|  |  |
| --- | --- |
| **JOIN Type** | **What You Get** |
| INNER JOIN | Matches in both tables only |
| LEFT JOIN | All from left + matches from right |
| RIGHT JOIN | All from right + matches from left |
| FULL JOIN | All from both tables (use UNION in MySQL) |
| CROSS JOIN | All combinations (Cartesian product) |
| SELF JOIN | Join table to itself (e.g. comparing rows) |

CREATE TABLE Products (

product\_id INT PRIMARY KEY,

product\_name VARCHAR(100),

category VARCHAR(50),

unit\_price DECIMAL(10, 2)

);

CREATE TABLE Sales (

sale\_id INT PRIMARY KEY,

product\_id INT,

quantity\_sold INT,

sale\_date DATE,

total\_price DECIMAL(10, 2),

FOREIGN KEY (product\_id) REFERENCES Products(product\_id)

);

INSERT INTO Products (product\_id, product\_name, category, unit\_price) VALUES

(101, 'Laptop', 'Electronics', 500.00),

(102, 'Smartphone', 'Electronics', 300.00),

(103, 'Headphones', 'Electronics', 30.00),

(104, 'Keyboard', 'Electronics', 20.00),

(105, 'Mouse', 'Electronics', 15.00);

SELECT \* FROM Sales;

SELECT product\_name, unit\_price FROM Products;

SELECT \* FROM Sales WHERE total\_price > 100;

SELECT \* FROM Products WHERE category = 'Electronics';

SELECT sale\_id, total\_price

FROM Sales

WHERE sale\_date = '2024-01-03';

SELECT product\_id, product\_name

FROM Products

WHERE unit\_price > 100;

SELECT SUM(total\_price) AS total\_revenue

FROM Sales;

SELECT AVG(unit\_price) AS average\_unit\_price

FROM Products;

SELECT SUM(quantity\_sold) AS total\_quantity\_sold

FROM Sales;

SELECT sale\_date, COUNT(\*) AS sales\_count

FROM Sales

GROUP BY sale\_date

ORDER BY sale\_date;

Select sale\_date,COUNT(\*) from Sales group by sale\_date order by sale\_date

**Retrieve product\_name and unit\_price from the Products table with the Highest Unit Price**

select product\_name,unit\_price from Products order by unit\_price desc limit 1;

**Retrieve the sale\_id, product\_id, and total\_price from the Sales table for sales with a quantity\_sold greater than 4.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SUBQUERY\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

A **subquery** (also called an inner query or nested query) is a query nested inside another SQL query. Subqueries are often used in SELECT, FROM, or WHERE clauses to filter or calculate data.

Here are **4 common examples** of subqueries:

🔸 **1. Subquery in the WHERE Clause**

**Find employees whose salary is greater than the average salary.**

sql

CopyEdit

SELECT emp\_name, salary

FROM employees

WHERE salary > (

SELECT AVG(salary)

FROM employees

);

🔸 **2. Subquery in the FROM Clause**

**Get the average salary per department from a derived table.**

sql

CopyEdit

SELECT dept\_id, AVG(salary) AS avg\_dept\_salary

FROM (

SELECT dept\_id, salary

FROM employees

) AS dept\_salaries

GROUP BY dept\_id;

🔸 **3. Subquery in the SELECT Clause**

**Display employees with their department name fetched from a subquery.**

sql

CopyEdit

SELECT e.emp\_name,

(SELECT d.dept\_name FROM departments d WHERE d.dept\_id = e.dept\_id) AS department

FROM employees e;

🔸 **4. Correlated Subquery**

**List employees who earn more than the average salary of their own department.**

sql

CopyEdit

SELECT emp\_name, dept\_id, salary

FROM employees e

WHERE salary > (

SELECT AVG(salary)

FROM employees

WHERE dept\_id = e.dept\_id

);

✅ A **correlated subquery** uses values from the outer query and executes once for each row in the outer query.