**Nested queries or subqueries in MySQL are queries that are embedded within another query. They are enclosed within parentheses and can be used in various parts of a SQL statement such as SELECT, FROM, WHERE, and HAVING clauses. Here are some examples to illustrate the usage of nested queries in different scenarios:**

**1. Filtering Data with a Subquery:**

SELECT \* FROM employees

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

- This query retrieves employees whose salary is higher than the average salary.

- The subquery calculates the average salary and provides it as a filter for the outer query.

**2. Selecting from a Set Returned by a Subquery:**

SELECT name, department FROM employees

WHERE department IN (SELECT department FROM departments WHERE location = 'New York');

- This query selects names and departments of employees who work in departments located in New York.

- The subquery fetches departments in New York, and the outer query uses those results to filter employee records.

**3. Updating Data Based on a Subquery:**

UPDATE employees

SET salary = salary \* 1.1

WHERE department IN (SELECT department FROM departments WHERE budget > 500000);

- This query increases the salary of employees in departments with a budget over 500,000 by 10%.

- The subquery identifies eligible departments, and the outer query applies the salary update.

**4. Inserting Data from a Subquery:**

INSERT INTO employees (name, department)

SELECT name, department FROM new\_hires;

- This query inserts employee names and departments from a table named new\_hires into the employees table.

- The subquery retrieves data from new\_hires, and the outer query inserts it into employees.

**5. Creating a Derived Table with a Subquery:**

SELECT \* FROM (SELECT name, MAX(salary) AS max\_salary FROM employees GROUP BY department) AS salary\_ranks;

- This query creates a temporary table (derived table) named salary\_ranks that contains the highest salary for each department.

- The subquery calculates the maximum salaries, and the outer query treats the result as a table to select from.

**6. Subquery in WHERE Clause:**

-- Find employees who earn more than the average salary in their department

SELECT employee\_id, employee\_name, salary, department\_id

FROM employees

WHERE salary > (SELECT AVG(salary) FROM employees GROUP BY department\_id);

**7. Subquery in SELECT Clause:**

-- Retrieve employee details along with their department's total salary

SELECT employee\_id, employee\_name, salary, department\_id,

(SELECT SUM(salary) FROM employees WHERE department\_id = e.department\_id) as department\_total\_salary

FROM employees e;

**8. Subquery in FROM Clause:**

-- Find the average salary of employees in each department and list departments with average salary greater than a threshold

SELECT department\_id, avg\_salary

FROM (SELECT department\_id, AVG(salary) as avg\_salary

FROM employees

GROUP BY department\_id) AS department\_avg

WHERE avg\_salary > 50000;

**9. Subquery in HAVING Clause:**

-- Find departments with more than 3 employees and their average salary is above a certain threshold

SELECT department\_id, COUNT(employee\_id) as num\_employees, AVG(salary) as avg\_salary

FROM employees

GROUP BY department\_id

HAVING num\_employees > 3 AND avg\_salary > 60000;

**10. Correlated Subquery:**

-- Find employees whose salary is above the average salary in their department

-- This uses a correlated subquery as it references the outer query

SELECT employee\_id, employee\_name, salary, department\_id

FROM employees e

WHERE salary > (SELECT AVG(salary) FROM employees WHERE department\_id = e.department\_id);

**11. Subquery with EXISTS:**

-- Find departments that have at least one employee

SELECT department\_id, department\_name

FROM departments d

WHERE EXISTS (SELECT 1 FROM employees e WHERE e.department\_id = d.department\_id);

**12. Subquery with IN:**

-- Find employees who work in departments with specific IDs

SELECT employee\_id, employee\_name, department\_id

FROM employees

WHERE department\_id IN (SELECT department\_id FROM departments WHERE location\_id = 1700);

**13. Subquery with NOT IN:**

-- Find employees who don't have a manager

SELECT employee\_id, employee\_name, manager\_id

FROM employees

WHERE manager\_id NOT IN (SELECT employee\_id FROM employees WHERE manager\_id IS NOT NULL);

**14. Scalar Subquery:**

-- Find employees and their bonus percentage based on the average salary

SELECT employee\_id, employee\_name, salary, salary / (SELECT AVG(salary) FROM employees) as bonus\_percentage

FROM employees;

**15. Subquery with ORDER BY and LIMIT:**

-- Find the top 5 highest-paid employees

SELECT employee\_id, employee\_name, salary

FROM employees

ORDER BY salary DESC

LIMIT 5;

**16. Subquery with Multiple Conditions:**

-- Find employees who work in departments with more than 10 employees and have a salary greater than 60000

SELECT employee\_id, employee\_name, department\_id, salary

FROM employees

WHERE department\_id IN (SELECT department\_id FROM employees GROUP BY department\_id HAVING COUNT(\*) > 10)

AND salary > 60000;

**17. Subquery with JOIN:**

-- Find employees and their department names using a subquery with a JOIN

SELECT employee\_id, employee\_name, department\_name

FROM employees e

JOIN (SELECT department\_id, department\_name FROM departments) d ON e.department\_id = d.department\_id;

**18. Subquery with Aggregation in HAVING Clause:**

-- Find departments with the highest average salary and list only those with an average salary greater than 70000

SELECT department\_id, AVG(salary) as avg\_salary

FROM employees

GROUP BY department\_id

HAVING AVG(salary) > (SELECT MAX(avg\_salary) FROM (SELECT AVG(salary) as avg\_salary FROM employees GROUP BY department\_id) AS department\_avg);

**19. Subquery with Date Comparison:**

-- Find employees hired after the average hiring date of the company

SELECT employee\_id, employee\_name, hire\_date

FROM employees

WHERE hire\_date > (SELECT AVG(hire\_date) FROM employees);

**20. Subquery with NULL Comparison:**

-- Find employees who do not have a manager

SELECT employee\_id, employee\_name, manager\_id

FROM employees

WHERE manager\_id IS NULL;

**21. Subquery for Inserting Records:**

-- Insert a new department and assign it a manager by retrieving the employee ID using a subquery

INSERT INTO departments (department\_name, manager\_id)

VALUES ('Marketing', (SELECT employee\_id FROM employees WHERE employee\_name = 'John Doe'));

**22. Subquery in a JOIN Condition:**

-- Find employees and their corresponding department names using a subquery in a JOIN condition

SELECT employee\_id, employee\_name, department\_name

FROM employees e

JOIN departments d ON e.department\_id = (SELECT department\_id FROM departments WHERE department\_name = 'Sales');

**23. Subquery with EXISTS in DELETE Statement:**

-- Delete employees who do not have any dependent records in the dependents table

DELETE FROM employees

WHERE NOT EXISTS (SELECT 1 FROM dependents WHERE employee\_id = employees.employee\_id);

**24. Subquery in CASE Statement:**

-- Classify employees based on their salary using a subquery in a CASE statement

SELECT employee\_id, employee\_name, salary,

CASE

WHEN salary > (SELECT AVG(salary) FROM employees) THEN 'Above Average'

ELSE 'Below Average'

END AS salary\_classification

FROM employees;

**25. Subquery with ALL:**

-- Find employees whose salary is higher than all employees in the Marketing department

SELECT employee\_id, employee\_name, salary

FROM employees

WHERE salary > ALL (SELECT salary FROM employees WHERE department\_id = (SELECT department\_id FROM departments WHERE department\_name = 'Marketing'));

**1. Find Employees Earning More Than Their Department's Average:**

* Write a query to list employee names, departments, salaries, and their department's average salary, only for those earning more than their department's average.

**2. Customers with Orders in Multiple Countries:**

* Retrieve customer names and countries who have placed orders shipped to at least three different countries.

**3. Products Never Ordered:**

* Identify product names and categories that have never been ordered.

**4. Second Highest Salary in Each Department:**

* Display the second highest salary in each department, along with the department name.

**5. Customers with Orders Exceeding a Specific Value:**

* List customer names, total order amounts, and their sales representatives for customers with total orders exceeding $5000.

**6. Products with Highest Quantity in Each Category:**

* Show the product name, category, and quantity on hand for the product with the highest quantity in each category.

**7. Employees with Same Manager as Highest Paid Employee:**

* Find employee names, managers, and salaries for employees who have the same manager as the highest paid employee in the company.

**8. Products Ordered by All Customers in a Specific City:**

* List product names and categories that have been ordered by all customers in a specified city (e.g., London).

**9. Departments with No Employees:**

* Identify department names that have no employees assigned to them.

**10. Customers with Orders in All Months:**

* Retrieve customer names and IDs who have placed orders in every month of the current year.

**Tips for Practice:**

* Use a sample database (e.g., Sakila) to practice.
* Break down complex queries into smaller subqueries.
* Use aliases for readability.
* Test your queries thoroughly.
* Explore different ways to achieve the same result.