AROR UNIVERSITY
OF ART, ARCHITECTURE,
DESIGN & HERITAGE,
SUKKUR, SINDH

Department of Artificial Intelligence & Multimedia Gamming

CSC-207: Database Systems

Lab # 07: To Work with SQL Sub-Queries

Objectives

1. Introduction of SUB-QUERIES
2. Types of SUB-QUERIES

Introduction of Sub-Queries

- In SQL, a subquery (also known as a nested query or inner query) is a query placed within another query.
- It allows you to retrieve data from one table based on the results of another query.
- An SQL Subquery is a SELECT query within another query. It is also known as Inner query or Nested query and the query containing it is the outer query.
- The outer query can contain the SELECT, INSERT, UPDATE, and DELETE statements. We can use the subquery as a column expression, as a condition in SQL clauses, and with operators like =, >, <, >=, <=, IN, BETWEEN, etc.

- **Rules**

Following are the rules to be followed while writing subqueries –

- Subqueries must be enclosed within parentheses.
- Subqueries can be nested within another subquery.
- A subquery must contain the SELECT query and the FROM clause always.
- A subquery consists of all the clauses an ordinary SELECT clause can contain: GROUP BY, WHERE, HAVING, DISTINCT etc. However, an ORDER BY clause is only used when a TOP clause is specified.

Example:

This SQL query selects the job_id and the average salary (avg_salary) for each job where the average salary is greater than 6000.

```
SELECT job_id, avg_salary
FROM (
   SELECT job_id, AVG(salary) AS avg_salary
   FROM employees
   GROUP BY job_id
) AS job_avg_salaries
WHERE avg_salary > 6000;
```

In above query highlighted part is inner query.

Same can be done with simple query.

```
SELECT job_id, AVG(salary) AS avg_salary
FROM employees
GROUP BY job_id
HAVING avg_salary > 6000;
```

Example:

Retrieves the department name along with the total number of employees in each department.

```
SELECT
   department_name,
   (SELECT COUNT(*)
    FROM employees
    WHERE department_id = d.department_id) AS total_employees
FROM
   departments d;
```

```
+----------------------+-----------------+
| department_name      | total_employees |
+----------------------+-----------------+
| Administration       |               1 |
| Marketing            |               2 |
| Purchasing           |               6 |
| Human Resources      |               1 |
| Shipping             |              45 |
| IT                   |               5 |
| Public Relations     |               1 |
| Sales                |              34 |
| Executive            |               3 |
| Finance              |               6 |
| Accounting           |               2 |
```

Or using joins

```
SELECT
   d.department_name,
   COUNT(e.employee_id) AS total_employees
FROM
   departments d
LEFT JOIN
   employees e ON d.department_id = e.department_id
GROUP BY
   d.department_name;
```

The main advantages of subqueries are:

- They allow queries that are structured so that it is possible to isolate each part of a statement.
- They provide alternative ways to perform operations that would otherwise require complex joins and unions.
- Many people find subqueries more readable than complex joins or unions.

Note:

Both JOINs and subqueries are used to retrieve data from multiple tables, but they do so in different ways:

JOINs:

- JOINs are used to combine rows from two or more tables based on a related column between them.
- They can improve query performance in many cases because the database engine can optimize the query execution plan to efficiently retrieve the data.
- JOINs are generally more readable and easier to understand, especially for complex queries involving multiple tables.

Subqueries:

- Subqueries are queries nested within another query.
- They are often used to perform operations that involve comparing values from one table with values from another table or with the result of another query.
- Subqueries can be less efficient than JOINs, especially if they are correlated subqueries (i.e., dependent on the outer query).
- They are sometimes preferred for certain scenarios, such as when the relationship between the tables is complex or when specific filtering or aggregation needs to be applied to the result.

In summary, JOINs are typically more efficient and readable for joining tables based on related columns, while subqueries are useful for more complex filtering or when the relationship between tables is not straightforward.

Example:

Write a query to display the employee ID, first name, last name, and department names of all employees. (Sample tables: employees & departments)

```
SELECT
  e.employee_id,
```

```
  e.first_name,
  e.last_name,
  (SELECT d.department_name
   FROM departments d
   WHERE d.department_id = e.department_id) AS department_name
FROM employees e;
```

Or using Joins

```
SELECT e.employee_id, e.first_name, e.last_name, d.department_name
FROM employees e
JOIN departments d ON e.department_id = d.department_id;
```

Scalar Sub Queries

- A scalar subquery is a subquery that returns a single value. It's called "scalar" because it produces a single result (i.e., a single scalar value), rather than a set of rows or columns.
- Scalar subqueries are often used in places where a single value is expected, such as in the SELECT list, WHERE clause, or as part of an expression.

Example:

Find the name of employee with maximum salary in employee table.

Select first_name, last_name from emplyees where salary = (select first_name from emplyees order by salary desc limit 1);

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

```
+-------------+
| first_name |
+-------------+
| Steven     |
+-------------+
```

Alternative way:

```
SELECT first_name
FROM employees
ORDER BY salary DESC
LIMIT 1;
```

select first_name from (select first_name from employees order by salary desc limit 5) as salary;

Example: Retrieve employees who earn a salary higher than the highest average salary among

all departments

```
SELECT first_name, last_name, salary, department_id
FROM employees e
WHERE salary > (
  SELECT MAX(AVG_salary)
  FROM (
    SELECT AVG(salary) AS AVG_salary
    FROM employees
    GROUP BY department_id
  ) AS dept_avg_salary
);
```

```
+------------+-----------+----------+----------------+
| first_name | last_name | salary   | department_id |
+------------+-----------+----------+----------------+
| Steven     | King      | 24000.00 |             90 |
+------------+-----------+----------+----------------+
1 row in set (0.00 sec)
```

```
SELECT first_name, last_name, salary, department_id
FROM employees e
WHERE salary > (
  SELECT Min(AVG_salary)
  FROM (
    SELECT AVG(salary) AS AVG_salary
    FROM employees
  ) AS dept_avg_salary
);
```

Comparisons using Sub-queries.

SQL provides various comparison operators that can be used in conjunction with subqueries, such as equal to, not equal to, greater than, less than, greater than or equal to, less than or equal to, IN, NOT IN, BETWEEN, LIKE, IS NULL, and IS NOT NULL.

Here are some common 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

Note:

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.

```
mysql> SELECT NULL <=> 1, NULL <=> NULL, 3 <=> NULL;
+------------+---------------+------------+
| NULL <=> 1 | NULL <=> NULL | 3 <=> NULL |
+------------+---------------+------------+
|          0 |             1 |          0 |
+------------+---------------+------------+
1 row in set (0.00 sec)

mysql>
mysql> select null=1;
+--------+
| null=1 |
+--------+
|   NULL |
+--------+
1 row in set (0.00 sec)
```

Equal to (=):

Compares if two values are equal.

Example: SELECT * FROM employees WHERE department_id = 10;

Not equal to (<> or !=):

Compares if two values are not equal.

Example: SELECT * FROM employees WHERE department_id <> 20;

Greater than (>):

Compares if one value is greater than another.

Example: SELECT * FROM employees WHERE salary > 50000;

Less than (<):

Compares if one value is less than another.

Example: SELECT * FROM employees WHERE salary < 30000;

Greater than or equal to (>=):

Compares if one value is greater than or equal to another.

Example: SELECT * FROM employees WHERE salary >= 40000;

Less than or equal to (<=):

Compares if one value is less than or equal to another.

Example: SELECT * FROM employees WHERE salary <= 60000;

IN:

Checks if a value matches any value in a subquery or a list of specified values.

Example: SELECT * FROM employees WHERE department_id IN (10, 20, 30);

NOT IN:

Checks if a value does not match any value in a subquery or a list of specified values.

Example: SELECT * FROM employees WHERE department_id NOT IN (40, 50);

BETWEEN:

Checks if a value lies within a range specified by two other values.

Example: SELECT * FROM employees WHERE salary BETWEEN 30000 AND 50000;

LIKE:

Compares a value to a pattern using wildcard characters (% for zero or more characters, _ for a single character).

Example: SELECT * FROM employees WHERE first_name LIKE 'J%';

IS NULL:

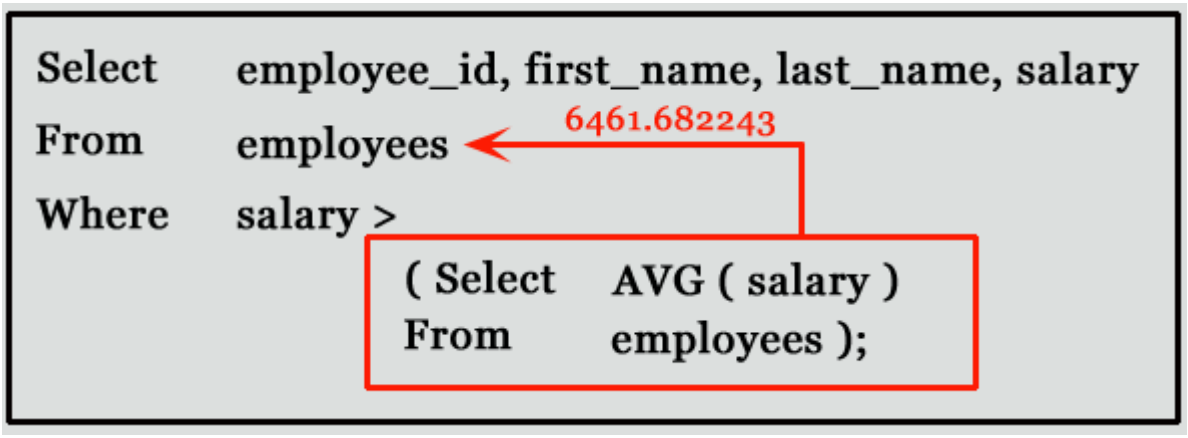
Checks if a value is NULL.

Example: SELECT * FROM employees WHERE commission_pct IS NULL;

IS NOT NULL:

Checks if a value is not NULL.

Example: SELECT * FROM employees WHERE commission_pct IS NOT NULL;



Example:

SELECT first_name, last_name, salary

FROM employees

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

first_name	last_name	salary
Steven	King	24000.00
Neena	Kochhar	17000.00
Lex	De Haan	17000.00
Alexander	Hunold	9000.00

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.

- ANY or SOME: Returns true if the condition is true for at least one value in the subquery.
- ALL: Returns true if the condition is true for all values in the subquery.

Using ALL:

Suppose we want to find employees whose salary is higher than the salary of all employees in department 30:

```
SELECT *
FROM employees
WHERE salary > ALL (
  SELECT salary
  FROM employees
  WHERE department_id = 30
);
```

This query retrieves employees whose salary is higher than the salary of every employee in department 30.

Using ANY:

Suppose we want to find employees whose salary is higher than the highest salary of any employee in department 30.

```
SELECT *
FROM employees
WHERE salary > ANY (
  SELECT salary
  FROM employees
  WHERE department_id = 30
);
```

This query retrieves employees whose salary is higher than the highest salary of any employee in department 30.

Using SOME:

Suppose we want to find employees whose salary is higher than at least one employee's salary in department 30:

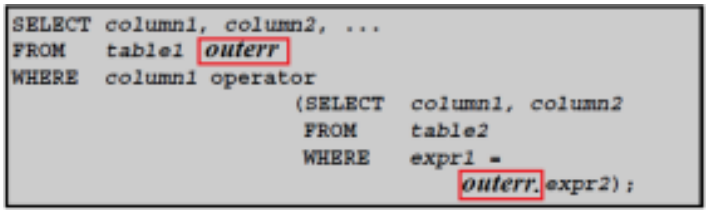
```
SELECT *
FROM employees
WHERE salary > SOME (
```

```
  SELECT salary
  FROM employees
  WHERE department_id = 30
);
```

This query retrieves employees whose salary is higher than the salary of at least one employee in department 30.

Correlated Sub-query

- A correlated subquery is a subquery that contains a reference to one or more tables (or columns) from the outer query, i.e., the parent query.
- This reference allows the subquery to be evaluated for each row processed by the outer query, making it dependent on the outer query's result set.
- MySQL evaluates correlated subqueries from inside to outside, or from bottom to top.
- For each row processed by the outer query, MySQL executes the subquery, and the result of the subquery depends on the values of the current row in the outer query.
- The subquery is re-executed for each row of the outer query, with the values from the current row in the outer query used in the evaluation of the subquery.



Example:

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

Outer Query Execution:

The outer query is evaluated first.

It retrieves data from the employees table aliased as e_outer.

For each row processed by the outer query, the correlated subquery is evaluated.

Correlated Subquery Execution:

The correlated subquery is executed once for each row returned by the outer query.

The subquery calculates the average salary (AVG(salary)) for each department_id that matches the department_id of the current row in the outer query (e_outer.department_id).

Evaluation:

The result of the correlated subquery (average salary for the corresponding department) is evaluated for each row returned by the outer query.

The outer query's WHERE clause filters the rows based on whether the salary of each employee (salary) is greater than the average salary of their respective department.

Correlated subqueries can be less efficient compared to non-correlated subqueries because they may need to be executed multiple times, once for each row in the outer query.

Subqueries with EXISTS or NOT EXISTS

EXISTS: This condition returns true if the subquery returns at least one row. It can be used in the WHERE clause of a query to filter rows based on the existence of rows returned by the subquery.

NOT EXISTS: This condition returns true if the subquery returns no rows. It can also be used in the WHERE clause to filter rows based on the absence of rows returned by the subquery.

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:

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);
```

Example:

Filters departments that have employees associated with them.

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

```
+------------------+
| department_name  |
+------------------+
| Administration   |
| Marketing        |
| Purchasing       |
| Human Resources  |
| Shipping         |
| IT               |
| Public Relations |
| Sales            |
```

Example:

Filters departments that have no employees associated with them.

```
SELECT
  department_name
FROM
  departments d
WHERE
 Not  EXISTS (
    SELECT 1
```

```
    FROM employees e
    WHERE e.department_id = d.department_id
);
```

```
+----------------------+
| department_name      |
+----------------------+
| Treasury             |
| Corporate Tax        |
| Control And Credit   |
| Shareholder Services |
| Benefits             |
| Manufacturing        |
| Construction         |
| Contracting          |
| Operations           |
```

Exercises (Class)

1. Add here all the tasks performed in lab.

Exercises (Weekly)

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)