# Advanced SQL Tasks

## Example Table:

### 1. employees Table

```
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR(50),
    salary INT,
    gender VARCHAR(10),
    dept_id INT,
    manager_id INT
);

INSERT INTO employees VALUES
(1, 'Alice', 70000, 'Female', 101, NULL),
(2, 'Bob', 50000, 'Male', 101, 1),
(3, 'Charlie', 50000, 'Male', 101, 1),
(4, 'David', 60000, 'Male', 102, 1),
(5, 'Eve', 75000, 'Female', 102, NULL),
(6, 'Frank', 80000, 'Male', 103, 5),
(7, 'Grace', 50000, 'Female', 103, 5),
(8, 'Heidi', 50000, 'Female', 101, 1),
(9, 'Ivan', 60000, 'Male', 101, 1),
(10, 'Judy', 70000, 'Female', 102, 5);
```

### 2. `students` Table

```
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(50),
    course_id INT
);

INSERT INTO students VALUES
(1, 'Sam', 100),
(2, 'John', 101),
(3, 'Sam', 100),
(4, 'Alice', 102),
(5, 'John', 101);
```

### 3. `invoices` Table

```
CREATE TABLE invoices (
    invoice_id INT PRIMARY KEY
);

INSERT INTO invoices VALUES
(100), (101), (103), (104), (106);
```

### 4. `attendance` Table

```
CREATE TABLE attendance (
    emp_id INT,
    attendance_date DATE,
    status VARCHAR(10)
);

INSERT INTO attendance VALUES
(1, '2024-06-01', 'Present'),
(1, '2024-06-02', 'Present'),
(1, '2024-06-03', 'Present'),
(2, '2024-06-01', 'Absent'),
(2, '2024-06-02', 'Present'),
(2, '2024-06-03', 'Present'),
(2, '2024-06-04', 'Present');
```

### 5. `project_assignment` Table

```
CREATE TABLE project_assignment (
    emp_id INT
);

INSERT INTO project_assignment VALUES
(2), (3), (5), (7);
```

### 6. `users` Table

```
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    dob DATE
);

INSERT INTO users VALUES
(1, '1990-05-21'),
(2, '1985-12-15'),
(3, '2000-08-08');
```

### 7. `products` Table

```
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    category_id INT,
    price INT
);

INSERT INTO products VALUES
(1, 10, 300),
(2, 10, 500),
```

```
(3, 10, 700),
(4, 20, 100),
(5, 20, 120),
(6, 20, 90),
(7, 30, 800),
(8, 30, 1000);
```

### 8. `sales` Table

```
CREATE TABLE sales (
    emp_id INT,
    amount INT
);

INSERT INTO sales VALUES
(1, 5000),
(2, 7000),
(3, 3000),
(4, 10000);
```

### 9. `orders` Table

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    amount INT
);

INSERT INTO orders VALUES
(101, 250),
(102, 800),
(103, 1200),
(104, 450);
```

### 10. `transactions` Table

```
CREATE TABLE transactions (
    txn_id INT PRIMARY KEY,
    user_id INT,
    txn_date DATE,
    amount INT
);

INSERT INTO transactions VALUES
(1, 101, '2024-06-01', 500),
(2, 101, '2024-06-05', 700),
(3, 102, '2024-06-02', 300),
(4, 102, '2024-06-04', 200),
(5, 103, '2024-06-03', 1000);
```

---

**1. Find the Second Highest Salary Without Using LIMIT or TOP**

**Table:** `employees(emp_id, emp_name, salary)`
**Task:** Write a query to find the second highest salary.

**Answer:**

```
SELECT MAX(salary) AS SecondHighestSalary
FROM employees
WHERE salary < (SELECT MAX(salary) FROM employees);
```

**2. Find Duplicates in a Table**

**Table:** `students(student_id, student_name, course_id)`
**Task:** Find all duplicate entries based on student name.

**Answer:**

```
SELECT student_name, COUNT(*)
FROM students
GROUP BY student_name
HAVING COUNT(*) > 1;
```

**3. Retrieve Nth Highest Salary Using CTE**

**Task:** Get the 3rd highest salary.

**Answer:**

```
WITH SalaryRank AS (
  SELECT salary, DENSE_RANK() OVER (ORDER BY salary DESC) AS
rnk
  FROM employees
)
SELECT salary
FROM SalaryRank
WHERE rnk = 3;
```

**4. Find Employees Who Have the Same Salary**

**Task:** List all employees who share the same salary with at least one other employee.

**Answer:**

```
SELECT *
FROM employees
WHERE salary IN (
  SELECT salary
  FROM employees
  GROUP BY salary
  HAVING COUNT(*) > 1
```

```
);
```

**5. Calculate Running Total (Cumulative Sum)**

**Task:** Get running total of salary.

**Answer:**

```
SELECT emp_id, emp_name, salary,
       SUM(salary) OVER (ORDER BY emp_id) AS running_total
FROM employees;
```

**6. Find Gaps in Sequence**

**Table:** `invoices(invoice_id)`
**Task:** Find missing invoice numbers.

**Answer:**

```
SELECT invoice_id + 1 AS missing_id
FROM invoices
WHERE (invoice_id + 1) NOT IN (SELECT invoice_id FROM
invoices);
```

**7. Find Departments With More Than 3 Employees**

**Table:** `employees(emp_id, dept_id)`
**Task:** Get department IDs having more than 3 employees.

**Answer:**

```
SELECT dept_id, COUNT(*) AS emp_count
FROM employees
GROUP BY dept_id
HAVING COUNT(*) > 3;
```

**8. Pivot Table Using CASE**

**Task:** Show number of male and female employees in each department.

**Answer:**

```
SELECT dept_id,
       COUNT(CASE WHEN gender = 'Male' THEN 1 END) AS
male_count,
       COUNT(CASE WHEN gender = 'Female' THEN 1 END) AS
female_count
FROM employees
GROUP BY dept_id;
```

### 9. Delete Duplicate Rows

**Table:** `students(student_id, student_name, course_id)`
**Task:** Delete duplicate records, keeping only one.

**Answer:**

```
DELETE FROM students
WHERE rowid NOT IN (
  SELECT MIN(rowid)
  FROM students
  GROUP BY student_name, course_id
);
```

### 10. Correlated Subquery Example

**Task:** Get employees who earn more than the average salary in their department.

**Answer:**

```
SELECT emp_id, emp_name, salary, dept_id
FROM employees e
WHERE salary > (
  SELECT AVG(salary)
  FROM employees
  WHERE dept_id = e.dept_id
);
```

### 11. Find Consecutive Attendance Records

**Table:** `attendance(emp_id, attendance_date, status)`
**Task:** Find employees who were present on **3 consecutive days**.

**Answer:**

```
SELECT DISTINCT a1.emp_id
FROM attendance a1
JOIN attendance a2 ON a1.emp_id = a2.emp_id AND
a2.attendance_date = a1.attendance_date + INTERVAL 1 DAY
JOIN attendance a3 ON a1.emp_id = a3.emp_id AND
a3.attendance_date = a1.attendance_date + INTERVAL 2 DAY
WHERE a1.status = 'Present' AND a2.status = 'Present' AND
a3.status = 'Present';
```

### 12. Find Managers with More Than 5 Employees

**Table:** `employees(emp_id, emp_name, manager_id)`
**Task:** List managers who manage more than 5 employees.

**Answer:**

```
SELECT manager_id, COUNT(*) AS employee_count
FROM employees
GROUP BY manager_id
HAVING COUNT(*) > 5;
```

### 13. Identify Employees with No Projects

**Tables:** `employees(emp_id)`,`project_assignment(emp_id)`
**Task:** Find employees who are **not assigned** to any project.

**Answer:**

```
SELECT emp_id
FROM employees
WHERE emp_id NOT IN (SELECT emp_id FROM project_assignment);
```

### 14. Calculate Age from Date of Birth

**Table:** `users(user_id, dob)`
**Task:** Display age of each user.

**Answer:**

```
SELECT user_id,
       TIMESTAMPDIFF(YEAR, dob, CURDATE()) AS age
FROM users;
```

### 15. Find Top N Records per Category

**Table:** `products(product_id, category_id, price)`
**Task:** Get **top 2 costliest products** in each category.

**Answer:**

```
SELECT *
FROM (
  SELECT *,
         DENSE_RANK() OVER (PARTITION BY category_id ORDER BY
price DESC) AS rnk
  FROM products
) ranked
WHERE rnk <= 2;
```

### 16. Identify Salary Difference from Department Average

**Table:** `employees(emp_id, dept_id, salary)`
**Task:** Show employees whose salary differs from their department's average.

**Answer:**

```
SELECT emp_id, salary, dept_id,
       salary - AVG(salary) OVER (PARTITION BY dept_id) AS
diff_from_avg
FROM employees;
```

### 17. Self Join to Find Colleagues in the Same Department

**Task:** List each employee along with their colleagues in the same department.

**Answer:**

```
SELECT e1.emp_name AS employee, e2.emp_name AS colleague
FROM employees e1
JOIN employees e2
ON e1.dept_id = e2.dept_id AND e1.emp_id != e2.emp_id;
```

### 18. Calculate Percentage Contribution to Total Sales

**Table:** `sales(emp_id, amount)`
**Task:** Find each employee's sales percentage.

**Answer:**

```
SELECT emp_id, amount,
       ROUND((amount / SUM(amount) OVER()) * 100, 2) AS
percentage
FROM sales;
```

### 19. Using CASE for Custom Grouping

**Table:** `orders(order_id, amount)`
**Task:** Classify orders as 'Small', 'Medium', or 'Large'.

**Answer:**

```
SELECT order_id, amount,
       CASE
         WHEN amount < 500 THEN 'Small'
         WHEN amount BETWEEN 500 AND 1000 THEN 'Medium'
         ELSE 'Large'
       END AS order_size
FROM orders;
```

### 20. Find Most Recent Record per Group

**Table:** `transactions(user_id, txn_date, amount)`
**Task:** Find the **latest transaction** for each user.

**Answer:**

```
SELECT *
FROM (
  SELECT *,
         ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY
txn_date DESC) AS rnk
  FROM transactions
) t
WHERE rnk = 1;
```