Chapter: Connecting Data for Better Insights

Topic: JOINS in SQL

Section 1: Learn

1.1 What Are JOINS in SQL?

In real-world databases, data is **spread across multiple tables**.

To analyze this data together, we use **JOINS** to combine related rows from different tables based on a common column.

Joins help you:

- Merge employee and department details
- Connect orders with customers
- Link students with their marks

1.2 Types of Joins

SQL offers several types of joins, each serving a different purpose.

1. **INNER JOIN** – Matches rows that exist in both tables
2. **LEFT JOIN** – Returns all rows from the left table, and matched rows from the right
3. **RIGHT JOIN** – Returns all rows from the right table, and matched rows from the left
4. **FULL JOIN (OUTER JOIN)** – Returns all rows when there is a match in either table

---

1.3 INNER JOIN

Use when: You want only the matching data from both tables.

Example:

```
SELECT employees.name, departments.name AS department
FROM employees
INNER JOIN departments
ON employees.dept_id = departments.id;
```

> Returns only employees who are assigned to a department.

1.4 LEFT JOIN (LEFT OUTER JOIN)

Use when: You want all records from the left table and matching ones from the right.

Example:

```
SELECT customers.name, orders.order_date
FROM customers
LEFT JOIN orders
ON customers.id = orders.customer_id;
```

> Includes all customers, even those who have not placed any orders.

1.5 RIGHT JOIN (RIGHT OUTER JOIN)

Use when: You want all records from the right table and matching ones from the left.

Example:

```
SELECT orders.order_id, customers.name
FROM orders
RIGHT JOIN customers
ON orders.customer_id = customers.id;
```

Returns all customers and their orders, including customers with no matching orders.

1.6 FULL OUTER JOIN

Use when: You want all data from both tables, whether they match or not.

Example (supported in PostgreSQL):

```
SELECT students.name, test_scores.score
FROM students
FULL OUTER JOIN test_scores
ON students.id = test_scores.student_id;
```

Returns students who didn't appear for tests and scores that don't match any student.

Note: MySQL does not directly support `FULL JOIN`. You can simulate using `UNION` of `LEFT JOIN` and `RIGHT JOIN`.

1.7 Choosing the Right JOIN

- Use **INNER JOIN** for strictly matched data
- Use **LEFT JOIN** when you want to preserve all from the primary table
- Use **RIGHT JOIN** in reverse situations

- Use **FULL JOIN** when you want a complete overview including non-matching rows

---

1.8 Join Conditions and Aliases

Using table aliases makes queries more readable.

```
SELECT e.name, d.name
FROM employees e
JOIN departments d
ON e.dept_id = d.id;
```

Improves clarity especially when working with long table names.

Section 2: Practise

Exercise 1: List Employees and Their Departments

```
SELECT e.name, d.name AS department
FROM employees e
INNER JOIN departments d
ON e.dept_id = d.id;
```

Exercise 2: Show All Customers With or Without Orders

```
SELECT c.name, o.order_id
FROM customers c
LEFT JOIN orders o
```

```
ON c.id = o.customer_id;
```

Exercise 3: List All Orders and Their Customers

```
SELECT o.order_id, c.name

FROM orders o

RIGHT JOIN customers c

ON o.customer_id = c.id;
```

Exercise 4: Get All Student and Score Data (including unmatched)

```
SELECT s.name, t.score

FROM students s

FULL OUTER JOIN test_scores t

ON s.id = t.student_id;
```

Exercise 5: Find Employees Without a Department

```
SELECT e.name

FROM employees e

LEFT JOIN departments d

ON e.dept_id = d.id

WHERE d.id IS NULL;
```

Section 3: FAQ – Know More

Q1. Can I join more than two tables?

Yes. You can join multiple tables by chaining JOINs:

```
SELECT o.order_id, c.name, p.product_name

FROM orders o

JOIN customers c ON o.customer_id = c.id

JOIN products p ON o.product_id = p.id;
```

Q2. What happens if join condition is missing?

It results in a **Cartesian Product** – every row of the first table is joined with every row of the second. This is rarely useful and should be avoided.

Q3. Can I filter joined results?

Yes. Use WHERE to filter the combined result:

```
SELECT *

FROM customers c

JOIN orders o ON c.id = o.customer_id

WHERE o.order_date >= '2024-01-01';
```

Q4. Which is better: JOIN or subquery?

It depends. Joins are faster for row-to-row matching. Subqueries are better for derived filtering and modular logic.

Q5. Do all databases support all join types?

Most support INNER, LEFT, and RIGHT joins.

FULL JOIN is not supported in MySQL by default — you must simulate it using UNION.

End of Notes for Chapter: Connecting Data for Better Insights