVER(PARTITION BY department) AS dept\_avg\_salary

FROM emp;

-- For each employee, give all details along with the department's average salary, ordered by salary descending

SELECT id, name, department, AVG(salary) OVER(PARTITION BY department ORDER BY salary DESC) AS dept\_avg\_salary

FROM emp;

-- For each employee, give all details along with the department's average salary, ordered by salary descending, and then order the results by salary

SELECT id, name, department, AVG(salary) OVER(PARTITION BY department ORDER BY salary DESC) AS dept\_avg\_salary

FROM emp

ORDER BY salary;

-- Now let's understand the syntax again from the slides

-- We have learned about using window functions with aggregate methods

-- Now we will learn using window functions with ranking functions

Window Function - 2

-- Switch to the desired database

USE db101;

-- Select all data from the 'emp' table

SELECT \* FROM emp;

-- Create a new column called row\_num and add the row number there

SELECT \*, ROW\_NUMBER() OVER (ORDER BY name) AS row\_num

FROM emp;

-- Select only even rows

WITH ranking AS (

SELECT \*, ROW\_NUMBER() OVER (ORDER BY name) AS row\_num

FROM emp

)

SELECT \* FROM ranking

WHERE ranking.row\_num % 2 = 0;

-- Find the row number based on each department

SELECT \*, RANK() OVER (ORDER BY salary DESC) AS row\_num

FROM emp;

-- Sort employees based on salary for each department along with calculating rank

SELECT \*, DENSE\_RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS curr\_rank

FROM emp;

-- Sort employees based on salary for each department along with rank and calculating row number

SELECT \*,

DENSE\_RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS curr\_rank,

ROW\_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC) AS row\_num

FROM emp;

-- Top 3 ranking employees for each department

WITH ranking AS (

SELECT \*, DENSE\_RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS curr\_rank

FROM emp

)

SELECT \* FROM ranking

WHERE curr\_rank <= 3;

-- Create a new column called salary\_rank and give rank of employees based on salary

SELECT \*, RANK() OVER (ORDER BY salary DESC) AS salary\_rank

FROM emp;

-- Give rank for each department based on salary

SELECT \*, RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS dept\_rank

FROM emp;

-- Instead of RANK() we can also use DENSE\_RANK()

-- Let's understand the difference

-- Using RANK()

SELECT \*, RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS dept\_rank

FROM emp;

-- Using DENSE\_RANK()

SELECT \*, DENSE\_RANK() OVER (PARTITION BY department ORDER BY salary DESC) AS dept\_dense\_rank

FROM emp;

-- Explanation:

-- RANK() will assign the same rank to employees with the same salary, but will skip subsequent ranks.

-- For example, if two employees have the highest salary, they will both be ranked 1, and the next employee will be ranked 3.

-- DENSE\_RANK() also assigns the same rank to employees with the same salary, but does not skip ranks.

-- For example, if two employees have the highest salary, they will both be ranked 1, and the next employee will be ranked 2.

Window Function - 3 -

-- USE database

USE b35\_db101;

-- Select all the data

SELECT \* FROM emp;

-- Add the first emp salary in row

SELECT \* ,

(FIRST\_VALUE(salary) OVER()) AS first\_emp\_salary

FROM emp;

-- Add the last emp salary in each row

SELECT \* ,

(LAST\_VALUE(salary) OVER()) AS last\_emp\_salary

FROM emp;

-- For each department who is the first emp

SELECT \* ,

(FIRST\_VALUE(salary) OVER(PARTITION BY department)) AS first\_emp\_salary

FROM emp;

SELECT \* ,

(FIRST\_VALUE(salary) OVER(PARTITION BY department ORDER BY salary)) AS first\_emp\_salary

FROM emp;

SELECT \* ,

(FIRST\_VALUE(salary) OVER(PARTITION BY department ORDER BY salary DESC)) AS first\_emp\_salary

FROM emp;

-- FETCH THE NEXT PERSON SALARY IN EACH DEPARTMENT

SELECT \* ,

(LEAD(salary) OVER(PARTITION BY department ORDER BY SALARY DESC)) AS next\_emp\_salary

FROM emp;

-- FETCH THE Previous PERSON SALARY IN EACH DEPARTMENT

SELECT \* ,

(LAG(salary) OVER(PARTITION BY department ORDER BY SALARY DESC)) AS next\_emp\_salary

FROM emp;

* Database Design Assignment Help - [Link](https://oxidized-sphynx-2a8.notion.site/SQL-Database-design-W2S2-cde58d318ec34e089ac1234ce67a9f40?pvs=4)
* Inventory Management ER Diagram: [Link](https://drawsql.app/teams/zakir-1/diagrams/inventory-management)