جامعة طرابلس
University of Tripoli
1957

MySQL,
The Comprehensive Course
الدورة الشاملة

إعداد وتقديم: د. عبدالناصر ضياف

08

الاستعلامات الفرعية

Sub-queries

- نتعرف هنا على
-
-
-

SUBQUERY

- A subquery is a query that is nested inside a SELECT, INSERT, UPDATE, or DELETE statement, or inside another subquery.
- A subquery can be used anywhere an expression is allowed.
- Use a subquery to return *only* a SINGLE value in SELECT clause (direct or indirect)

Example

```
SELECT emp_last_name "Last Name", emp_first_name
  "First Name",
     emp_salary "Salary"
FROM employee
WHERE emp_salary =
     (SELECT MIN(emp_salary)
      FROM employee);
```

قائمة الموظفين ذوي أقل دخل

```
Last Name     First Name    Salary
---------------- ----------------- -----
Markis          Marcia          $25,000
Amin            Hyder            $25,000
Prescott        Sherri           $25,000
```

SUBQUERY TYPES

- There are three basic types of subqueries:

1. A subquery that returns only a single scalar value such as a row attribute or an aggregate function to be used as a row attribute or in the WHERE clause with a comparison operator in its outer query.
2. A subquery that returns a list of single values (a column-like) to be used by the IN or EXISTS operators in its outer query.
3. A subquery that returns a list of multiple values (a table-like) to be used by the EXISTS operator in its outer query, because EXISTS deals with rows instead of columns.

SUBQUERY – General Rules

- A subquery SELECT statement is very similar to the SELECT statement used to begin a regular query.
- A subquery nested in the outer SELECT statement has the following components:
  - A regular SELECT query including the regular select list components.
  - A regular FROM clause including one or more table or view names.
  - An optional WHERE clause.
  - An optional GROUP BY clause.
  - An optional HAVING clause.
- The subquery is always enclosed in parentheses.
- It may only include an ORDER BY clause when a LIMIT clause is also specified.

SUBQUERIES AND THE IN Operator

- Subqueries that are introduced with the keyword **IN** take the general form:
  - WHERE expression [NOT] IN (subquery)

Example الموظفين الذين لديهم أولاد ذكور

```
SELECT emp_last_name "Last Name",
    emp_first_name "First Name"
FROM employee
WHERE emp_nid IN
     (SELECT dep_emp_nid
      FROM dependent
      WHERE dep_gender = 'M');
```

```
Last Name    First Name
------------ -------------
Bock         Douglas
Zhu          Waiman
Joyner       Suz
```

SUBQUERIES AND THE IN Operator

- Conceptually, this statement is evaluated in two steps.
- First, the inner query returns the identification numbers of those employees that have male dependents.

```
SELECT dep_emp_nid
FROM dependent
WHERE dep_gender = 'M';

DEP_EMP_NID
----------
999444444
999555555
999111111
```

SUBQUERIES AND THE IN Operator

- Next, these national identification number values are substituted into the outer query as the listing that is the object of the IN operator. So, from a conceptual perspective, the outer query now looks like the following:

```
SELECT emp_last_name "Last Name",
emp_first_name "First Name"
FROM employee
WHERE emp_nid IN (999444444, 999555555, 999111111);

        Last Name    First Name
        ------------------------
        Joyner        Suzanne
        Zhu           Waiman
        Bock          Douglas
```

The NOT IN Operator

- Unlike the IN operator which take FALSE in the beginning and compares each row-value against the given key and returns TRUE once matched, the NOT IN operator takes TRUE in the beginning and returns FALSE once the key is matched.

قائمة الطلاب الذين لم ينزّلو موادهم

```
SELECT id, name
FROM student
WHERE status='active'  AND
    id NOT IN
    (SELECT student_id FROM grade WHERE status='in-progress');
```

```
Id              Name
---------------------
2013005         Amin
2014112         Zainab
2014217         Hyder
```

MULTIPLE LEVELS OF NESTING

- Subqueries may themselves contain subqueries.
- When the WHERE clause of a subquery has as its object another subquery, these are termed *nested subqueries*.
- Most of RDBMSs places no practical limit on the number of queries that can be nested in a WHERE clause.

Example

- Consider the problem of producing a listing of employees that worked more than 10 hours on the project named *HighWay-401*:

```
SELECT full_name ‘Full Name’
FROM employee
WHERE emp_nid IN
    (SELECT work_emp_nid
     FROM assignment
     WHERE work_hours>10 AND work_proj_number IN
        (SELECT proj_number
         FROM project
         WHERE proj_name = ‘HighWay-401 ') );
```

```
Full Name
---------------
Bock Douglas
Prescott Sherri
```

Understanding SUBQUERIES

- In order to understand how this query executes, we need to execute each query independently starting with the most inner subquery.

SUBQUERIES AND COMPARISON OPERATORS

- The general form of the WHERE clause with a comparison operator is similar to that used thus far in the text.
- Note that the subquery is again enclosed by parentheses.

```
WHERE <expression> <comparison_operator>
  (subquery)
```

SUBQUERIES AND COMPARISON OPERATORS

- The most important point to remember when using a subquery with a comparison operator is that the subquery can only return a single or *scalar* value.
- This is also termed a *scalar subquery* because an aggregate function value or a single column of a single row is returned by the subquery.
- If a subquery returns more than one value, the RDBMS will generate an error message, and the query will fail to execute.

SUBQUERIES AND COMPARISON OPERATORS

- Let's examine a subquery that will not execute because it violates the "single value" rule.
- The subquery shown below returns multiple values for the *emp_salary* column.

```
SELECT emp_nid
FROM employee
   WHERE emp_salary >
    (SELECT emp_salary
     FROM employee
        WHERE emp_salary > 40000);
```

#1242 - Subquery returns more than 1 row (in MySQL)

Aggregate Functions and Comparison Operators

- The aggregate functions (AVG, SUM, MAX, MIN, and COUNT) always return a *scalar* result table.

```
SELECT emp_last_name "Last Name",
    emp_first_name "First Name",
    emp_salary "Salary"
FROM employee
WHERE emp_salary >
       (SELECT AVG(emp_salary)
        FROM employee);

Last Name    First Name   Salary
--------------- --------------- ----------
Bordoloi       Bijoy           $55,000
Joyner         Suzanne         $43,000
Zhu            Waiman          $43,000
Joshi          Dinesh          $38,000
```

Comparison Operators Modified with the ALL or ANY Keywords

- The ALL and ANY keywords can modify a comparison operator to allow an outer query to accept multiple values from a subquery.
- The general form of the WHERE clause for this type of query is shown here.

```
        WHERE <expression> <comparison_operator>
[ALL |          ANY] (subquery)
```

- Subqueries that use these keywords may also include GROUP BY and HAVING clauses.

The ALL Keyword

- The ALL keyword modifies the greater than comparison operator to mean greater than all values.

```
SELECT emp_last_name "Last Name",
    emp_first_name "First Name",
    emp_salary "Salary"
FROM employee
WHERE emp_salary > ALL
    (SELECT emp_salary
        FROM employee
        WHERE emp_dpt_number = 7);
Last Name    First Name   Salary
--------------- --------------- --------
Bordoloi      Bijoy         $55,000
```

The ANY Keyword

- The ANY keyword is not as restrictive as the ALL keyword.
- When used with the greater than comparison operator, "> ANY" means greater than <u>some</u> value.

Example

```
SELECT emp_last_name "Last Name",
    emp_first_name "First Name",
    emp_salary "Salary"
FROM employee
WHERE emp_salary > ANY
    (SELECT emp_salary
        FROM employee
        WHERE emp_salary > 30000);

Last Name     First Name    Salary
--------------- --------------- --------
Bordoloi      Bijoy          $55,000
Joyner        Suzanne          $43,000
Zhu           Wai              $43,000
```

An "= ANY" (Equal Any) Example

- The "= ANY" operator is exactly equivalent to the IN operator.
- For example, to find the names of employees that have male dependents, you can use either IN or "= ANY" – both of the queries shown below will produce an identical result table.

```
SELECT emp_last_name "Last Name", emp_first_name "First Name"
FROM employee
WHERE emp_ssn IN
    (SELECT dep_emp_ssn
     FROM dependent
     WHERE dep_gender = 'M');

SELECT emp_last_name "Last Name", emp_first_name "First Name"
FROM employee
WHERE emp_ssn = ANY
    (SELECT dep_emp_ssn
     FROM dependent
     WHERE dep_gender = 'M');
```

An "= ANY" (Equal Any) Example

- OUTPUT

```
Last Name   First Name
----------------- -----------------
Bock            Douglas
Zhu              Waiman
Joyner           Suzanne
```

A "!= ANY" (Not Equal Any) Example

- The "= ANY" is identical to the IN operator.
- However, the "!= ANY" (not equal any) is **not** equivalent to the NOT IN operator.
- If a subquery of employee salaries produces an intermediate result table with the salaries $38,000, $43,000, and $55,000, then the WHERE clause shown here means "NOT $38,000" AND "NOT $43,000" AND "NOT $55,000".

```
WHERE NOT IN (38000, 43000, 55000);
```

- However, the "!= ANY" comparison operator and keyword combination shown in this next WHERE clause means "NOT $38,000" OR "NOT $43,000" OR "NOT $55,000".

CORRELATED SUBQUERIES

- A *correlated subquery* is one where the inner query depends on values provided by the outer query.
- This means the inner query is executed repeatedly, **once for each row that might be selected by the outer query.**

CORRELATED SUBQUERIES

```
SELECT emp_last_name "Last Name",
 emp_first_name "First Name",
 emp_dpt_number "Dept",
 emp_salary "Salary"
FROM employee e1 WHERE emp_salary =
     (SELECT MAX(emp_salary)
       FROM employee
       WHERE emp_dpt_number =
  e1.emp_dpt_number);
```

CORRELATED SUBQUERIES

- Output

```
Last Name   FirstName   Dept  Salary
----------- ----------- ----- --------
Bordoloi    Bijoy           1  $55,000
Joyner      Suzanne         3  $43,000
Zhu         Waiman          7  $43,000
```

CORRELATED SUBQUERIES

- The subquery in this SELECT statement cannot be resolved independently of the main query.
- Notice that the outer query specifies that rows are selected from the *employee* table with an alias name of *e1*.
- The inner query compares the employee department number column (*emp_dpt_number*) of the *employee* table to the same column for the alias table name *e1*.

CORRELATED SUBQUERIES

- The value of *e1.emp_dpt_number* is treated like a variable – it changes as the MySQL server examines each row of the employee table.
- The subquery's results are correlated with each individual row of the main query – thus, the term *correlated subquery*.

Subqueries and the EXISTS operator

- When a subquery uses the EXISTS operator, the subquery functions as an *existence test*.
- The WHERE clause of the outer query tests for the existence of rows returned by the inner query.
- The subquery does not actually produce any data; rather, it returns a value of TRUE or FALSE.

Subqueries and the EXISTS operator

- The general format of a subquery WHERE clause with an EXISTS operator is shown here.
- Note that the NOT operator can also be used to negate the result of the EXISTS operator.

```
WHERE [NOT] EXISTS (subquery)
```

Example

```
SELECT emp_last_name "Last Name",
  emp_first_name "First Name"
FROM employee
WHERE EXISTS
     (SELECT *
      FROM dependent
      WHERE employee.emp_ssn = dep_emp_ssn);
```

```
Last Name  First Name
---------- ---------------
Joyner     Suzanne
Zhu        Waiman
Bock       Douglas
```

Subqueries and the EXISTS operator

- Subqueries using an EXISTS operator are a bit different from other subqueries, in the following ways:

1. The keyword EXISTS is not preceded by a column name, constant, or other expression.
2. The SELECT clause list of a subquery that uses an EXISTS operator almost always consists of an asterisk (*). This is because there is no real point in listing column names since you are simply testing for the existence of rows that meet the conditions specified in the subquery.

Subqueries and the EXISTS operator

3. The subquery evaluates to TRUE or FALSE rather than returning any data.
4. A subquery that uses an EXISTS operator will always be a correlated subquery.

Subqueries and the EXISTS operator

- The EXISTS operator is very important, because there is often no alternative to its use.
- All queries that use the IN operator or a modified comparison operator (=, <, >, etc. modified by ANY or ALL) can be expressed with the EXISTS operator.
- However, some queries formulated with EXISTS cannot be expressed in any other way!

Subqueries and the EXISTS operator

```
SELECT emp_last_name
FROM employee
WHERE emp_ssn = ANY
    (SELECT dep_emp_ssn
     FROM dependent);
```

```
EMP_LAST_NAME
-------------
Bock
Zhu
Joyner
```

```
SELECT emp_last_name
FROM employee
WHERE EXISTS
     (SELECT *
      FROM dependent
      WHERE emp_ssn =
   dep_emp_ssn);
```

```
EMP_LAST_NAME
----------------
Bock
Zhu
Joyner
```

Subqueries and the EXISTS operator

- The NOT EXISTS operator is the mirror-image of the EXISTS operator.
- A query that uses NOT EXISTS in the WHERE clause is satisfied if the subquery returns no rows.

Subqueries and the ORDER BY Clause

- The SELECT statement shown below adds the ORDER BY clause to specify sorting by first name within last name.
- Note that the ORDER BY clause is placed after the WHERE clause, and that this includes the subquery as part of the WHERE clause.

```
SELECT emp_last_name "Last Name",
 emp_first_name "First Name"
FROM employee
WHERE EXISTS
    (SELECT *
     FROM dependent
     WHERE emp_ssn = dep_emp_ssn)
ORDER BY emp_last_name, emp_first_name;
```

Subqueries and the ORDER BY Clause

Output:

```
Last Name  First Name
---------- ---------------
Bock       Douglas
Joyner     Suzanne
Zhu        Waiman
```