**1. A subquery is a query nested inside another query,**

SELECT

\*

FROM table1

WHERE column1 IN (

SELECT

column2

FROM table2

WHERE column3 = 'value'

);

**2. Subqueries can also be used in conjunction with joins to further refine the data returned.**

SELECT

\*

FROM table1

INNER JOIN (

SELECT column2

FROM table2

WHERE column3 = 'value'

) AS subquery ON table1.column = subquery.column2;

**1. Retrieve those Employee -> (Ern Salery more) (than average sallery ) (Base On Department) (3 query bna ge)**

Suppose we want to retrieve a list of all employees who have ever earned a salary greater than the average salary of their department. We can use a subquery to accomplish this, like so:

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ table s\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TABLE employees (

employee\_id INT PRIMARY KEY,

employee\_name VARCHAR(50),

department\_id INT

);

CREATE TABLE salaries (

employee\_id INT PRIMARY KEY,

salary\_amount DECIMAL(10,2)

);

INSERT INTO employees VALUES

(1, 'John Smith', 1),

(2, 'Jane Doe', 1),

(3, 'Bob Johnson', 2),

(4, 'Alice Brown', 2),

(5, 'Tom Jones', 3),

(6, 'Sara Lee', 3),

(7, 'Mike Smith', 4),

(8, 'Lisa Wang', 4),

(9, 'David Kim', 5),

(10, 'Emily Chen', 5);

INSERT INTO salaries VALUES

(1, 50000),

(2, 60000),

(3, 45000),

(4, 55000),

(5, 65000),

(6, 70000),

(7, 60000),

(8, 55000),

(9, 75000),

(10, 80000);

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ sub Query \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

FROM employees e

JOIN salaries s ON e.employee\_id = s.employee\_id

WHERE salary\_amount > (

SELECT AVG(salary\_amount)

FROM salaries

WHERE employee\_id IN (

SELECT employee\_id

FROM employees

WHERE department\_id = e.department\_id

)

);

**2. More complex**

Sure, here's an example of a more complex query that uses both joins and subqueries in MSSQL:

SELECT

orders.order\_id,

orders.order\_date,

customers.customer\_name,

SUM(order\_details.unit\_price \* order\_details.quantity) AS total\_cost

FROM orders

INNER JOIN customers ON orders.customer\_id = customers.customer\_id

INNER JOIN (

SELECT order\_id, product\_id, unit\_price, quantity FROM order\_details WHERE unit\_price > 50

) AS order\_details ON orders.order\_id = order\_details.order\_id

GROUP BY orders.order\_id, orders.order\_date, customers.customer\_name

HAVING SUM(order\_details.unit\_price \* order\_details.quantity) > 1000

ORDER BY total\_cost DESC;

This query retrieves information about orders where the unit price of the products is over 50, grouped by order ID and customer name, with a total cost over 1000. The result set includes the order ID, order date, customer name, and total cost of the order.

The query uses an INNER JOIN to combine data from the **orders** and **customers** tables based on the **customer\_id** column. It also uses a subquery to retrieve only the relevant rows from the **order\_details** table where the unit price is over 50, which is then joined to the main query using an INNER JOIN on the **order\_id** column.

The GROUP BY clause groups the results by the order ID, order date, and customer name, while the HAVING clause filters the results to only include orders with a total cost over 1000. Finally, the ORDER BY clause sorts the results in descending order by total cost.

**3. More complex**

Here's another example of a more complex query that uses joins and subqueries in MSSQL:

SELECT

employees.employee\_id,

employees.first\_name,

employees.last\_name,

departments.department\_name, AVG(salaries.salary) AS avg\_salary

FROM employees

INNER JOIN departments ON employees.department\_id = departments.department\_id

INNER JOIN (

SELECT employee\_id, AVG(salary) AS salary

FROM salaries

GROUP BY employee\_id

) AS salaries ON employees.employee\_id = salaries.employee\_id

WHERE employees.hire\_date >= '2010-01-01'

GROUP BY employees.employee\_id, employees.first\_name, employees.last\_name, departments.department\_name

HAVING AVG(salaries.salary) > 50000

ORDER BY avg\_salary DESC;

This query retrieves information about employees who were hired on or after January 1st, 2010 and have an average salary over 50000, grouped by employee ID, first name, last name, and department name, and sorted in descending order by average salary.

The query uses an INNER JOIN to combine data from the **employees** and **departments** tables based on the **department\_id** column. It also uses a subquery to calculate the average salary for each employee in the **salaries** table, which is then joined to the main query using an INNER JOIN on the **employee\_id** column.

The WHERE clause filters the results to only include employees hired on or after January 1st, 2010. The GROUP BY clause groups the results by the employee ID, first name, last name, and department name, while the HAVING clause filters the results to only include employees with an average salary over 50000. Finally, the ORDER BY clause sorts the results in descending order by average salary.

**4. Last Complex example**

Here's another example of a more complex query that uses joins and subqueries in MSSQL:

WITH ranked\_orders AS (

SELECT

order\_id,

customer\_id,

order\_date,

order\_total,

ROW\_NUMBER() OVER(PARTITION BY customer\_id ORDER BY order\_total DESC) AS rank FROM orders ),

top\_orders AS ( SELECT order\_id, customer\_id, order\_date, order\_total FROM ranked\_orders WHERE rank = 1

)

SELECT

customers.customer\_id,

customers.customer\_name,

COUNT(top\_orders.order\_id) AS num\_top\_orders,

MAX(top\_orders.order\_total) AS max\_top\_order\_total

FROM customers

LEFT JOIN top\_orders ON customers.customer\_id = top\_orders.customer\_id

GROUP BY customers.customer\_id, customers.customer\_name;

This query retrieves information about customers and their top orders. It calculates the top order for each customer based on the order total and then counts the number of top orders and finds the maximum total for each customer.

The query uses two subqueries. The first subquery, **ranked\_orders**, uses the **ROW\_NUMBER()** function to assign a rank to each order based on the order total, partitioned by customer ID. The second subquery, **top\_orders**, selects the orders with a rank of 1 for each customer, effectively selecting the top order for each customer.

The query uses a LEFT JOIN to join the **customers** table with the **top\_orders** subquery, which may result in NULL values for customers who have not made any orders. The GROUP BY clause groups the results by customer ID and customer name, while the COUNT and MAX functions calculate the number of top orders and maximum top order total for each customer.