Excellent — now we'll move to **serious, structured, advanced SQL notes** with:

- Clean theory
- Practical explanations
- Large enough dataset
- Realistic queries
- Performance notes
- Interview-level clarity

We will use a **Corporate HR + Sales System** dataset (much better for advanced practice).

🔷 STEP 1 — ADVANCED PRACTICE DATABASE

Create a file:

```
advanced_practice.sql
```

Import with:

```
mysql -u root -p < advanced_practice.sql
```

📦 FULL PRACTICE DATASET

```
CREATE DATABASE IF NOT EXISTS advanced_db;
USE advanced_db;

-- Departments
CREATE TABLE departments (
    dept_id INT PRIMARY KEY AUTO_INCREMENT,
    dept_name VARCHAR(50) UNIQUE NOT NULL
);
```

```sql
-- Employees
CREATE TABLE employees (
    emp_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    salary DECIMAL(10,2) CHECK (salary > 0),
    dept_id INT,
    manager_id INT,
    hire_date DATE,
    FOREIGN KEY (dept_id) REFERENCES departments(dept_id),
    FOREIGN KEY (manager_id) REFERENCES employees(emp_id)
);

-- Customers
CREATE TABLE customers (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_name VARCHAR(100),
    city VARCHAR(50),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Orders
CREATE TABLE orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    order_date DATE,
    total_amount DECIMAL(12,2),
    status VARCHAR(20),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

-- Products
CREATE TABLE products (
    product_id INT PRIMARY KEY AUTO_INCREMENT,
    product_name VARCHAR(100),
    price DECIMAL(10,2),
    stock INT
);

-- Order Items
```

```
CREATE TABLE order_items (
    order_item_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT,
    product_id INT,
    quantity INT,
    price DECIMAL(10,2),
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

Sample Data (Enough for Analytics)

```
INSERT INTO departments (dept_name)
VALUES ('Engineering'), ('Sales'), ('HR'), ('Finance');

INSERT INTO employees (first_name, last_name, email, salary,
dept_id, manager_id, hire_date)
VALUES
('Aarav','Sharma','aarav@corp.com',90000,1,NULL,'2019-01-10'),
('Isha','Patel','isha@corp.com',80000,1,1,'2020-03-12'),
('Rohan','Verma','rohan@corp.com',70000,2,NULL,'2021-06-18'),
('Meera','Kapoor','meera@corp.com',75000,2,3,'2022-07-20'),
('Anjali','Rao','anjali@corp.com',60000,3,NULL,'2023-02-15');

INSERT INTO customers (customer_name, city)
VALUES
('Rahul','Delhi'),
('Sneha','Mumbai'),
('Arjun','Delhi'),
('Kiran','Pune');

INSERT INTO products (product_name, price, stock)
VALUES
('Laptop',70000,50),
('Phone',40000,100),
('Tablet',30000,75),
('Headphones',5000,200);

INSERT INTO orders (customer_id, order_date, total_amount, status)
VALUES
```

```
(1,'2023-01-10',75000,'Completed'),
(2,'2023-02-15',40000,'Completed'),
(1,'2023-03-01',30000,'Pending'),
(3,'2023-03-20',5000,'Completed');

INSERT INTO order_items (order_id, product_id, quantity, price)
VALUES
(1,1,1,70000),
(1,4,1,5000),
(2,2,1,40000),
(3,3,1,30000),
(4,4,1,5000);
```

🔷 PART 1 — ADVANCED SELECT & FILTERING

Logical Execution Order of SQL

Important for interviews:

1. FROM
2. JOIN
3. WHERE
4. GROUP BY
5. HAVING
6. SELECT
7. ORDER BY
8. LIMIT

Complex Filtering

```sql
SELECT *
FROM employees
WHERE salary BETWEEN 70000 AND 90000
AND dept_id IN (1,2)
AND hire_date > '2020-01-01';
```

Pattern Matching

```sql
SELECT * FROM employees
WHERE first_name LIKE 'A%';
```

🔷 PART 2 — AGGREGATION & ANALYTICS

Group Analytics

```sql
SELECT dept_id, COUNT(*) AS total_employees,
       AVG(salary) AS avg_salary
FROM employees
GROUP BY dept_id;
```

Find Departments With Avg Salary > 75000

```sql
SELECT dept_id, AVG(salary)
FROM employees
GROUP BY dept_id
HAVING AVG(salary) > 75000;
```

Business Insight Query

Find total revenue per customer:

```sql
SELECT c.customer_name,
       SUM(o.total_amount) AS total_spent
```

```
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id;
```

🔷 PART 3 — ADVANCED JOINS

Multi-Table Join

```
SELECT c.customer_name,
       p.product_name,
       oi.quantity
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id;
```

Self Join (Manager Relationship)

```
SELECT e.first_name AS employee,
       m.first_name AS manager
FROM employees e
LEFT JOIN employees m
ON e.manager_id = m.emp_id;
```

🔷 PART 4 — SUBQUERIES

Employees Earning Above Average

```
SELECT *
FROM employees
WHERE salary >
      (SELECT AVG(salary) FROM employees);
```

Correlated Subquery

Find employees earning more than department average:

```
SELECT e.*
FROM employees e
WHERE salary >
(
    SELECT AVG(salary)
    FROM employees
    WHERE dept_id = e.dept_id
);
```

🔷 PART 5 — INDEXES (Performance)

Create Index

```
CREATE INDEX idx_salary ON employees(salary);
```

Check Execution Plan

```
EXPLAIN SELECT * FROM employees WHERE salary > 80000;
```

🔷 PART 6 — TRANSACTIONS

Example: Stock Deduction + Order Creation

```
START TRANSACTION;

UPDATE products
SET stock = stock - 1
WHERE product_id = 1;

INSERT INTO orders(customer_id, order_date, total_amount, status)
VALUES (2, CURDATE(), 70000, 'Pending');
```

```
COMMIT;
```

If failure:

```
ROLLBACK;
```

🔷 PART 7 — VIEWS

```
CREATE VIEW high_value_customers AS
SELECT c.customer_name,
       SUM(o.total_amount) AS total_spent
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id
HAVING total_spent > 50000;
```

🔷 PART 8 — STORED PROCEDURE

```
DELIMITER //

CREATE PROCEDURE GetEmployeeByDept(IN dept INT)
BEGIN
    SELECT * FROM employees
    WHERE dept_id = dept;
END //

DELIMITER ;
```

Call:

```
CALL GetEmployeeByDept(1);
```

🔷 PART 9 — WINDOW FUNCTIONS (Advanced)

(If using MySQL 8+)

Rank Employees by Salary

```
SELECT first_name,
       salary,
       RANK() OVER (ORDER BY salary DESC) AS salary_rank
FROM employees;
```

Running Total

```
SELECT order_date,
       total_amount,
       SUM(total_amount)
       OVER (ORDER BY order_date) AS running_total
FROM orders;
```

🔷 PART 10 — PRACTICE QUESTIONS (Advanced)

1. Find top 2 highest paid employees per department.
2. Find customers who placed more than 1 order.
3. Find product generating highest revenue.
4. Find monthly revenue.
5. Find managers whose team salary > 150000.
6. Write transaction to cancel order and restore stock.