1. Write a Query to Find the Second Highest Salary

```
SELECT MAX(salary)
FROM employee
WHERE salary < (SELECT MAX(salary) FROM employee);
```

- **Explanation:** This query uses a subquery to find the highest salary and then uses it in the WHERE clause to get the second highest salary.

2. Fetch Employees With the Highest Salary in Each Department

```
SELECT department, employee_name, salary
FROM (
   SELECT department, employee_name, salary,
        RANK() OVER (PARTITION BY department ORDER BY salary DESC) as rank
   FROM employee
) ranked_salaries
WHERE rank = 1;
```

- **Explanation:** This query uses the RANK() window function to rank employees based on salary within each department and fetches only those with rank 1.

3. Write a Query to Find Duplicate Rows in a Table

```
SELECT name, COUNT(*)
FROM employees
GROUP BY name
HAVING COUNT(*) > 1;
```

- **Explanation:** The GROUP BY clause groups rows by the name column, and HAVING filters groups with more than one entry.

4. Delete Duplicate Rows from a Table

```
WITH duplicates AS (
   SELECT name, ROW_NUMBER() OVER(PARTITION BY name ORDER BY id) as row_num
```

```
   FROM employees
)
DELETE FROM employees
WHERE id IN (SELECT id FROM duplicates WHERE row_num > 1);
```

- **Explanation:** The ROW_NUMBER() function assigns a unique number to duplicate rows, and the query deletes rows where the assigned number is greater than 1.

5. Find Employees Who Joined in the Last 6 Months

```
SELECT *
FROM employees
WHERE join_date >= DATEADD(month, -6, GETDATE());
```

- **Explanation:** The query fetches employees who joined within the last 6 months using the DATEADD function.

6. Write a Query to Find the Number of Employees in Each Department

```
SELECT department, COUNT(*) as total_employees
FROM employees
GROUP BY department;
```

- **Explanation:** This query uses GROUP BY to count the number of employees in each department.

7. Write a Query to Find the Nth Highest Salary

```
SELECT DISTINCT salary
FROM employees
ORDER BY salary DESC
LIMIT 1 OFFSET n-1;
```

- **Explanation:** The LIMIT and OFFSET keywords help fetch the nth highest salary. Replace n with the desired rank (e.g., n = 3 for the third highest).

8. Self-Join to Find Employees With the Same Manager

```
SELECT e1.employee_name, e2.employee_name AS colleague_name, e1.manager_id
FROM employees e1
JOIN employees e2 ON e1.manager_id = e2.manager_id AND e1.employee_id !=
e2.employee_id;
```

- **Explanation:** This query performs a self-join to find employees working under the same manager but are not the same employee.

9. Find Employees Who Do Not Have a Manager Assigned (Using LEFT JOIN)

```
SELECT e.employee_name
FROM employees e
LEFT JOIN managers m ON e.manager_id = m.manager_id
WHERE m.manager_id IS NULL;
```

- **Explanation:** The LEFT JOIN fetches all employees, and the WHERE clause filters those who do not have a corresponding manager.

10. Display the Department Names and Their Total Salaries

```
SELECT d.department_name, SUM(e.salary) as total_salary
FROM employees e
JOIN departments d ON e.department_id = d.department_id
GROUP BY d.department_name;
```

- **Explanation:** This query joins the employees and departments tables, groups by department, and calculates the total salary per department.

11. Fetch Top 3 Salaries from Each Department

```
SELECT department, employee_name, salary
FROM (
   SELECT department, employee_name, salary,
        DENSE_RANK() OVER (PARTITION BY department ORDER BY salary DESC) as rank
   FROM employees
) ranked_salaries
WHERE rank <= 3;
```

- **Explanation:** The DENSE_RANK() function is used to rank salaries within each department, and the query fetches only the top 3.

12. Find Common Employees Between Two Tables (Using INTERSECT)

```
SELECT employee_id, employee_name
FROM employees_2023
INTERSECT
SELECT employee_id, employee_name
FROM employees_2024;
```

- **Explanation:** This query uses the INTERSECT keyword to find common rows between two employee tables.

13. Find All Managers Who Have More Than 5 Employees Reporting to Them

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

- **Explanation:** This query groups employees by their manager and filters those with more than five direct reports.

14. Subquery to Find Employees With Salary Above Department Average

```
SELECT employee_name, salary, department
FROM employees e
WHERE salary > (SELECT AVG(salary) FROM employees WHERE department = e.department);
```

- **Explanation:** The subquery calculates the average salary for each department, and the outer query fetches employees earning above this average.

15. Update a Column with Conditional Logic (Using CASE Statement)

```
UPDATE employees
```

```
SET bonus = CASE
   WHEN salary > 100000 THEN 10000
   WHEN salary BETWEEN 50000 AND 100000 THEN 5000
   ELSE 2000
END;
```

- **Explanation:** This query uses the CASE statement to update the bonus column based on salary ranges.

16 Find the Manager with the Maximum Number of Direct Reports

```
SELECT manager_id, COUNT(*) as num_reports
FROM employees
GROUP BY manager_id
ORDER BY num_reports DESC
LIMIT 1;
```

- **Explanation:** This query groups employees by manager_id and finds the one with the most direct reports using ORDER BY and LIMIT.

17. Display All Employees Whose Name Starts with 'A' and Ends with 'N'

```
SELECT *
FROM employees
WHERE employee_name LIKE 'A%N';
```

- **Explanation:** The LIKE clause with % wildcard checks for names starting with 'A' and ending with 'N'.

18. Find Employees Who Joined in the Year 2023

```
SELECT employee_name, join_date
FROM employees
WHERE YEAR(join_date) = 2023;
```

- **Explanation:** This query extracts the year part of join_date using the YEAR() function.

19. Calculate the Cumulative Salary by Department Using SUM with OVER Clause

```
SELECT department, employee_name, salary,
    SUM(salary) OVER (PARTITION BY department ORDER BY employee_name) as
cumulative_salary
FROM employees;
```

- **Explanation:** This uses the window function SUM with OVER to compute the cumulative salary within each department.

20. Fetch the Median Salary of All Employees

```
SELECT AVG(salary) as median_salary
FROM (
  SELECT salary
  FROM employees
  ORDER BY salary
  LIMIT 2 - (SELECT COUNT(*) FROM employees) % 2
  OFFSET (SELECT (COUNT(*) - 1) / 2 FROM employees)
) as median;
```

- **Explanation:** This query finds the median salary using a subquery to get the middle value(s) and calculates the average if there are two middle values.

21. Write a Query to Swap Two Column Values Without Using a Temporary Column

```
UPDATE employees
SET column1 = column1 + column2,
  column2 = column1 - column2,
  column1 = column1 - column2;
```

- **Explanation:** This uses arithmetic operations to swap values between column1 and column2 without a temporary variable.

22. Find Departments Without Any Employees

```
SELECT d.department_id, d.department_name
FROM departments d
```

```
LEFT JOIN employees e ON d.department_id = e.department_id
WHERE e.employee_id IS NULL;
```

- **Explanation:** The LEFT JOIN fetches all departments, and the WHERE clause filters out those with no employees.

23. Find the Total Salary by Department and Also Include Departments with No Employees

```
SELECT d.department_name, IFNULL(SUM(e.salary), 0) as total_salary
FROM departments d
LEFT JOIN employees e ON d.department_id = e.department_id
GROUP BY d.department_name;
```

- **Explanation:** The query uses LEFT JOIN to include all departments, even those without employees, and IFNULL to handle NULL cases.

24. Fetch Top 5 Employees with the Highest Salaries Without Using LIMIT

```
SELECT *
FROM (
   SELECT employee_name, salary,
        DENSE_RANK() OVER (ORDER BY salary DESC) as salary_rank
   FROM employees
) ranked_salaries
WHERE salary_rank <= 5;
```

- **Explanation:** This query uses DENSE_RANK() to rank salaries and fetch the top 5 without relying on LIMIT.

25. Display Employee Names in a Single String for Each Department (GROUP_CONCAT)

```
SELECT department, GROUP_CONCAT(employee_name ORDER BY employee_name ASC) as
employees
FROM employees
GROUP BY department;
```

- **Explanation:** The GROUP_CONCAT() function concatenates employee names in each department into a single string.