# 🎯 SQL JOINS – Complete Concept (PostgreSQL Version)

## 🔷 1. Introduction

👉 Definition: A JOIN is used to combine rows from two or more tables based on a related column between them.

👉 Real-time Use: In companies, data is usually stored in multiple tables (for normalization).

For example:

Employee details in one table (employees)

Department details in another (departments) To generate reports, we need to fetch combined data — that's where JOINS are used.

## 2. Common Tables for Example

```
CREATE TABLE departments (
    dept_id SERIAL PRIMARY KEY,
    dept_name VARCHAR(50)
);

CREATE TABLE employees (
    emp_id SERIAL PRIMARY KEY,
    emp_name VARCHAR(50),
    salary NUMERIC(10,2),
    dept_id INT REFERENCES departments(dept_id)
);

INSERT INTO departments (dept_name)
VALUES ('HR'), ('Finance'), ('IT'), ('Marketing');
```

```
INSERT INTO employees (emp_name, salary, dept_id)
VALUES
('Ravi', 50000, 1),
('Kiran', 60000, 2),
('Sita', 55000, 3),
('Lalitha', 45000, NULL);
```

◈ 3. Types of Joins

(A) INNER JOIN

☞ Definition:

Fetches only matching rows from both tables.

☞ Syntax:

```
SELECT e.emp_name, d.dept_name
FROM employees e
INNER JOIN departments d
ON e.dept_id = d.dept_id;
```

☞ Output:

Only employees with valid department IDs will be shown.

☞ Real-time Use:

When we want only active employees with a valid department.

(B) LEFT JOIN (LEFT OUTER JOIN)

☞ Definition:

Fetches all records from the left table (employees) and matching ones from the right (departments).

👉 Syntax:

```
SELECT e.emp_name, d.dept_name
FROM employees e
LEFT JOIN departments d
ON e.dept_id = d.dept_id;
```

👉 Output:

Shows all employees — even those without a department (NULL dept).

👉 Real-time Use:

When company wants to see which employees are not assigned to any department.

(C) RIGHT JOIN (RIGHT OUTER JOIN)

👉 Definition:

Fetches all records from the right table (departments) and matching ones from the left (employees).

👉 Syntax:

```
SELECT e.emp_name, d.dept_name
FROM employees e
RIGHT JOIN departments d
ON e.dept_id = d.dept_id;
```

👉 Output:

All departments shown — even those with no employees.

👉 Real-time Use:

When admin wants to see which departments currently don’t have employees.

(D) FULL OUTER JOIN

👉 Definition:

Fetches all records when there is a match in either left or right table.

👉 Syntax:

```
SELECT e.emp_name, d.dept_name

FROM employees e

FULL OUTER JOIN departments d

ON e.dept_id = d.dept_id;
```

👉 Output:

All employees + all departments — even unmatched ones.

👉 Real-time Use:

To get a complete overview of data mismatches.

(E) CROSS JOIN

👉 Definition:

Produces Cartesian Product — every row of one table joins with every row of the other.

☞ Syntax:

```
SELECT e.emp_name, d.dept_name
FROM employees e
CROSS JOIN departments d;
```

☞ Output:

If 4 employees and 4 departments → 16 rows.

☞ Real-time Use:

Used rarely — for generating combinations (ex: time slots, product variants, etc.).

(F) SELF JOIN

☞ Definition:

A table joins with itself (commonly used for hierarchical data).

☞ Syntax:

```
SELECT a.emp_name AS Employee, b.emp_name AS Manager
FROM employees a
JOIN employees b
ON a.manager_id = b.emp_id;
```

☞ Real-time Use:

Employee–Manager relationships, reporting hierarchies, etc.

## 📓 4. Important Notes

| **Concept** | **Explanation** |
|---|---|
| **INNER JOIN** | Only matching records |
| **LEFT JOIN** | All from left + matched from right |
| **RIGHT JOIN** | All from right + matched from left |
| **FULL JOIN** | All records from both sides |
| **CROSS JOIN** | Every combination of rows |
| **SELF JOIN** | Table joins with itself |
| **NATURAL JOIN** | Joins automatically on same column names (⚠ avoid in real-time) |

## ▯ SQL JOINS — Real-time Interview Q&A (PostgreSQL)

### 🔰 LEVEL – 1: BEGINNER QUESTIONS

Q1. What is a JOIN in SQL?

Answer:

A JOIN is used to combine data from two or more tables based on a related column (usually a primary key–foreign key relationship).

Example:

```
SELECT e.emp_name, d.dept_name
FROM employees e
JOIN departments d ON e.dept_id = d.dept_id;
```

Real-time use: To fetch employee details along with their department names.

Q2. What is the difference between INNER JOIN and LEFT JOIN?

Answer:

INNER JOIN: returns only matching records from both tables.

LEFT JOIN: returns all records from the left table and matching ones from the right; unmatched rows show NULLs.

Example:

```
-- INNER JOIN

SELECT e.emp_name, d.dept_name

FROM employees e

INNER JOIN departments d ON e.dept_id = d.dept_id;


-- LEFT JOIN

SELECT e.emp_name, d.dept_name

FROM employees e

LEFT JOIN departments d ON e.dept_id = d.dept_id;
```

Real-time use:

To find which employees have or don’t have departments.

Q3. Can we join more than two tables?

Answer:

Yes, we can join multiple tables in a single query.

Example:

```
SELECT e.emp_name, d.dept_name, l.city

FROM employees e
```

```
JOIN departments d ON e.dept_id = d.dept_id
JOIN locations l ON d.location_id = l.location_id;
```

Real-time use:

In company reports, employee → department → location data is often joined together.

Q4. What is a SELF JOIN?

Answer:

A self join is when a table joins with itself.

Example:

```
SELECT e1.emp_name AS Employee, e2.emp_name AS Manager
FROM employees e1
JOIN employees e2 ON e1.manager_id = e2.emp_id;
```

Real-time use:

Used for hierarchical structures — employee–manager relationships.

Q5. What is a CROSS JOIN?

Answer:

A cross join returns the Cartesian product — every row of one table with every row of another.

Example:

```
SELECT e.emp_name, d.dept_name
FROM employees e
CROSS JOIN departments d;
```

Real-time use:

Used for generating combinations (e.g., products × colors × sizes).

⚙️ LEVEL – 2: INTERMEDIATE QUESTIONS

Q6. How do you find employees who don’t belong to any department?

Answer:

```
SELECT e.emp_name
FROM employees e
LEFT JOIN departments d ON e.dept_id = d.dept_id
WHERE d.dept_id IS NULL;
```

Real-time use:

To check unassigned employees or data mismatches.

Q7. How to get all departments even if they don’t have employees?

Answer:

```
SELECT d.dept_name, e.emp_name
FROM departments d
LEFT JOIN employees e ON d.dept_id = e.dept_id;
```

Real-time use:

For management reports that must show all departments.

Q8. What is the difference between WHERE and HAVING in joins?

Answer:

WHERE filters rows before grouping.

HAVING filters rows after grouping.

Example:

```
SELECT d.dept_name, COUNT(e.emp_id)
FROM departments d
LEFT JOIN employees e ON d.dept_id = e.dept_id
GROUP BY d.dept_name
HAVING COUNT(e.emp_id) > 2;
```

Real-time use:

To show only departments having more than two employees.

Q9. How to find department-wise highest salary using JOIN?

Answer:

```
SELECT d.dept_name, MAX(e.salary) AS highest_salary
FROM departments d
JOIN employees e ON d.dept_id = e.dept_id
GROUP BY d.dept_name;
```

Real-time use:

For salary analysis or payroll reports.

Q10. How to show employees with their department and salary greater than 50000?

Answer:

```
SELECT e.emp_name, d.dept_name, e.salary
FROM employees e
JOIN departments d ON e.dept_id = d.dept_id
WHERE e.salary > 50000;
```

Real-time use:

To generate HR reports of high-paid employees.

🚀 LEVEL – 3: ADVANCED (REAL-TIME COMPANY SCENARIOS)

Q11. What is the performance difference between JOIN and SUBQUERY?

Answer:

JOIN: Better performance when combining large datasets (optimized by the query planner).

SUBQUERY: Slower in some cases, especially when nested.

Best practice:

Use JOIN instead of subquery when you need data from multiple tables.

Q12. How do you find mismatched records between two tables?

Answer:

```
SELECT e.emp_id, e.emp_name, d.dept_id
FROM employees e
FULL OUTER JOIN departments d
ON e.dept_id = d.dept_id
WHERE e.dept_id IS NULL OR d.dept_id IS NULL;
```

Real-time use:

Used for data validation or data migration testing in companies.

Q13. Can we use JOIN inside a VIEW?

Answer:

Yes.

Example:

```
CREATE VIEW employee_department_view AS
SELECT e.emp_name, d.dept_name, e.salary
FROM employees e
JOIN departments d ON e.dept_id = d.dept_id;
```

Real-time use:

Views simplify complex reports in production systems.

Q14. How to find department names that have no employees and also employees with no departments (in a single query)?

Answer:

```
SELECT e.emp_name, d.dept_name
FROM employees e
FULL OUTER JOIN departments d
ON e.dept_id = d.dept_id
WHERE e.dept_id IS NULL OR d.dept_id IS NULL;
```

Real-time use:

Data audit — to find broken references between tables.

Q15. What is a NATURAL JOIN? Should we use it in real-time?

Answer:

A NATURAL JOIN automatically joins tables based on columns with the same name.

Example:

```
SELECT * FROM employees NATURAL JOIN departments;
```

⚠ Not recommended in real-time, because:

It joins automatically on all same-named columns (risk of wrong results).

Explicit JOIN ON condition is always safer.

Q16. How to join three tables to get Employee → Department → Location in PostgreSQL?

Answer:

```
SELECT e.emp_name, d.dept_name, l.city
FROM employees e
JOIN departments d ON e.dept_id = d.dept_id
JOIN locations l ON d.location_id = l.location_id;
```

Real-time use:

To prepare HR dashboards showing location-wise employee details.

Q17. Can we use alias names in JOINs?

Answer:

Yes, alias names make queries cleaner and shorter.

Example:

```
SELECT e.emp_name, d.dept_name
FROM employees AS e
JOIN departments AS d
ON e.dept_id = d.dept_id;
```

Real-time use:

Every project query uses aliases for readability.

Q18. What is the difference between LEFT JOIN and RIGHT JOIN in practical use?

Answer:

Both give similar results but with different perspective:

LEFT JOIN: keeps all data from the first (left) table.

RIGHT JOIN: keeps all data from the second (right) table.

Best practice:

Choose the one that aligns with your primary table focus.

Q19. What are some performance tuning tips for joins?

Answer:

1. Create indexes on columns used in JOIN conditions.
2. Avoid unnecessary FULL OUTER JOINs on big tables.
3. Use EXPLAIN ANALYZE in PostgreSQL to check query performance.
4. Always join on indexed key fields (PK–FK).

Q20. Real-time Scenario (Advanced):

> You have three tables: employees, departments, and projects.

Show all employees, their department names, and project names — even if any of them is missing.

Answer:

```
SELECT e.emp_name, d.dept_name, p.project_name
FROM employees e
FULL OUTER JOIN departments d ON e.dept_id = d.dept_id
FULL OUTER JOIN projects p ON e.project_id = p.project_id;
```

Real-time use:

To ensure complete visibility of data including missing references.