#### ON vs USING in JOINs - Full Notes with Examples

# Purpose:

Both ON and USING are used to **specify join conditions**, but they differ in syntax, flexibility, and behavior.

- 1. ON Clause Flexible, Standard
- Syntax:

sql

CopyEdit

SELECT ...

FROM table1

JOIN table2

ON table1.colA = table2.colB;

- Features:
  - Works with any column names (even if different)
  - Can use **complex conditions** (e.g., >, <, !=, AND)
  - Must specify both sides of the comparison

## Example:

sql

CopyEdit

SELECT s.Name, d.DeptName

FROM Students s

JOIN Departments d

ON s.DeptID = d.DeptID;

- **2. USING Clause Simpler, Shorter**
- Syntax:

sql

CopyEdit

SELECT ...

FROM table1

JOIN table2

USING (common\_column);

## **✓** Features:

- Works only if both tables have a column with the same name
- Automatically assumes table 1.col = table 2.col
- Simplifies code, but less flexible

## Example:

If both Students and Departments have DeptID:

sql

CopyEdit

SELECT Name, DeptName

**FROM Students** 

JOIN Departments

USING (DeptID);

# 兿 ON vs USING – Quick Comparison

| Feature      | ON Clause                              | USING Clause                 |
|--------------|--|------------------------------|
| Column names | Can be different                       | Must be same in both tables  |
| Syntax       | Verbose (more typing)                  | Shorter, cleaner             |
| Flexibility  | ✓ Complex conditions allowed           | X Only equals (=) comparison |
| Readability  | Preferred for clarity in large queries | Simpler but limited          |

| Feature              | ON Clause                        | USING Clause            |
|----------------------|----------------------------------|-------------------------|
| NULL handling        | Same behavior                    | Same (both use = logic) |
| NATURAL JOIN<br>link | Similar to USING for all columns | Only for listed column  |

#### **NATURAL JOIN - Full Notes with Examples**



A NATURAL JOIN is a type of join that:

✓ Automatically joins tables based on columns with the same names and compatible data types in both tables.

No need to write ON or USING. The database engine detects the columns automatically.

## Syntax:

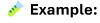
sql

CopyEdit

SELECT \*

FROM table1

NATURAL JOIN table2;



#### **Table 1: Students**

## **StudentID Name DeptID**

1 Riya 101

2 Aman 102

3 Sneha NULL

#### **Table 2: Departments**

## **DeptID DeptName**

101 Computer

102 Mechanical

Common column: DeptID

## **Query:**

sql

CopyEdit

SELECT \*

**FROM Students** 

NATURAL JOIN Departments;

## **Output:**

## StudentID Name DeptID DeptName

1 Riya 101 Computer

2 Aman 102 Mechanical

X Sneha is not included → DeptID is NULL → no match (just like INNER JOIN behavior)

#### ▲ Behavior Details:

- Auto-detects common columns
- Uses = as condition (like USING)
- Cannot use with different column names
- Joins are always **INNER JOINs** by default

#### Points to Remember:

#### **Feature**

#### **NATURAL JOIN Behavior**

Based on common column(s) <a>Ves</a>, must match in name & type

Join condition specified? X No ON/USING written

Match logic Uses equality (=) only

NULL match X NULL ≠ NULL → row excluded

Flexibility X Less flexible than ON or USING

Aliases with NATURAL JOIN 1. Tricky — avoid column ambiguity

#### **NULL Behavior in JOINs - What Happens & Why It Matters**

## Why This Is Important:

In SQL, NULL is **not equal to anything**, **not even another NULL**.

So during JOIN operations, **NULLs often cause rows to be excluded** or displayed with NULL values.

# Behavior with Different JOIN Types:

JOIN Type NULL in Join Column What Happens

**INNER JOIN** X Row is excluded No match, so row is skipped

**LEFT JOIN** ✓ Row included Right side = NULL

**RIGHT JOIN** Row included Left side = NULL

**FULL JOIN** ✓ Row included Missing side = NULL

NATURAL JOIN X Row is excluded Behaves like INNER JOIN



Table A: Employees

#### **EmpID Name DeptID**

- 1 Riya 101
- 2 Aman NULL

#### **Table B: Departments**

## **DeptID DeptName**

101 Computer

#### **✓** INNER JOIN:

sql

CopyEdit

SELECT e.Name, d.DeptName

FROM Employees e

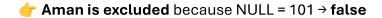
INNER JOIN Departments d

ON e.DeptID = d.DeptID;

## **Output:**

#### Name DeptName

Riya Computer



#### LEFT JOIN:

sql

CopyEdit

SELECT e.Name, d.DeptName

FROM Employees e

LEFT JOIN Departments d

ON e.DeptID = d.DeptID;

#### Output:

# Name DeptName Riya Computer Aman NULL

👉 All employees shown, even if DeptID is NULL

## Important Rule:

## Any comparison with NULL (even NULL = NULL) results in FALSE

This is why:

- INNER JOIN ignores such rows
- LEFT/RIGHT/FULL JOINs preserve them (with NULL on the unmatched side)

#### Multi-Table JOINs (3 or More Tables) - Full Notes with Examples



A **Multi-table JOIN** is when **three or more tables** are joined together in a single query to fetch related data from all of them.

# Syntax:

sql

CopyEdit

SELECT columns

FROM Table1

JOIN Table 2 ON Table 1.col = Table 2.col

JOIN Table3 ON Table2.col = Table3.col

... and so on;

# Real Example:

🧱 Tables:

#### **Students**

## **StudentID Name DeptID**

1 Riya 101

2 Aman 102

## **Departments**

## **DeptID DeptName CityID**

101 CS 1

102 Mech 2

#### Cities

## CityID CityName

1 Pune

2 Mumbai

## **Query:**

sql

CopyEdit

SELECT s.Name AS Student, d.DeptName, c.CityName

FROM Students s

JOIN Departments d ON s.DeptID = d.DeptID

JOIN Cities c ON d.CityID = c.CityID;

# **Output:**

## **Student DeptName CityName**

Riya CS Pune

Aman Mech Mumbai

# **Q** Explanation:

- First: Join Students with Departments via DeptID
- Second: Join result with Cities via CityID

# ★ Points to Remember (for Exam):

| Rule / Concept                | Meaning  |
|-------------------------------|--|
| Join order                    | JOINs are executed <b>left to right</b>              |
| Must join table by table      | One table at a time using proper ON clauses          |
| ✓ Use aliases                 | Use s, d, c to avoid confusion                       |
| ✓ NULLs matter                | Same as in 2-table JOINs — INNER JOIN skips<br>NULLs |
| ✓ Join logic should be valid  | Foreign key → Primary key chain must be respected    |
| Mix JOIN types (LEFT + INNER) | You can, but be careful with logic                   |