**Day 8-10: The World of JOINs**

**Subtopic 1: INNER JOIN**

* **Definition:** An INNER JOIN retrieves rows from two or more tables by matching a common column between them. It only returns the rows where the join condition is met in **both** tables. You can think of it as the intersection of two sets.
* **Use/Importance:** This is the most common type of join. It's used to combine related data from different tables. For example, fetching order details from an Orders table and the corresponding customer names from a Customers table.
* **Syntax/Structure:**

SELECT table1.column1, table2.column2

FROM table1

INNER JOIN table2

ON table1.common\_column = table2.common\_column;

* **Example:** To get a list of orders and the names of the customers who placed them.

SELECT o.order\_id, c.customer\_name

FROM Orders AS o

INNER JOIN Customers AS c

ON o.customer\_id = c.customer\_id;

**Subtopic 2: LEFT JOIN and RIGHT JOIN**

* **Definition:**
  + LEFT JOIN: Returns **all** rows from the left table (the first one mentioned) and the matching rows from the right table. If there is no match, the columns from the right table will contain NULL.
  + RIGHT JOIN: The inverse of a LEFT JOIN. It returns **all** rows from the right table and matching rows from the left.
* **Use/Importance:** Essential for analysis where you need to see all items from one category, regardless of whether they have a match in another. The most common use is finding items that *don't* have a corresponding entry in another table (e.g., "Show me all customers, even those who haven't placed an order").
* **Syntax/Structure:**

SQL

SELECT table1.column1, table2.column2

FROM table1

LEFT JOIN table2 ON table1.common\_column = table2.common\_column;

* **Example:** To list all customers and any order IDs they have, including customers who have never ordered.

SQL

SELECT c.customer\_name, o.order\_id

FROM Customers AS c

LEFT JOIN Orders AS o ON c.customer\_id = o.customer\_id;

**Subtopic 3: FULL OUTER JOIN**

* **Definition:** A FULL OUTER JOIN combines the results of both LEFT JOIN and RIGHT JOIN. It returns all rows from both tables. It will place NULL in the columns of the table where a matching row is not found.
* **Use/Importance:** Used when you need a complete and exhaustive view of two tables, showing all data from both and indicating where matches exist and where they don't. For example, listing all employees and all departments, showing which employees are unassigned and which departments are empty.
* **Syntax/Structure:**

SELECT table1.column1, table2.column2

FROM table1

FULL OUTER JOIN table2 ON table1.common\_column = table2.common\_column;

* **Example:** To list all employees and all company cars, showing who is assigned which car, which employees have no car, and which cars are unassigned.

SELECT e.employee\_name, c.car\_model

FROM Employees AS e

FULL OUTER JOIN CompanyCars AS c ON e.employee\_id = c.assigned\_employee\_id;

**Subtopic 4: UNION and UNION ALL**

* **Definition:** These operators are used to stack the results of two or more SELECT statements vertically.
  + UNION: Combines the results and removes any duplicate rows.
  + UNION ALL: Combines the results but includes all rows, including duplicates.
* **Use/Importance:** Useful for combining data from tables that have a similar structure but hold different sets of data (e.g., Active\_Customers and Archived\_Customers). UNION ALL is faster because it doesn't have to check for duplicates.
* **Syntax/Structure:**

SELECT column\_name(s) FROM table1

UNION -- or UNION ALL

SELECT column\_name(s) FROM table2;

* **Example:** To create a single mailing list from both customers and suppliers.

SELECT email FROM Customers

UNION

SELECT email FROM Suppliers;

**Quick Recap:**

* **INNER JOIN** is for finding the overlap or intersection between two tables.
* **LEFT JOIN** is for getting everything from the left table, plus any matching data from the right. Use it to find things that *don't* have a match.
* **FULL OUTER JOIN** provides a complete view of both tables, showing all matches and all non-matches from both sides.
* **UNION** stacks rows from multiple queries on top of each other and removes duplicates.
* When joining, NULL values appear on the side of the join that lacks a matching row (e.g., in a LEFT JOIN, NULLs will be in the columns from the right table).

**Practice Tasks:**

1. **Task 1:** Given a Customers table and an Orders table, write a query to list all customer names and the total number of orders they've placed. Crucially, **include customers who have never ordered**, who should show a count of 0.
2. **Task 2:** You have three tables: Products (with product\_id, product\_name), OrderItems (with order\_id, product\_id), and Orders (with order\_id, order\_date). Write a query to find the names of all products sold in January 2025. This will require joining three tables.
3. **Task 3:** Given a Departments table and an Employees table, write a query to find the names of any departments that have no employees.
4. **Task 4:** You have two tables: Sales\_Q1 and Sales\_Q2, both with identical columns (sale\_id, product\_id, amount). Write a query to get a single, combined list of all sales from the first half of the year. Include all records, even if a sale was recorded in both tables by mistake.