🔍 Definition

A SQL JOIN is used to combine rows from two or more tables based on a common column between them.

It retrieves data that is related logically between multiple tables.

🧩 Basic Syntax

SELECT columns\_from\_both\_tables

FROM table1

JOIN table2

ON table1.column1 = table2.column2;

Explanation:

table1 and table2 → Tables you want to join

column1 and column2 → Common columns between the two tables

The ON keyword specifies the matching condition

Example 1 — Join Two Tables Based on a Common Column

Tables:

Customers Table

customer\_id first\_name

1 John

2 Mary

3 Bob

Orders Table

order\_id customer item amount

101 1 Laptop 1200

102 2 Phone 700

103 3 Tablet 900

Query

SELECT Customers.customer\_id, Customers.first\_name, Orders.item

FROM Customers

JOIN Orders

ON Customers.customer\_id = Orders.customer;

✅ Explanation:

This joins the Customers and Orders tables using the customer\_id column.

🧾 Result:

customer\_id first\_name item

1 John Laptop

2 Mary Phone

3 Bob Tablet

Example 2 — Join With WHERE Clause

SELECT Customers.customer\_id, Customers.first\_name, Orders.amount

FROM Customers

JOIN Orders

ON Customers.customer\_id = Orders.customer

WHERE Orders.amount >= 800;

✅ Explanation:

This returns only those customers whose order amount ≥ 800.

🧾 Result:

customer\_id first\_name amount

1 John 1200

3 Bob 900

JOIN Multiple Tables

You can join three or more tables by chaining JOIN operations.

Example 3 — Joining Three Tables

Let’s add a Shippings table.

shipping\_id customer status

1 1 Delivered

2 2 Pending

3 3 Shipped

SELECT Customers.first\_name, Orders.item, Shippings.status

FROM Customers

JOIN Orders ON Customers.customer\_id = Orders.customer

JOIN Shippings ON Customers.customer\_id = Shippings.customer;

🧾 Result:

first\_name item status

John Laptop Delivered

Mary Phone Pending

Bob Tablet Shipped

✅ Explanation:

This joins three tables based on matching customer\_id (or customer) fields and retrieves columns from all of them.

🧱 Types of SQL JOINs

Type Description

INNER JOIN Returns only rows with matching values in both tables

LEFT JOIN Returns all rows from the left table, and matching rows from the right table (NULL where no match)

RIGHT JOIN Returns all rows from the right table, and matching rows from the left table

FULL OUTER JOIN Returns all rows from both tables (NULL where no match on either side)

Example 4 — INNER JOIN

SELECT Customers.first\_name, Orders.item

FROM Customers

INNER JOIN Orders

ON Customers.customer\_id = Orders.customer;

🧾 Result: Only customers who have placed orders.

Example 5 — LEFT JOIN

SELECT Customers.first\_name, Orders.item

FROM Customers

LEFT JOIN Orders

ON Customers.customer\_id = Orders.customer;

🧾 Result: All customers are shown, even if they have no orders (those will show NULL for item).

Example 6 — RIGHT JOIN

SELECT Customers.first\_name, Orders.item

FROM Customers

RIGHT JOIN Orders

ON Customers.customer\_id = Orders.customer;

🧾 Result: All orders are shown, even if no matching customer record exists.

Example 7 — FULL OUTER JOIN

(Some databases like MySQL need UNION instead)

SELECT Customers.first\_name, Orders.item

FROM Customers

LEFT JOIN Orders ON Customers.customer\_id = Orders.customer

UNION

SELECT Customers.first\_name, Orders.item

FROM Customers

RIGHT JOIN Orders ON Customers.customer\_id = Orders.customer;

🧾 Result: Returns all customers and all orders — matched or unmatched.

SQL SELF JOIN

A Self Join joins a table to itself — useful for comparing rows within the same table.

Example 8 — Self Join

SELECT

C1.first\_name AS FirstPerson,

C2.first\_name AS SecondPerson,

C1.country

FROM Customers C1, Customers C2

WHERE C1.country = C2.country

AND C1.first\_name != C2.first\_name;

✅ Explanation:

This finds pairs of customers from the same country, but with different names.

🧾 Result:

FirstPerson SecondPerson country

John Mary USA

Mary John USA

SQL JOIN With Table Aliases

You can simplify your queries using AS aliases.

Example 9 — Table Alias

SELECT C.customer\_id, C.first\_name, O.amount

FROM Customers AS C

JOIN Orders AS O

ON C.customer\_id = O.customer;

✅ Explanation:

Aliases (C and O) make the query shorter and more readable.

SQL JOIN With Column Aliases

You can also rename column headers temporarily in the result set.

Example 10 — Column Alias

SELECT

C.customer\_id AS cid,

C.first\_name AS name,

O.amount AS total\_amount

FROM Customers AS C

JOIN Orders AS O

ON C.customer\_id = O.customer;

🧾 Result:

cid name total\_amount

1 John 1200

2 Mary 700

3 Bob 900

JOIN With WHERE Clause (Filtering Results)

Example 11 — JOIN + WHERE

SELECT C.customer\_id, C.first\_name, O.amount

FROM Customers AS C

JOIN Orders AS O

ON C.customer\_id = O.customer

WHERE O.amount >= 500;

✅ Explanation:

Returns customers with order amount ≥ 500 after joining.

✅ Summary Table

Join Type Returns

INNER JOIN Only matching rows

LEFT JOIN All left + matching right rows

RIGHT JOIN All right + matching left rows

FULL OUTER JOIN All rows from both tables

SELF JOIN Table joined to itself

JOIN with Alias Simplifies table/column references

JOIN with WHERE Adds conditional filtering  
🧩 Employee Database Schema

Employees

Column Type Description

EmpID INT (PK) Employee ID

FirstName VARCHAR Employee’s first name

LastName VARCHAR Employee’s last name

ManagerID INT (FK) References EmpID (self-join for manager hierarchy)

DepartmentID INT Foreign key to Departments

Salary DECIMAL Employee’s salary

Departments

Column Type Description

DepartmentID INT (PK) Department ID

DepartmentName VARCHAR Name of the department

✅ 1. Retrieve Each Employee’s Full Name and Their Manager’s Full Name

This is a self-join query because employees and their managers are stored in the same table.

Query

SELECT

E.EmpID,

CONCAT(E.FirstName, ' ', E.LastName) AS EmployeeName,

CONCAT(M.FirstName, ' ', M.LastName) AS ManagerName

FROM Employees AS E

LEFT JOIN Employees AS M

ON E.ManagerID = M.EmpID;

✅ Explanation:

E = employee

M = manager

A LEFT JOIN ensures all employees are included even if they have no manager (e.g., CEO).

✅ 2. Find Employees Who Work in the ‘Sales’ Department

We will write two equivalent queries — both yield the same result but differ in approach.

(a) Using INNER JOIN

SELECT

E.EmpID,

CONCAT(E.FirstName, ' ', E.LastName) AS EmployeeName,

D.DepartmentName

FROM Employees AS E

INNER JOIN Departments AS D

ON E.DepartmentID = D.DepartmentID

WHERE D.DepartmentName = 'Sales';

(b) Using Subquery in WHERE Clause

SELECT

EmpID,

CONCAT(FirstName, ' ', LastName) AS EmployeeName

FROM Employees

WHERE DepartmentID IN (

SELECT DepartmentID

FROM Departments

WHERE DepartmentName = 'Sales'

);

✅ Equivalence Explanation:

Both queries produce the same result set.

Query (a) uses explicit join (faster in most optimizers).

Query (b) uses subquery filtering (logically equivalent but sometimes less efficient).

✅ 3. Recursive SQL Query to Find Management Chain

This retrieves all managers up the hierarchy for a given employee.

Recursive CTE Query

WITH RECURSIVE ManagementChain AS (

-- Base case: start with the given employee

SELECT

EmpID,

FirstName,

LastName,

ManagerID

FROM Employees

WHERE EmpID = 105 -- Example: given employee ID

UNION ALL

-- Recursive case: find the manager of the current employee

SELECT

E.EmpID,

E.FirstName,

E.LastName,

E.ManagerID

FROM Employees E

INNER JOIN ManagementChain MC

ON E.EmpID = MC.ManagerID

)

SELECT \* FROM ManagementChain;

✅ Explanation:

The base case selects the starting employee.

The recursive part repeatedly finds that employee’s manager, their manager’s manager, etc.

It terminates when a manager has no ManagerID.

✅ 4. Optimized Recursive Query and Heuristics

Recursive queries can be expensive due to multiple self-joins.

Let’s rewrite and optimize the query for performance.

Optimized Recursive Query

WITH RECURSIVE ManagementChain AS (

SELECT

EmpID,

FirstName,

LastName,

ManagerID,

1 AS Level

FROM Employees

WHERE EmpID = 105

UNION ALL

SELECT

M.EmpID,

M.FirstName,

M.LastName,

M.ManagerID,

MC.Level + 1 AS Level

FROM Employees M

INNER JOIN ManagementChain MC

ON M.EmpID = MC.ManagerID

)

SELECT EmpID, CONCAT(FirstName, ' ', LastName) AS ManagerName, Level

FROM ManagementChain

ORDER BY Level;

🔧 Optimization Heuristics Used

Heuristic Description

H1: Selection Pushdown Filter early (WHERE EmpID = 105) before recursion begins. Reduces initial data.

H2: Projection Pushdown Select only necessary columns (EmpID, ManagerID, names).

H3: Avoid Redundant Joins Join only once per recursion level.

H4: Limit Recursion Depth Add a stopping condition if organization hierarchy depth is known.

H5: Indexing Create an index on ManagerID to speed up recursive lookup.

✅ 5. Query Equivalence Testing and Optimization Heuristics

(a) Testing Query Equivalence

Two SQL queries Q1 and Q2 are equivalent if:

They produce the same result set for all possible valid database states.

Steps to Test:

Run both queries on the same data.

Compare outputs — number of rows and content.

Verify schema (column names and data types).

Use EXCEPT / MINUS operator:

(Q1)

EXCEPT

(Q2);

If this returns no rows, the queries are equivalent.

(b) Heuristics for Optimizing Complex Queries

Heuristic Description / Example

1. Selection Pushdown Apply WHERE conditions as early as possible in the query tree.

Example: Filter employees by department before joining.

2. Projection Pushdown Retrieve only required columns.

3. Join Ordering Join smaller tables first or those with selective conditions.

4. Combine Selections Merge multiple filters into one where possible.

5. Avoid Redundant Subqueries Replace nested subqueries with joins if faster.

6. Use Indexes On join keys (DepartmentID, ManagerID).

7. Materialize Intermediate Results Cache recursive or repeated results to avoid recomputation.

8. Use Recursive Depth Limits Control recursion to prevent performance issues in hierarchies.

9. Decompose Complex Joins Break large joins into smaller manageable ones.

10. Query Rewriting / Equivalence Rules Example: σ(condition)(R ⨝ S) ≡ R ⨝ σ(condition)(S) when condition involves only S.

🧠 Summary

Task Concept Key SQL Feature

1 Employee–Manager relationship Self-Join

2 Department filtering JOIN vs Subquery Equivalence

3 Management chain hierarchy Recursive CTE

4 Query optimization Heuristics & Indexing

5 Equivalence testing EXCEPT operator & Result comparison