

Chapter 14: MySQL Subqueries

14.1 Introduction to Subqueries

- * A **subquery (or inner query)** is a query nested inside another SQL statement.
- * The subquery is **executed first**, and its result is used by the outer query.
- * Subqueries can be used with **SELECT, INSERT, UPDATE, and DELETE** statements.
- * They help in solving **complex problems** in a simple and efficient way.

Key Points :

- ✓ A subquery is a query inside another query.
- ✓ The inner query runs first, then the outer query uses its result.
- ✓ Subqueries make SQL statements more powerful and flexible.

14.2 Scalar Subqueries

- * A scalar subquery returns **exactly one value** (one row, one column).
- * It can be used in **SELECT, WHERE, HAVING**, and other clauses.
- * The scalar subquery is used with comparison operators like **=, >, <, >=, <=, <>**.

Syntax :

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator (SELECT column_name
                             FROM table_name WHERE condition);
```

Example 1: Retrieve the name of the employee(s) who have the highest salary.

Query :

```
SELECT name
FROM employees
WHERE salary = (SELECT MAX(salary)
                FROM employees);
```

Explanation :

The **inner query** (SELECT MAX(salary) FROM employees) returns the highest salary. The outer query fetches the employee name(s) whose salary is equal to that value.

Example 2: Retrieve employees who earn more than the average salary.

Query :

```
SELECT name, salary
FROM employees
WHERE salary > (SELECT AVG(salary)
                FROM employees);
```

Explanation :

The **inner query** returns the average salary. The outer query gets all employees whose salary is greater than that average.

Key Takeaway :



Scalar subqueries return a **single value** and are used to perform comparisons in SQL queries, making queries **concise** and **powerful**.



14.3 Single-Row Subqueries

- * A single-row subquery returns **zero or one row**.
- * It is used with comparison operators like **=**, **>**, **<**, **>=**, **<=**, **<>**.
- * It is often used with aggregate functions like **MAX()**, **MIN()**, **COUNT()**, **SUM()**, **AVG()** that return a single value.
- * If the subquery returns more than one row, an error occurs.



Example 1: Find the employee(s) with the minimum salary.

Query :

```
SELECT name, salary
FROM employees
WHERE salary = (SELECT MIN(salary)
                FROM employees);
```

Explanation :

The inner query (**SELECT MIN(salary) FROM employees**) returns the minimum salary. The outer query gets the employee(s) whose salary is equal to that minimum value.



Example 2: List employees who earn more than the highest salary in the 'Sales' department.

Query :

```
SELECT name, salary
FROM employees
WHERE salary > (SELECT MAX(salary)
                FROM employees
                WHERE department = 'Sales');
```

Explanation :

The inner query returns the highest salary from the Sales department. The outer query gets all employees whose salary is greater than that value.



Key Points :

- ✓ Single-row subqueries return zero or one row.
- ✓ They are used with comparison operators.
- ✓ They are very useful with aggregate functions.



14.4 Multiple-Row Subqueries

- * A multiple-row subquery returns **more than one row**.
- * It is used with operators like **IN**, **NOT IN**, **ANY**, **ALL**.
- * These operators allow comparisons with a list of values.
- * Multiple-row subqueries are very common and powerful.



Example 1: List employees who belong to the departments located in 'New York'.

Query :

```
SELECT name, department
FROM employees
WHERE department IN (SELECT department
                    FROM departments
                    WHERE location = 'New York');
```

Explanation :

The inner query returns all departments located in New York. The outer query gets employees whose department is in that list.



Example 2: Find employees whose salary is greater than every salary in the 'Sales' department.

Query :

```
SELECT name, salary
FROM employees
WHERE salary > ALL (SELECT salary
                   FROM employees
                   WHERE department = 'Sales');
```

Explanation :

The inner query returns all salaries from Sales. The outer query gets employees whose salary is greater than all those salaries.



Key Points :

- ✓ Multiple-row subqueries return more than one row.
- ✓ They are used with **IN**, **NOT IN**, **ANY**, **ALL** operators.
- ✓ They allow comparisons with a list of values.



14.5 Multiple-Column Subqueries

- * A multiple-column subquery returns **multiple columns**.
- * It is used with comparison operators like **IN**.
- * The number of columns in the subquery must match the number of columns in the outer query.
- * It is useful when the result needs to match a combination of values.



Example 1: List employees who work in the same department and location as 'John Doe'.

Query :

```
SELECT name, department, location
FROM employees
WHERE (department, location) IN
      (SELECT department, location
       FROM employees
       WHERE name = 'John Doe');
```

Explanation :

The inner query returns the department and location of 'John Doe'. The outer query gets all employees whose department and location match that combination.



Example 2: List employees who have the same (manager_id, department) as the employee with id 101.

Query :

```
SELECT name, manager_id, department
FROM employees
WHERE (manager_id, department) IN
      (SELECT manager_id, department
       FROM employees
       WHERE id = 101);
```

Explanation :

The inner query returns the manager_id and department of employee with id 101. The outer query gets employees having the same manager and belonging to the same department.



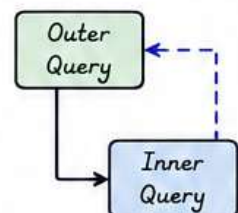
Key Points :

- ✓ Multiple-column subqueries return more than one column.
- ✓ They are used with the IN operator.
- ✓ The number of columns must be the same in both queries.



14.6 Correlated Subqueries

- * A correlated subquery is a subquery that **depends on the outer query**.
- * It uses values from the outer query in its **WHERE** clause.
- * The subquery is executed **once for each row** processed by the outer query.
- * It is used with comparison operators like **=, >, <, >=, <=, <>**.
- * It is powerful for row-by-row comparisons.



Example 1: List employees who earn more than the average salary of their own department.

Query :

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

Explanation :

The inner query calculates the average salary for the department of the current employee (e.department). The outer query gets employees whose salary is greater than that department's average.



Example 2: List employees who earn more than their manager.

Query :

```
SELECT e.name, e.salary, m.name AS manager
FROM employees e
WHERE e.manager_id = m.id
      AND e.salary > (SELECT salary
                       FROM employees m
                       WHERE m.id = e.manager_id);
```

Explanation :

The inner query gets the salary of the manager for the current employee (using e.manager_id). The outer query lists employees whose salary is greater than their manager's salary.



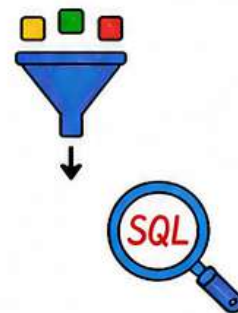
Key Points :

- ✓ Correlated subqueries depend on the outer query.
- ✓ They are executed once per row of the outer query.
- ✓ They are very useful for row-by-row comparisons and complex conditions.



14.7 Subqueries with WHERE

- * Subqueries with WHERE are used to **filter rows** based on the result of a subquery.
- * They are used with comparison operators and **IN, NOT IN, ANY, ALL**.
- * The subquery can return **one value, multiple values** or **multiple rows**.
- * These subqueries help in applying **complex conditions** in the **WHERE** clause.



Example 1: List employees who belong to the 'HR' department.

Query:

```
SELECT name, department
FROM employees
WHERE department = (SELECT department
                    FROM departments
                    WHERE dept_name = 'HR');
```

Explanation:

The subquery returns the department name where dept_name is 'HR'. The outer query gets employees whose department is equal to that value.



Example 2: List employees whose salary is greater than the average salary.

Query:

```
SELECT name, salary
FROM employees
WHERE salary > (SELECT AVG(salary)
                FROM employees);
```

Explanation:

The subquery calculates the average salary of all employees. The outer query gets employees whose salary is greater than that average.



Example 3: List employees who are not in the 'Sales' department.

Query:

```
SELECT name, department
FROM employees
WHERE department NOT IN (SELECT department
                        FROM departments
                        WHERE dept_name = 'Sales');
```

Explanation:

The subquery returns the department name 'Sales'. The outer query gets employees whose department is not in that list.



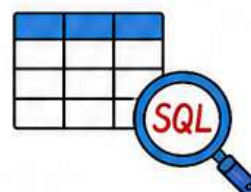
Key Points:

- ✓ Subqueries with WHERE are used to filter rows.
- ✓ They are used with comparison operators and **IN, NOT IN, ANY, ALL**.
- ✓ They help in applying complex conditions on data.



14.8 Subqueries with FROM

- * Subqueries with FROM are used as **derived tables (temporary tables)**.
- * The subquery in FROM acts like a table in the outer query.
- * It helps in **summarizing data** before applying conditions.
- * An **alias** must be given to the subquery.



Example 1: Find the average salary in each department.

Query:

```
SELECT department, avg_salary
FROM (SELECT department, AVG(salary) AS avg_salary
      FROM employees
      GROUP BY department) AS dept_avg;
```

Explanation:

The subquery calculates the average salary for each department. The outer query displays the results from that derived table.



Example 2: List departments where the average salary is greater than 50000.

Query:

```
SELECT department, avg_salary
FROM (SELECT department, AVG(salary) AS avg_salary
      FROM employees
      GROUP BY department) AS dept_avg
WHERE avg_salary > 50000;
```

Explanation:

The subquery computes the average salary for each department. The outer query filters departments where the average salary is greater than 50000.



Key Points:

- ✓ Subqueries with FROM are used as derived tables.
- ✓ They must have an alias.
- ✓ They help in summarizing data before filtering or joining.



14.9 Subqueries with SELECT

- * Subqueries with SELECT are used to return **calculated data** or values as part of the result.
- * The subquery can return **one value**, **multiple values** or **multiple columns**.
- * They are useful in **reports**, **analysis** and **data comparisons**.
- * An **alias** must be given to the subquery.



Example 1: Show each employee along with the average salary of their department.

Query:

```
SELECT e.name, e.department, d.avg_salary
FROM employees e,
     (SELECT department, AVG(salary) AS avg_salary
      FROM employees
      GROUP BY department) AS d
WHERE e.department = d.department;
```

Explanation:

The subquery calculates the average salary for each department. The outer query joins employees with the derived table (subquery result) using department.



Example 2: Show the total salary payable for each department along with the number of employees.

Query:

```
SELECT d.department, d.total_salary, d.emp_count
FROM (SELECT department,
      SUM(salary) AS total_salary,
      COUNT(*) AS emp_count
      FROM employees
      GROUP BY department) AS d;
```

Explanation:

The subquery computes total salary and employee count for each department. The outer query simply displays the results from the subquery.



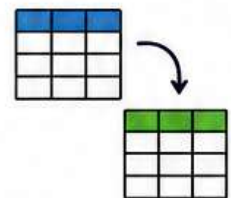
Key Points:

- ✓ Subqueries with SELECT return calculated or derived data.
- ✓ They can return one value, multiple values or multiple columns.
- ✓ An alias is required for the subquery.
- ✓ They are very useful in reporting and data analysis.



14.10 Subqueries with INSERT

- * Subqueries with INSERT are used to **insert** data into a table using the result of a subquery.
- * The number of columns in the INSERT table must match the number of columns returned by the subquery.
- * It helps in **copying**, **archiving** or **moving data** between tables.
- * The data types of both must be compatible.



Example 1: Insert all employees from the 'Sales' department into the sales_backup table.

Query:

```
INSERT INTO sales_backup (id, name, department, salary)
SELECT id, name, department, salary
FROM employees
WHERE department = 'Sales';
```

Explanation:

The subquery selects all columns of employees from the Sales department. The result is inserted into the sales_backup table.



Example 2: Insert the highest paid employee from each department into the top_earners table.

Query:

```
INSERT INTO top_earners (department, name, salary)
SELECT department, name, salary
FROM employees e1
WHERE salary = (SELECT MAX(salary)
                FROM employees e2
                WHERE e1.department = e2.department);
```

Explanation:

The subquery finds the maximum salary in each department. The outer query gets employees who earn that maximum salary and inserts them into top_earners.



Key Points:

- ✓ Subqueries with INSERT are used to insert data returned by a subquery.
- ✓ The number of columns must match in both INSERT table and subquery.
- ✓ They are useful for copying, archiving or transferring data.
- ✓ Data types in both must be compatible.



14.11 Subqueries with UPDATE

- * Subqueries with UPDATE are used to **update existing data** using values returned by a subquery.
- * The subquery can return **one value** or **multiple values**.
- * The subquery is usually used in the **SET** or **WHERE** clause.
- * It helps in **modifying data** based on conditions or calculations.
- * The number of columns and data types must be compatible.

Example 1: Increase the salary of all employees in the 'Sales' department by 10%.

Query:

```
UPDATE employees
SET salary = salary * 1.10
WHERE department = (SELECT department
                    FROM departments
                    WHERE dept_name = 'Sales');
```

Explanation:

The subquery returns the department name 'Sales'. The outer query updates the salary of all employees whose department is equal to that value by increasing it by 10%.



Example 2: Set the salary of all employees to the average salary of their respective department.

Query:

```
UPDATE employees e
SET salary = (SELECT AVG(salary)
              FROM employees
              WHERE department = e.department);
```

Explanation:

The subquery calculates the average salary for the department of each employee (correlated by e.department). The outer query updates each employee's salary with that department's average.



Key Points:

- ✓ Subqueries with UPDATE are used to modify existing data.
- ✓ They can return one value or multiple values.
- ✓ They are placed in SET or WHERE clause.
- ✓ The number of columns and data types must be compatible.
- ✓ They are useful for complex updates based on conditions or calculations.



14.12 Subqueries with DELETE

- * Subqueries with DELETE are used to **remove records** from a table using the result of a subquery.
- * The subquery can return **one value** or **multiple values**.
- * The subquery is usually used in the **WHERE** clause.
- * It helps in **deleting data** based on conditions.
- * The data types in both must be compatible.

Example 1: Delete all employees who do not belong to any department.

Query:

```
DELETE FROM employees
WHERE department_id NOT IN
  (SELECT department_id
   FROM departments);
```

Explanation:

The subquery returns all department_ids from departments. The outer query deletes employees whose department_id is not present in that list.



Example 2: Delete employees whose salary is less than the average salary of the entire company.

Query:

```
DELETE FROM employees
WHERE salary < (SELECT AVG(salary)
                FROM employees);
```

Explanation:

The subquery calculates the average salary of all employees. The outer query deletes employees whose salary is less than that average.



Key Points:

- ✓ Subqueries with DELETE are used to remove records.
- ✓ They are placed only in the WHERE clause.
- ✓ They can return one value or multiple values.
- ✓ They are very useful in removing unwanted or condition-based data.



14.13 EXISTS

- * EXISTS is used to test whether a subquery returns **any rows**.
- * It returns **TRUE** if the subquery returns at least one row, otherwise **FALSE**.
- * The subquery can return **one or more columns**.
- * EXISTS is often faster than using IN for large datasets.



Example 1: List employees who have at least one dependent.

Query :

```
SELECT name
FROM employees e
WHERE EXISTS (SELECT 1
              FROM dependents d
              WHERE d.emp_id = e.id);
```

Explanation :

The subquery checks if there exists at least one dependent for the employee (i.e., $d.emp_id = e.id$).
The outer query returns employees for whom this condition is TRUE.



Example 2: List departments that have at least one employee.

Query :

```
SELECT department_name
FROM departments d
WHERE EXISTS (SELECT 1
              FROM employees e
              WHERE e.department_id = d.id);
```

Explanation :

The subquery checks if there exists at least one employee in the department (i.e., $e.department_id = d.id$).
The outer query returns departments that satisfy this condition.



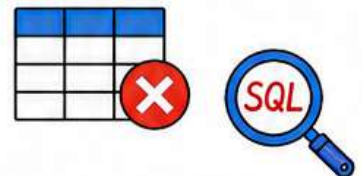
Key Points :

- ✓ EXISTS returns TRUE if the subquery returns at least one row.
- ✓ It can return one or more columns.
- ✓ It is used to check for the presence of related data.
- ✓ It is often more efficient than IN with large result sets.



14.14 NOT EXISTS

- * NOT EXISTS is used to test whether a subquery returns **no rows**.
- * It returns **TRUE** if the subquery returns zero rows, otherwise **FALSE**.
- * The subquery can return **one or more columns**.
- * It is used to find records that do not have matching data.



Example 1: List employees who do not have any dependent.

Query :

```
SELECT name
FROM employees e
WHERE NOT EXISTS (SELECT 1
                  FROM dependents d
                  WHERE d.emp_id = e.id);
```

Explanation :

The subquery checks if there does not exist any dependent for the employee.
The outer query returns employees for whom no dependent is found.



Example 2: List departments that do not have any employees.

Query :

```
SELECT department_name
FROM departments d
WHERE NOT EXISTS (SELECT 1
                  FROM employees e
                  WHERE e.department_id = d.id);
```

Explanation :

The subquery checks if there does not exist any employee in the department.
The outer query returns departments with no employees.



Key Points :

- ✓ NOT EXISTS returns TRUE if the subquery returns zero rows.
- ✓ It can return one or more columns.
- ✓ It is used to find records with no matching data.
- ✓ It is often more efficient than NOT IN, especially with NULL values.



14.15 ANY

- * ANY is used with comparison operators to compare a value with **any** one value returned by a subquery.
- * It returns **TRUE** if the condition is TRUE for at least one value, otherwise **FALSE**.
- * It is similar to the **OR** condition.
- * ANY can be used with operators like **=, >, <, >=, <=, <>**.
- * It works well with **multiple-row subqueries**.

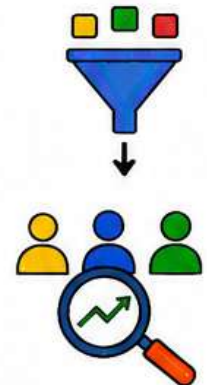
Example 1 : List employees whose salary is greater than any salary in the 'Sales' department.

Query :

```
SELECT name, salary
FROM employees
WHERE salary > ANY
      (SELECT salary
       FROM employees
       WHERE department = 'Sales');
```

Explanation :

The subquery returns all salaries of employees in the Sales department. The outer query returns employees whose salary is greater than at least one of those salaries.



Example 2 : List employees whose salary is equal to any salary in the 'HR' department.

Query :

```
SELECT name, salary
FROM employees
WHERE salary = ANY
      (SELECT salary
       FROM employees
       WHERE department = 'HR');
```

Explanation :

The subquery returns all salaries of employees in the HR department. The outer query returns employees whose salary matches any one of those salaries.



Key Points :

- ✓ ANY returns TRUE if the condition is TRUE for at least one value.
- ✓ It is used with comparison operators.
- ✓ It is similar to the OR condition.
- ✓ It works with multiple-row subqueries.
- ✓ It returns one or more columns.



14.16 ALL

- * ALL is used with comparison operators to compare a value with **all** values returned by a subquery.
- * It returns **TRUE** if the condition is TRUE for **every** value, otherwise **FALSE**.
- * It is similar to the **AND** condition.
- * ALL can be used with operators like **=, >, <, >=, <=, <>**.
- * It is used when the condition must hold for **all values**.

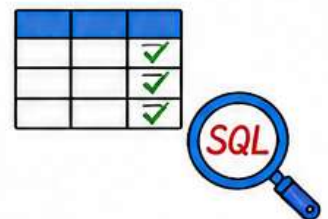
Example 1 : List employees whose salary is greater than all salaries in the 'Sales' department.

Query :

```
SELECT name, salary
FROM employees
WHERE salary > ALL
      (SELECT salary
       FROM employees
       WHERE department = 'Sales');
```

Explanation :

The subquery returns all salaries of employees in the Sales department. The outer query returns employees whose salary is greater than every salary in the Sales department.



Example 2 : List employees whose salary is less than or equal to all salaries in the 'HR' department.

Query :

```
SELECT name, salary
FROM employees
WHERE salary <= ALL
      (SELECT salary
       FROM employees
       WHERE department = 'HR');
```

Explanation :

The subquery returns all salaries of employees in the HR department. The outer query returns employees whose salary is less than or equal to every salary in the HR department.



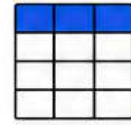
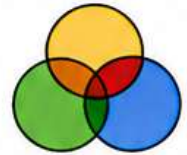
Key Points :

- ✓ ALL returns TRUE if the condition is TRUE for every value.
- ✓ It is used with comparison operators.
- ✓ It is similar to the AND condition.
- ✓ It works with multiple-row subqueries.
- ✓ It returns one or more columns.



15.7 Practical Set Operation Queries

- * Set operations allow us to **combine** the result sets of two or more SELECT statements.
- * They are very useful in **real-world applications**.
- * The columns in all SELECT statements must be the **same in number, data type and order**.
- * Duplicates are **removed** in UNION, INTERSECT and EXCEPT (except UNION ALL).



Example 1 : List all employees who work in either 'Sales' or 'HR' departments.

Query :

```
SELECT name, department
FROM employees
WHERE department = 'Sales'
UNION
SELECT name, department
FROM employees
WHERE department = 'HR';
```

Result

name	department
Alice	Sales
Bob	Sales
Cathy	Sales
David	HR
Eva	HR
Frank	HR

Explanation :

This query returns all employees who belong to either Sales or HR department. Duplicates (if any) are removed automatically by **UNION**.



Example 2 : List employees who work in both 'Sales' and 'HR' departments.

Query :

```
SELECT name, department
FROM employees
WHERE department = 'Sales'
INTERSECT
SELECT name, department
FROM employees
WHERE department = 'HR';
```

Result

name	department
David	HR

(if David is also in Sales)

Explanation :

This query returns only those employees who are present in both Sales and HR departments.



Example 3 : List employees who work in 'Sales' but not in 'HR'.

Query :

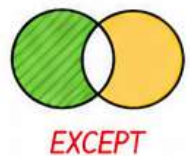
```
SELECT name, department
FROM employees
WHERE department = 'Sales'
EXCEPT
SELECT name, department
FROM employees
WHERE department = 'HR';
```

Result

name	department
Alice	Sales
Bob	Sales
Cathy	Sales

Explanation :

This query returns employees who are in Sales department but not in HR department.



Example 4 : List all unique departments from Sales and Finance.

Query :

```
SELECT department
FROM employees
WHERE department = 'Sales'
UNION
SELECT department
FROM employees
WHERE department = 'Finance';
```

Result

department
Sales
Finance

Explanation :

This query returns unique department names from Sales and Finance.



Key Points :

- ✓ UNION combines results and removes duplicates.
- ✓ UNION ALL combines results and includes duplicates.
- ✓ INTERSECT returns common rows from both queries.
- ✓ EXCEPT returns rows from the first query that are not in the second.
- ✓ Column count, data type and order must be the same in all SELECT statements.
- ✓ ORDER BY can be applied only once at the end of the entire set operation.

