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.