A **subquery** is like a story within a story. Imagine you're solving a puzzle, but to complete it, you first need to solve a smaller puzzle inside. That smaller puzzle is the subquery.

**Here's the breakdown:**

* A **subquery** is a query nested inside another query.
* It acts as a "helper query," providing data that the outer query needs to proceed.

Think of it like this:  
You’re trying to find out which employees earn more than the average salary in a company. To do this:

1. First, you calculate the **average salary** (this is the subquery).
2. Then, you use that result to identify employees with a salary greater than this average (the outer query).

**Example:**

SELECT Name, Salary

FROM Employees

WHERE Salary > (

SELECT AVG(Salary)

FROM Employees

);

**What’s Happening?**

1. The subquery (SELECT AVG(Salary) FROM Employees) calculates the average salary.
2. The outer query takes this result and compares it against each employee’s salary to filter the results.

**Why is it beautiful?**

* **Nested logic**: Subqueries let you break down complex problems into simpler steps, making queries easier to write and understand.
* **Modularity**: You can think of subqueries as building blocks, where each block has a clear purpose.
* **Elegance in action**: Subqueries make your concise and powerful by avoiding redundancy and tackling challenges step by step.

It's like having a skilled assistant working in the background to fetch the exact piece of information you need to complete your task.

subqueries come in various types, each serving a unique purpose. Here's an overview of the types, along with real-time examples to illustrate their practical applications:

**Dataset Assumption:**

**Employees Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EmployeeID** | **Name** | **Salary** | **DepartmentID** | **JobTitle** |
| **1** | **Alice** | **60000** | **101** | **Engineer** |
| **2** | **Bob** | **70000** | **102** | **Manager** |
| **3** | **Charlie** | **55000** | **101** | **Analyst** |
| **4** | **Diana** | **90000** | **103** | **Director** |
| **5** | **Edward** | **75000** | **102** | **Engineer** |

**Departments Table:**

|  |  |
| --- | --- |
| **DepartmentID** | **DepartmentName** |
| **101** | **IT** |
| **102** | **Sales** |
| **103** | **HR** |

**Sales Table:**

|  |  |  |
| --- | --- | --- |
| **SaleID** | **EmployeeID** | **SaleAmount** |
| **1** | **1** | **500** |
| **2** | **2** | **1200** |

**1. Single-Row Subquery**

* **Definition**: Returns one row with one value.
* **Use Case**: When you need a single result to compare or use in the outer query.

**Example**: Find employees who earn the same as the highest salary in the company.

SELECT Name, Salary

FROM Employees

WHERE Salary = (

SELECT MAX(Salary)

FROM Employees

);

**Subquery Output**:  
MAX(Salary) = **90000**

**Final Output**:

|  |  |
| --- | --- |
| **Name** | **Salary** |
| Diana | 90000 |

**2. Multi-Row Subquery**

* **Definition**: Returns multiple rows as output.
* **Use Case**: When you need to compare values from multiple rows.

**Example**: Find employees working in departments with IDs 101, 102, or 103.

SELECT Name, DepartmentID

FROM Employees

WHERE DepartmentID IN (

SELECT DepartmentID

FROM Departments

WHERE DepartmentID IN (101, 102, 103)

);

**Subquery Output**:

|  |
| --- |
| **DepartmentID** |
| 101 |
| 102 |
| 103 |

**Final Output**:

|  |  |
| --- | --- |
| **Name** | **DepartmentID** |
| Alice | 101 |
| Bob | 102 |
| Charlie | 101 |
| Diana | 103 |
| Edward | 102 |

**3. Multi-Column Subquery**

* **Definition**: Returns multiple columns as output.
* **Use Case**: When the subquery provides multiple values for comparison.

**Example**: Find employees whose department and job title match specific criteria.

SELECT Name, DepartmentID, JobTitle

FROM Employees

WHERE (DepartmentID, JobTitle) IN (

SELECT DepartmentID, JobTitle

FROM Employees

WHERE JobTitle = 'Engineer'

);

**Subquery Output**:

|  |  |
| --- | --- |
| **DepartmentID** | **JobTitle** |
| 101 | Engineer |
| 102 | Engineer |

**Final Output**:

|  |  |  |
| --- | --- | --- |
| **Name** | **DepartmentID** | **JobTitle** |
| Alice | 101 | Engineer |
| Edward | 102 | Engineer |

**4. Correlated Subquery**

* **Definition**: The subquery references a column from the outer query, creating a dependency.
* **Use Case**: When the subquery needs to be executed repeatedly for each row of the outer query.

**Example**: Find employees who earn more than the average salary of their respective departments.

SELECT Name, Salary, DepartmentID

FROM Employees E

WHERE Salary > (

SELECT AVG(Salary)

FROM Employees

WHERE DepartmentID = E.DepartmentID

);

**Subquery Outputs** (calculated for each department):

| **DepartmentID** | **AVG(Salary)** |
| --- | --- |
| 101 | 57500 |
| 102 | 72500 |
| 103 | 90000 |

**Final Output**:

**Final Output:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Salary** | **DepartmentID** |
| Alice | 60,000 | 101 |

**5. Scalar Subquery**

* **Definition**: A subquery that returns a single value (like a single-row subquery) and is used wherever a single value is expected.
* **Use Case**: Used in SELECT clauses or calculations.

**Example**: Show the total salary as a percentage of the highest salary.

SELECT Name, Salary,

(Salary / (SELECT MAX(Salary) FROM Employees) \* 100) AS SalaryPercentage

FROM Employees;

**Subquery Output**:  
MAX(Salary) = **90000**

**Final Output**:

|  |  |  |
| --- | --- | --- |
| **Name** | **Salary** | **SalaryPercentage** |
| Alice | 60000 | 66.67% |
| Bob | 70000 | 77.78% |
| Charlie | 55000 | 61.11% |
| Diana | 90000 | 100.00% |
| Edward | 75000 | 83.33% |

**6. Nested Subquery (For Jolly, little difficult, need practice)**

* **Definition**: A subquery inside another subquery.
* **Use Case**: For deeply complex queries requiring multiple levels of processing.

**Example**: Find employees working in the department with the highest average salary.

SELECT Name

FROM Employees

WHERE DepartmentID = (

SELECT DepartmentID

FROM (

SELECT DepartmentID, AVG(Salary) AS AvgSalary

FROM Employees

GROUP BY DepartmentID

ORDER BY AvgSalary DESC

LIMIT 1

) AS TopDepartment

);

**Inner Subquery Output**:

|  |  |
| --- | --- |
| **DepartmentID** | **AvgSalary** |
| 102 | 72500 |

**Final Output**:

|  |
| --- |
| **Name** |
| Bob |
| Edward |

**7. Exists Subquery**

* **Definition**: Checks if a subquery returns any rows, used with EXISTS.
* **Use Case**: To test the existence of certain records.

**Example**: Find employees who have made at least one sale.

SELECT Name

FROM Employees E

WHERE EXISTS (

SELECT 1

FROM Sales S

WHERE S.EmployeeID = E.EmployeeID

);

**Subquery Output**:  
Sales records exist for **Alice** (EmployeeID 1) and **Bob** (EmployeeID 2).

**Final Output**:

|  |
| --- |
| **Name** |
| Alice |
| Bob |

**8. NOT EXISTS Subquery**

* **Definition**: Checks if a subquery returns no rows.
* **Use Case**: To filter out records not meeting certain criteria.

**Example**: Find employees who haven’t made any sales.

SELECT Name

FROM Employees E

WHERE NOT EXISTS (

SELECT 1

FROM Sales S

WHERE S.EmployeeID = E.EmployeeID

);

**Subquery Output**:  
No sales records exist for **Charlie**, **Diana**, and **Edward**.

**Final Output**:

|  |
| --- |
| **Name** |
| Charlie |
| Diana |
| Edward |

With this data, managers can create tailored **performance improvement plans** for employees who aren't making sales.

**Summary of the Subquery Types (Including Non-Exists)**

|  |  |  |
| --- | --- | --- |
| Type | Description | Keyword |
| Single-Row Subquery | **Returns one row.** | **=, <, >** |
| Multi-Row Subquery | **Returns multiple rows.** | **IN, ANY, ALL** |
| Scalar Subquery | **Returns one value.** | **Used in SELECT** |
| Correlated Subquery | **References outer query.** | **Can be used anywhere** |
| Exists Subquery | **Checks if rows exist.** | **EXISTS** |
| Non-Exists Subquery | **Checks if no rows exist that satisfy the condition.** | **NOT EXISTS** |
| Nested Subqueries | **Subqueries within subqueries.** | **Combined keywords** |

some **practical use cases** of subqueries in real-world business scenarios:

**1. Finding Top Performers in a Department**

A company wants to find the top performers (employees) in each department based on their salary. Using a subquery, the company can find the highest salary in each department and then retrieve the employees who earn those top salaries.

**Scenario:** Find the employees in each department who have the highest salary.

SELECT Name, DepartmentID, Salary

FROM Employees E

WHERE Salary = (

SELECT MAX(Salary)

FROM Employees

WHERE DepartmentID = E.DepartmentID

);

**Use Case**:

* This could be used by HR to identify key talent or top performers for **recognition, bonuses, or promotions**.

**2. Calculating Average Sales by Product**

A retail company wants to know which products are above average in sales. Using a subquery, the company can calculate the average sales and find products that perform better than average.

**Scenario:** Find products whose sales are greater than the average sales across all products.

SELECT ProductName, SalesAmount

FROM Products

WHERE SalesAmount > (

SELECT AVG(SalesAmount)

FROM Products

);

**Use Case**:

* This is helpful for **marketing campaigns** to focus on high-performing products or **inventory management** to ensure bestsellers are stocked.

**3. Identifying Customers Who Haven't Made Purchases**

In e-commerce, businesses often need to identify customers who have registered but haven't made any purchases. A subquery can check if a customer has any purchase records and return those who haven't.

**Scenario:** Find customers who have not made any purchases.

SELECT Name

FROM Customers C

WHERE NOT EXISTS (

SELECT 1

FROM Orders O

WHERE O.CustomerID = C.CustomerID

);

**Use Case**:

* The company could target these customers with **promotional offers** or **reminder emails** to encourage them to make a purchase.

**4. Determining Employees Who Have Not Completed Required Training**

A company wants to track employees who haven't completed mandatory training courses. By using a subquery, the company can identify employees who are missing from the training records.

**Scenario:** Find employees who have not completed mandatory training.

SELECT Name

FROM Employees E

WHERE NOT EXISTS (

SELECT 1

FROM Training T

WHERE T.EmployeeID = E.EmployeeID

AND T.TrainingType = 'Mandatory'

);

**Employees Table:**

|  |  |  |
| --- | --- | --- |
| **EmployeeID** | **Name** | **Department** |
| 1 | Alice | IT |
| 2 | Bob | HR |
| 3 | Charlie | Marketing |
| 4 | Diana | IT |

**Training Table:**

|  |  |
| --- | --- |
| **EmployeeID** | **TrainingType** |
| 1 | Mandatory |
| 2 | Optional |
| 3 | Mandatory |

**Result:**

|  |
| --- |
| **Name** |
| Bob |
| Diana |

**Use Case**:

* This is useful for **HR departments** to ensure **compliance** with required certifications or trainings.

**5. Subquery for Filtering Records Based on Another Table's Data**

A sales company may need to get sales representatives who have achieved a certain sales target, but the target is stored in a separate table. A subquery can filter the representatives who meet this criterion.

**Scenario:** Find sales representatives whose sales exceed a specific target.

SELECT Name, SalesAmount

FROM SalesRepresentatives S

WHERE SalesAmount > (

SELECT SalesTarget

FROM SalesTargets

WHERE Department = 'North'

);

**Use Case**:

* This helps in recognizing high-performing employees and could be useful for **reward systems** or **bonus allocation**.

**6. Aggregating Data Across Multiple Levels**

A company needs to calculate the average salary of employees who are in departments with above-average salaries. A subquery can be used to first calculate the department's average salary, then aggregate based on that information.

**Scenario:** Find employees whose salary is greater than the average salary of their department, where the department's average salary is above the overall company average.

SELECT Name, DepartmentID, Salary

FROM Employees E

WHERE Salary > (

SELECT AVG(Salary)

FROM Employees

WHERE DepartmentID = E.DepartmentID

)

AND DepartmentID IN (

SELECT DepartmentID

FROM Employees

GROUP BY DepartmentID

HAVING AVG(Salary) > (SELECT AVG(Salary) FROM Employees)

);

**Use Case**:

* This could be used by HR to identify **top earners** in departments performing above company average, useful for **rewarding high performers**.

**7. Retrieving Data with Multiple Conditional Criteria**

A company wants to find out which employees have worked in a department for over a year and have received a performance rating above a certain threshold. Using a subquery, this can be done with a specific condition.

**Scenario:** Find employees who have been in the department for more than a year and have a performance rating above 8.

SELECT Name, DepartmentID, HireDate, PerformanceRating

FROM Employees E

WHERE HireDate < (

SELECT DATE\_SUB(CURRENT\_DATE, INTERVAL 1 YEAR)

)

AND PerformanceRating > 8;

**Use Case**:

* This is useful for **HR** or **managers** looking to identify employees for **promotion consideration**, **bonuses**, or **recognition**.

**8. Comparing Current Data with Historical Data**

A retail company wants to compare sales in the current month to the previous month. A subquery can fetch the previous month's sales and compare them.

**Scenario:** Find products with sales in the current month that exceed sales in the previous month.

SELECT ProductName, SalesAmount

FROM Sales S

WHERE SalesAmount > (

SELECT SalesAmount

FROM Sales

WHERE ProductID = S.ProductID

AND MONTH(SaleDate) = MONTH(CURRENT\_DATE) - 1

);

**Use Case**:

* This is useful for **marketing departments** to assess the **effectiveness of promotions** or **seasonal product demand**.

**9. Dynamic Data Retrieval Based on User Input**

A system might need to fetch records dynamically based on parameters set by the user. A subquery can be used to create dynamic conditions based on changing requirements.

**Scenario:** Retrieve employees who have worked more than the average tenure of employees in their department.

SELECT Name, HireDate

FROM Employees E

WHERE DATEDIFF(CURRENT\_DATE, HireDate) > (

SELECT AVG(DATEDIFF(CURRENT\_DATE, HireDate))

FROM Employees

WHERE DepartmentID = E.DepartmentID

);

**Use Case**:

* This could be useful for managers to identify **long-term employees** or those who are more **experienced** than their peers, for **mentoring** or **leadership opportunities**.

**Summary of Practical Uses:**

1. **Employee and performance tracking**: Identifying top performers, underperformers, and those needing training or performance improvement.
2. **Sales analysis**: Comparing current sales performance with historical data, identifying high-performing products, and assessing sales efficiency.
3. **Customer segmentation**: Targeting customers who have not made a purchase or those who have purchased specific products.
4. **Operational efficiency**: Detecting missing or incomplete records (e.g., uncompleted training or missed sales targets).
5. **Dynamic decision-making**: Providing tailored insights based on changing conditions, customer behavior, or employee performance.

Subqueries allow businesses to extract powerful insights by providing **flexibility, modularity, and efficiency** in querying relational databases.

Here are some **Multiple-Choice Questions (MCQs)** on **Subqueries** in :

**1. What is a subquery in ?**

a) A query that returns a single row  
b) A query that is nested inside another query  
c) A query that can only be used with SELECT statements  
d) A query that runs on the server only once

**Answer**: b) A query that is nested inside another query

**2. Which of the following is an example of a correlated subquery?**

a) SELECT Name FROM Employees WHERE Salary > (SELECT AVG(Salary) FROM Employees);  
b) SELECT Name FROM Employees E WHERE Salary > (SELECT MAX(Salary) FROM Employees);  
c) SELECT Name FROM Employees E WHERE Salary > (SELECT AVG(Salary) FROM Employees WHERE E.DepartmentID = DepartmentID);  
d) SELECT \* FROM Orders WHERE CustomerID IN (SELECT CustomerID FROM Customers);

**Answer**: c) SELECT Name FROM Employees E WHERE Salary > (SELECT AVG(Salary) FROM Employees WHERE E.DepartmentID = DepartmentID);

**3. Which clause cannot be used with a subquery?**

a) FROM  
b) WHERE  
c) HAVING  
d) GROUP BY

**Answer**: d) GROUP BY

SELECT DepartmentID, AVG(Salary)

FROM Employees

WHERE DepartmentID IN (SELECT DepartmentID FROM Employees WHERE Salary > 50000)

GROUP BY DepartmentID;

In this example:

* The subquery is used in the WHERE clause to filter DepartmentIDs based on a condition (Salary > 50000).
* The outer query groups the results by DepartmentID and calculates the average salary for each department.

**Correct Answer**: None of the options is entirely correct. Subqueries can be used with FROM, WHERE, HAVING, and GROUP BY clauses, depending on the context

**4. What type of subquery is executed once for the entire query and can reference columns from the outer query?**

a) Scalar subquery  
b) Correlated subquery  
c) Non-correlated subquery  
d) Inline view

**Answer**: c) Non-correlated subquery

**5. Which of the following statements is TRUE about subqueries?**

a) A subquery can return more than one column in the SELECT clause.  
b) A subquery cannot be used in the WHERE clause.  
c) Subqueries can only be used in UPDATE statements.  
d) A subquery can be used inside the SELECT, INSERT, UPDATE, and DELETE statements.

**Answer**: d) A subquery can be used inside the SELECT, INSERT, UPDATE, and DELETE statements.

**Example of an invalid subquery**:

SELECT Name FROM Employees WHERE DepartmentID IN (SELECT DepartmentID, Salary FROM Employees);

This would throw an error because the subquery is trying to return two columns (DepartmentID and Salary).

**6. Which keyword is used to return a boolean value from a subquery to check for existence?**

a) IN  
b) EXISTS  
c) LIKE  
d) ALL

**Answer**: b) EXISTS

**7. Which of the following is an example of using a subquery in the SELECT clause?**

a) SELECT Name, (SELECT AVG(Salary) FROM Employees) FROM Employees;  
b) SELECT Name FROM Employees WHERE DepartmentID = (SELECT MAX(DepartmentID) FROM Employees);  
c) SELECT \* FROM Employees WHERE Salary > (SELECT AVG(Salary) FROM Employees);  
d) SELECT DepartmentID FROM Employees WHERE DepartmentID = (SELECT DepartmentID FROM Employees WHERE Salary > 50000);

**Answer**: a) SELECT Name, (SELECT AVG(Salary) FROM Employees) FROM Employees;

**8. What is the purpose of a NOT EXISTS subquery?**

a) To check for records that match a condition  
b) To check for records that do not match a condition  
c) To find the first record in a subquery  
d) To count rows in a subquery

**Answer**: b) To check for records that do not match a condition

**9. Which of the following statements about subqueries is FALSE?**

a) A subquery can return more than one row of data.  
b) A subquery can return only a single value.  
c) A subquery can be nested inside another subquery.  
d) A subquery can only be used in the SELECT statement.

**Answer**: d) A subquery can only be used in the SELECT statement.

Subqueries in SQL can be used in **multiple clauses**, not just in the SELECT statement. Here are the places where subqueries can be used:

**SELECT Clause**: Subqueries can be used to return a value or calculate an aggregate. This is the most common place where subqueries are used.

SELECT Name,

(SELECT MAX(Salary) FROM Employees) AS Highest\_Salary

FROM Employees;

**FROM Clause**: A subquery can be used as a table, often referred to as an **inline view** or **derived table**. This allows the subquery to act as a temporary table for the outer query.

SELECT D.Department, AVG(E.Salary)

FROM (SELECT \* FROM Employees WHERE Salary > 50000) E

JOIN Departments D ON E.Department = D.Department

GROUP BY D.Department;

**Assumed Data**

Let's assume the Employees and Departments tables have the following data:

**Employees Table:**

| **EmployeeID** | **Name** | **Department** | **Salary** |
| --- | --- | --- | --- |
| 1 | Alice | HR | 50000 |
| 2 | Bob | IT | 70000 |
| 3 | Charlie | HR | 60000 |
| 4 | Diana | IT | 80000 |
| 5 | Eve | Sales | 55000 |
| 6 | Frank | Sales | 65000 |

**Departments Table:**

| **Department** | **Manager** |
| --- | --- |
| HR | Alice |
| IT | Diana |
| Sales | Eve |

**Step-by-Step Execution:**

1. **Subquery**: The subquery filters employees with salaries greater than 50,000:

| **EmployeeID** | **Name** | **Department** | **Salary** |
| --- | --- | --- | --- |
| 2 | Bob | IT | 70000 |
| 3 | Charlie | HR | 60000 |
| 4 | Diana | IT | 80000 |
| 5 | Eve | Sales | 55000 |
| 6 | Frank | Sales | 65000 |

1. **Join**: The subquery's result (E) is joined with the Departments table (D) on the Department column:

| **EmployeeID** | **Name** | **Department** | **Salary** | **Manager** |
| --- | --- | --- | --- | --- |
| 2 | Bob | IT | 70000 | Diana |
| 3 | Charlie | HR | 60000 | Alice |
| 4 | Diana | IT | 80000 | Diana |
| 5 | Eve | Sales | 55000 | Eve |
| 6 | Frank | Sales | 65000 | Eve |

1. **Grouping**: The results are grouped by Department, and the average salary for each department is calculated:

**HR**: Average salary = (60000) = 60,000

**IT**: Average salary = (70000 + 80000) / 2 = 75,000

**Sales**: Average salary = (55000 + 65000) / 2 = 60,000

**Final Output:**

| **Department** | **AVG(Salary)** |
| --- | --- |
| HR | 60000 |
| IT | 75000 |
| Sales | 60000 |

**WHERE Clause**: Subqueries can be used in conditions (e.g., =, IN, EXISTS, NOT EXISTS, etc.) to filter results based on the output of another query.

SELECT Name

FROM Employees

WHERE Department IN (

SELECT Department

FROM Departments

WHERE Manager = 'Alice'

);

**HAVING Clause**: Subqueries can be used with aggregate functions in the HAVING clause to filter groups.

SELECT Department, AVG(Salary)

FROM Employees

GROUP BY Department

HAVING AVG(Salary) > (SELECT AVG(Salary) FROM Employees);

**INSERT, UPDATE, and DELETE Statements**: Subqueries can also be used in INSERT, UPDATE, and DELETE statements, typically for inserting data based on another query, updating values based on conditions, or deleting records based on another query.

**INSERT with Subquery**:

INSERT INTO Employees (Name, Department, Salary)

SELECT 'John', 'IT', 70000

WHERE NOT EXISTS (

SELECT 1

FROM Employees

WHERE Name = 'John'

);

**UPDATE with Subquery**:

UPDATE Employees

SET Salary = (SELECT MAX(Salary) FROM Employees WHERE Department = 'IT')

WHERE Department = 'HR';

**DELETE with Subquery**:

DELETE FROM Employees

WHERE EmployeeID NOT IN (

SELECT EmployeeID

FROM ActiveEmployees

);

**Conclusion:**

Subqueries are not limited to the SELECT statement; they can be used in various SQL clauses such as WHERE, HAVING, FROM, INSERT, UPDATE, and DELETE. This flexibility makes subqueries a powerful tool for complex queries and data manipulation tasks.

**10. Which of the following subquery types is used when a query needs to be evaluated for each row processed by the outer query?**

a) Scalar subquery  
b) Correlated subquery  
c) Non-correlated subquery  
d) Join subquery

**Answer**: b) Correlated subquery

**11. Which of the following queries returns employees who have the highest salary in each department using a subquery?**

a)

SELECT Name, DepartmentID, Salary

FROM Employees

WHERE Salary = (SELECT MAX(Salary) FROM Employees WHERE DepartmentID = Employees.DepartmentID);

b)

SELECT Name, DepartmentID, Salary

FROM Employees

WHERE Salary = (SELECT MIN(Salary) FROM Employees WHERE DepartmentID = Employees.DepartmentID);

c)

SELECT Name, DepartmentID, Salary

FROM Employees

WHERE Salary < (SELECT MAX(Salary) FROM Employees WHERE DepartmentID = Employees.DepartmentID);

d)

SELECT Name, DepartmentID, Salary

FROM Employees

WHERE Salary > (SELECT MAX(Salary) FROM Employees WHERE DepartmentID = Employees.DepartmentID);

**Answer**: a)

SELECT Name, DepartmentID, Salary

FROM Employees

WHERE Salary = (SELECT MAX(Salary) FROM Employees WHERE DepartmentID = Employees.DepartmentID);

**12. Which of the following is a valid subquery type in ?**

a) Scalar subquery  
b) List subquery  
c) Aggregate subquery  
d) View subquery

**Answer**: a) Scalar subquery

**13. In a correlated subquery, the outer query can refer to which of the following?**

a) Only the columns from the subquery  
b) Only the columns from the outer query  
c) Both the columns from the subquery and the outer query  
d) Neither the columns from the subquery nor the outer query

**Answer**: c) Both the columns from the subquery and the outer query