Lab 06: Working with SUBQUERIES

Objective(s):

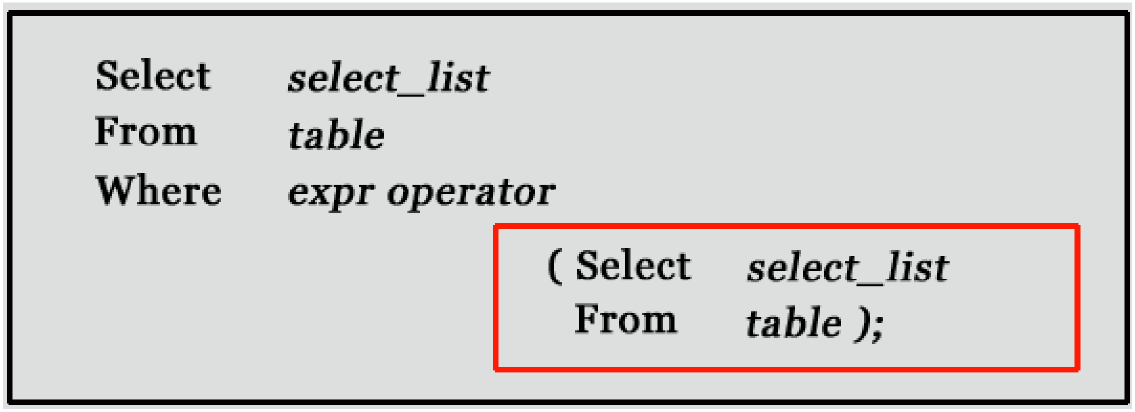
1. Introduction of SUBQUERIES
2. Types of SUBQUERIES

1: Introduction of SUBQUERIES

A subquery is a SQL query nested inside a larger query.

- A subquery may occur in:
  - A SELECT clause
  - A FROM clause
  - A WHERE clause
- In MySQL subquery can be nested inside a SELECT, INSERT, UPDATE, DELETE, SET, or DO statement or inside another subquery.
- A subquery is usually added within the WHERE Clause of another SQL SELECT statement.
- You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, SOME, or ALL.
- A subquery can be treated as an inner query, which is a SQL query placed as a part of another query is called as outer query.
- The inner query executes first before its parent query so that the results of the inner query can be passed to the outer query.

Syntax:



```
Select      select_list
From        table
Where       expr operator
                        ( Select   select_list
                          From     table );
```

The subquery (inner query) executes once before the main query (outer query) executes.

The main query (outer query) use the subquery result.

Subquery syntax as specified by the SQL standard and supported in MySQL.

A subquery can return a scalar (a single value), a single row, a single column, or a table (one or more rows of one or more columns). These are called scalar, column, row, and table subqueries.

Subqueries: Guidelines

There are some guidelines to consider when using subqueries:

- A subquery must be enclosed in parentheses.
- Use single-row operators with single-row subqueries, and use multiple-row operators with multiple-row subqueries.
- If a subquery (inner query) returns a null value to the outer query, the outer query will not return any rows when using certain comparison operators in a WHERE clause.

2: Types of SUBQUERIES

1. The Subquery as Scalar Operand
2. Comparisons using Subqueries
3. Subqueries with ALL, ANY, IN, or SOME
4. Row Subqueries
5. Subqueries with EXISTS or NOT EXISTS
6. Correlated Subqueries
7. Subqueries in the FROM Clause

MySQL Subquery as Scalar Operand

A scalar subquery is a subquery that returns exactly one column value from one row. A scalar subquery is a simple operand, and you can use it almost anywhere a single column value or literal is legal. If the subquery returns 0 rows then the value of scalar subquery expression in NULL and if the subquery returns more than one row then MySQL returns an error.

There is some situation where a scalar subquery cannot be used. If a statement permits only a literal value, you cannot use a subquery. For example, LIMIT requires literal integer arguments, and LOAD DATA INFILE requires a literal string file name. You cannot use subqueries to supply these values.

Comparisons using Subqueries

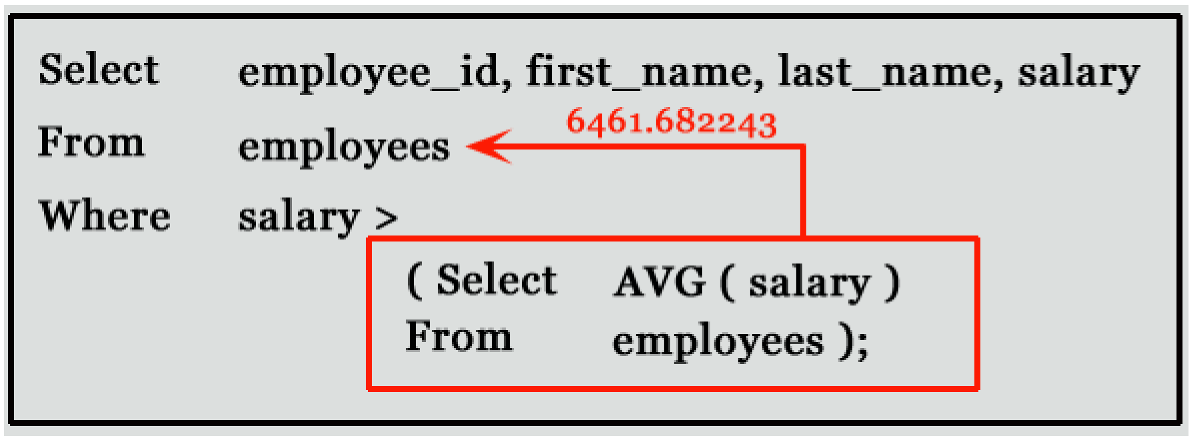
A subquery can be used before or after any of the comparison operators. The subquery can return at most one value. The value can be the result of an arithmetic expression or a column function. SQL then compares the value that results from the subquery with the value on the other side of the comparison operator. You can use the following comparison operators:

Operator	**Description**
=	Equal to
>	Greater than

>=	Greater than or equal to
<	Less than
<=	Less than or equal to
!=	Not equal to
<>	Not equal to
<=>	NULL-safe equal to operator

MySQL **null safe equal** (<=>) to operator performs an equality comparison like the equal to (=) operator, but returns 1 rather than NULL if both operands are NULL, and 0 rather than NULL if one operand is NULL.

For example, suppose you want to find the employee id, first_name, last_name, and salaries for employees whose average salary is higher than the average salary throughout the company.



```
Select    employee_id, first_name, last_name, salary
From      employees
Where     salary >
              ( Select    AVG ( salary )
                From      employees );
```

MySQL Subqueries with ALL, ANY, IN, or SOME

You can use a subquery after a comparison operator, followed by the keyword ALL, ANY, or SOME.

The ALL operator compares value to every value returned by the subquery. Therefore ALL operator (which must follow a comparison operator) returns TRUE if the comparison is TRUE for ALL of the values in the column that the subquery returns.

The ALL operator returns true if all of the subquery values meet the condition.

ALL Syntax:

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

NOT IN is an alias for <> ALL. Thus, these two statements are the same:

```
SELECT c1 FROM t1 WHERE c1 <> ALL (SELECT c1 FROM t2);
SELECT c1 FROM t1 WHERE c1 NOT IN (SELECT c1 FROM t2);
```

The following query selects the department with the highest average salary. The subquery finds the average salary for each department, and then the main query selects the department with the highest average salary.

```
SELECT department_id, AVG(SALARY)
FROM EMPLOYEES GROUP BY department_id
HAVING AVG(SALARY)>=ALL
(SELECT AVG(SALARY) FROM EMPLOYEES GROUP BY department_id);
```

Note: Here we have used ALL keyword for this subquery as the department selected by the query must have an average salary greater than or equal to all the average salaries of the other departments.

The **ANY** operator compares the value to each value returned by the subquery. Therefore ANY keyword (which must follow a comparison operator) returns TRUE if the comparison is TRUE for ANY of the values in the column that the subquery returns.

The ANY operator returns true if any of the subquery values meet the condition.

ANY Syntax:

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

The following query selects any employee who works in the location 1700. The subquery finds the department id in the 1700 location, and then the main query selects the employees who work in any of these departments.

```
SELECT first_name, last_name,department_id
FROM employees WHERE department_id= ANY
(SELECT DEPARTMENT_ID FROM departments WHERE location_id=1700);
```

Note: We have used ANY keyword in this query because it is likely that the subquery will find more than one departments in 1700 location. If you use the ALL keyword instead of the ANY keyword, no data is selected because no employee works in all departments of 1700 location

When used with a subquery, the word IN (equal to any member of the list) is an alias for = ANY. Thus, the following two statements are the same:

```
SELECT c1 FROM t1 WHERE c1 = ANY (SELECT c1 FROM t2);
SELECT c1 FROM t1 WHERE c1 IN (SELECT c1 FROM t2);
```

The word SOME is an alias for ANY. Thus, these two statements are the same:

```
SELECT c1 FROM t1 WHERE c1 <> ANY (SELECT c1 FROM t2);
SELECT c1 FROM t1 WHERE c1 <> SOME (SELECT c1 FROM t2);
```

Row Subqueries

A row subquery is a subquery that returns a single row and more than one column value. You can use = , >, <, >=, <=, <>, !=, <=> comparison operators. See the following examples:

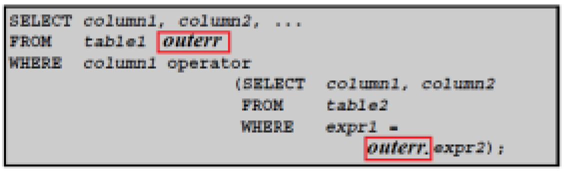
```
SELECT * FROM table1 WHERE (col1,col2) = (SELECT col3, col4 FROM table2 WHERE
id = 10);
SELECT * FROM table1 WHERE ROW(col1,col2) = (SELECT col3, col4 FROM table2
WHERE id = 10);
```

Example:

```
SELECT first_name
FROM employees
WHERE ROW(department_id, manager_id) = (SELECT department_id,
manager_id FROM departments WHERE location_id = 1800);
```

MySQL Correlated Subqueries

A correlated subquery is a subquery that contains a reference to a table (in the parent query) that also appears in the outer query. MySQL evaluates from inside to outside.



```
SELECT column1, column2, ...
FROM   table1 outerr
WHERE  column1 operator
                        (SELECT  column1, column2
                         FROM    table2
                         WHERE   expr1 =
                                   outerr.expr2);
```

Example: MySQL Correlated Subqueries

Following query find all employees who earn more than the average salary in their department.

```
SELECT last_name, salary, department_id
FROM employees outerr
WHERE salary > (SELECT AVG(salary) FROM employees WHERE
department_id = outerr.department_id);
```

Subqueries with EXISTS or NOT EXISTS

The EXISTS operator tests for the existence of rows in the results set of the subquery. If a subquery row value is found, EXISTS subquery is TRUE and in this case **NOT EXISTS subquery is FALSE**.

Syntax:

```
SELECT column1
FROM table1
WHERE EXISTS (SELECT * FROM table2);
```

In the above statement, if table2 contains any rows, even rows with NULL values, the EXISTS condition is TRUE. Generally, an EXISTS subquery starts with SELECT *, but it could begin with SELECT 'X', SELECT 5, or SELECT column1 or anything at all. MySQL ignores the SELECT list in such a subquery, so it makes no difference.

Example:

Find employees (employee_id, first_name, last_name, job_id, department_id) who have at least one person reporting to them.

```
SELECT employee_id, first_name, last_name, job_id, department_id
FROM employees E
WHERE EXISTS (SELECT * FROM employees WHERE manager_id =
E.employee_id);
```

Example: MySQL Subqueries with NOT EXISTS

NOT EXISTS subquery almost always contains correlations.

Find all departments (department_id, department_name) that do not have any employees.

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT * FROM employees WHERE department_id =
d.department_id);
```

MySQL Subqueries in the FROM Clause

Subqueries work in a SELECT statement's FROM clause.

```
SELECT ...
FROM (subquery) [AS] name ...
```

Every table in a FROM clause must have a name, therefore the [AS] name clause is mandatory. Any columns in the subquery select list must have unique names.

Lab Task(s):

Exercise

1. Write a query in SQL to display details of those employees who have changed jobs at least once. (**Sample tables:** employees & job_history)
2. Write a query to find the name (first_name, last_name) and the salary of the employees who have a higher salary than the employee whose last_name='Bull'. (**Sample tables:** employees)
3. Write a query to find the name (first_name, last_name) of all employees who works in the IT department. (**Sample tables:** employees)
4. Write a query to find the name (first_name, last_name) of the employees who have a manager and worked in a USA based department. (**Sample tables:** employees, departments & locations)
5. Write a query to find the name (first_name, last_name) of the employees who are managers. (**Sample tables:** employees)

6. Write a query to find the name (first_name, last_name), and salary of the employees whose salary is greater than the average salary. (**Sample tables:** employees)
7. Write a query to find the name (first_name, last_name), and salary of the employees whose salary is equal to the minimum salary for their job grade. (**Sample tables:** employees & jobs)
8. Write a query to find the name (first_name, last_name), and salary of the employees who earns more than the average salary and works in any of the IT departments. (**Sample tables:** employees & departments)
9. Write a query to find the name (first_name, last_name), and salary of the employees who earns more than the earning of Mr. Bell. (**Sample tables:** employees & departments)
10. Write a query to find the name (first_name, last_name), and salary of the employees who earn the same salary as the minimum salary for all departments. (**Sample tables:** employees & departments)
11. Write a query to find the name (first_name, last_name), and salary of the employees whose salary is greater than the average salary of all departments. (**Sample tables:** employees)
12. Write a query to find the name (first_name, last_name) and salary of the employees who earn a salary that is higher than the salary of all the Shipping Clerk (JOB_ID = 'SH_CLERK'). Sort the results of the salary of the lowest to highest. . (**Sample tables:** employees)
13. Write a query to find the name (first_name, last_name) of the employees who are not supervisors. (**Sample tables:** employees)
14. Write a query to display the employee ID, first name, last name, and department names of all employees. (**Sample tables:** employees & departments)
15. Write a query to display the employee ID, first name, last name, salary of all employees whose salary is above average for their departments. (**Sample tables:** employees & departments)

END