Subqueries

Overview

There are numerous occasions when a query will be based on some unknown value that is already contained in the database. One option is to first look up the unknown value and then issue the query. The alternative is to create a subquery—one query nested inside another query. Subqueries are widely used in application development. This lesson addresses single-row, multiple-row, and multiple-column subqueries.

Objectives

After completing this lesson, you should be able to do the following:

- Determine when it is appropriate to use a subquery
- Identify which clauses can contain subqueries
- Distinguish between an outer query and a subquery
- Use a single-row subquery in a WHERE clause
- Use a single-row subquery in a HAVING clause
- Use a single-row subquery in a SELECT clause
- Distinguish between single-row and multiple-row comparison operators
- Use a multiple-row subquery in a WHERE clause
- Use a multiple-row subquery in a HAVING clause
- Use a multiple-column subquery in a WHERE clause
- Create an inline view using a multiple-column subquery in a FROM clause
- Distinguish between correlated and uncorrelated subqueries

Contents

1. Subqueries

A subquery is necessary when a search is based on an unknown value that is contained within a database table.

The basic characteristics of a subquery**:**

- A subquery is a query (SELECT statement) inside a query
- A subquery is normally expressed inside parentheses
- The **first query** in the SQL statement is known as the **outer query**
- The query inside the SQL statement is known as the **inner query**
- The **inner query** is executed first
- The output of an inner query is used as the input for the outer query
- The entire SQL statement is sometimes referred to as a **nested query**

There are three basic types of subqueries:

1. **Single-Row Subqueries (one column and one row).** This subquery is used anywhere a single value is expected, as in the right side of a comparison expression. Obviously, when you assign a value to an attribute, that value is a single value, not a list of values. Therefore, the subquery must return only one value (one column, one row). If the query returns multiple values, the DBMS will generate an error. There is a special case of a single-row subquery that contains exactly one column. This type of subquery is called a scalar subquery.
2. **Multiple-Row Subqueries (one column and multiple rows).** This type of subquery is used anywhere a list of values is expected, such as when using the IN clause. This type of subquery is used frequently in combination with the IN operator in a WHERE conditional expression.
3. **Multiple-Column Subqueries (multicolumn, multirow set of values).** This type of subquery can be used anywhere a table is expected, such as when using the FROM clause.

Also, there are two subtypes of subqueries that can return single or multiple rows:

1. **Correlated subqueries** reference one or more columns in the outer SQL statement. These are called correlated subqueries because they are related to the outer SQL statement through the same columns.
2. **Nested subqueries** are placed within another subquery.

It's important to note that a subquery can return no values at all; it is a NULL. In such cases, the output of the outer query may result in error or a null empty set depending on where the subquery is used (in a comparison, an expression, or a table set).

2. Single-Row Subqueries

A single-row subquery can only return a single data element. A single-row subquery is specified by the use of a single-row operator. Valid single row operators include any of the mathematical comparison operators. A single-row subquery is specified in the WHERE clause when the comparison is not based on a group condition. If the subquery results are used for comparison against grouped data, the subquery must be nested in a HAVING clause.

2.1. Single-Row Subquery in a WHERE Clause

The most common type of subquery uses an inner SELECT subquery on the right side of a WHERE comparison expression.

Suppose we want to find all of the people who work in the same department as employee 111111101. We could write a query to find that department number.

```
SELECT dept_number
FROM employee
WHERE SSN = 111111101;
```

And then write a query to find employees in that department.

```
SELECt ssn, last_name as "Employee",dept_number
FROM employee
WHERE dept_number = 6
```

But we can also build these two queries into one query by using a subquery.

A simple subquery- we want to find employees who work in the same department as an employee with SSN 111111101.

```
SELECT SSN, last_name as "Employee."
FROM employee
WHERE dept_number = (SELECT dept_number
                     FROM employee WHERE SSN = 111111101);
```

The inner query, the subquery, is a complete query that could stand by itself. It returns a single value- the department number where this employee works.

The subquery is enclosed in parentheses, and the result of the subquery is used by the outer query to return the other employees from that department.

Example 1: Find the salaries of all employees with a salary greater than or equal to the average salary, you write the following query:

```
mysql> SELECT SSN, first_name,last_name, salary
       FROM employee
       WHERE SALARY >= (SELECT AVG( SALARY ) FROM employee );
+-----------+------------+-----------+----------+
| ssn       | first_name | last_name | salary   |
+-----------+------------+-----------+----------+
| 111111100 | Jared      | James     | 85000.00 |
| 222222200 | Evan       | Wallis    | 92000.00 |
| 222222201 | Josh       | Zell      | 56000.00 |
| 222222203 | Tom        | Brand     | 62500.00 |
| 222222204 | Jenny      | Vos       | 61000.00 |
| 333333300 | Kim        | Grace     | 79000.00 |
| 444444400 | Alex       | Freed     | 89000.00 |
| 444444401 | Bonnie     | Bays      | 70000.00 |
| 444444402 | Alec       | Best      | 60000.00 |
| 555555500 | John       | James     | 81000.00 |
| 555555501 | Nandita    | Ball      | 62000.00 |
| 666666600 | Bob        | B         | 96000.00 |
| 834565577 | James      | Nestor    | 54000.00 |
| 834565999 | Ken        | Numoto    | 54000.00 |
| 888665555 | James      | Borg      | 55000.00 |
| 888665577 | Rick       | Brown     | 54000.00 |
+-----------+------------+-----------+----------+
```

Example 2: Find details of those employees whose salary is greater than the average salary for all employees. Output salary in descending order.

```
mysql> SELECT first_name, last_name, salary
       FROM employee
       WHERE salary > (SELECT avg( salary )FROM employee)
       ORDER BY salary DESC;
+------------+-----------+----------+
| first_name | last_name | salary   |
+------------+-----------+----------+
| James      | Nestor    | 54000.00 |
| Ken        | Numoto    | 54000.00 |
| Rick       | Brown     | 54000.00 |
| James      | Borg      | 55000.00 |
| Josh       | Zell      | 56000.00 |
| Alec       | Best      | 60000.00 |
| Jenny      | Vos       | 61000.00 |
| Nandita    | Ball      | 62000.00 |
| Tom        | Brand     | 62500.00 |
| Bonnie     | Bays      | 70000.00 |
| Kim        | Grace     | 79000.00 |
| John       | James     | 81000.00 |
| Jared      | James     | 85000.00 |
| Alex       | Freed     | 89000.00 |
| Evan       | Wallis    | 92000.00 |
| Bob        | B         | 96000.00 |
+------------+-----------+----------+
16 rows in set (0.00 sec)
```

The inner query is executed first, and this query's result is passed to the outer query.

Note that this type of query, when used in a >, <, =, >=, or <= conditional expression, requires a subquery that returns only **one single value** (one column, one row). The value generated by the subquery must be of a "comparable" data type; if the attribute to the left of the comparison symbol is a character type, the subquery must return a character string. Also, if the query returns more than a single value, the DBMS will generate an error.

The syntax of a WHERE clause that uses a comparison operator WHERE expression

```
comparison_operator [SOME|ANY|ALL](subquery)
```

SOME Operator

The phrase "**greater than at least one**" is represented in SQL by > some.
Definition of the SOME Clause SOME => **'at least one member.'**

Example 3: Display the last name and salary of every employee who reports to Wong.

```
mysql> SELECT last_name, salary
       FROM employee
       WHERE super_ssn = (SELECT ssn
                          FROM employee
                          WHERE last_name = "Wong");
+-----------+----------+
| last_name | salary   |
+-----------+----------+
| English   | 25000.00 |
| Narayan   | 38000.00 |
+-----------+----------+
2 rows in set (0.00 sec)
```

Example 4: Find the names of all employees whose salary is greater than at least one employee in the department number 7.

```
mysql> SELECT first_name, last_name, salary
       FROM employee
       WHERE SALARY > SOME (SELECT SALARY
                            FROM employee
                            WHERE dept_number = 7);
+------------+-----------+----------+
| first_name | last_name | salary   |
+------------+-----------+----------+
| Jared      | James     | 85000.00 |
| Jon        | Jones     | 45000.00 |
| Brad       | Knight    | 44000.00 |
| Evan       | Wallis    | 92000.00 |
| Josh       | Zell      | 56000.00 |
| Andy       | Vile      | 53000.00 |
| Tom        | Brand     | 62500.00 |
| Jenny      | Vos       | 61000.00 |
| Kim        | Grace     | 79000.00 |
| Jeff       | Chase     | 44000.00 |
| Alex       | Freed     | 89000.00 |
| Bonnie     | Bays      | 70000.00 |
| Alec       | Best      | 60000.00 |
| Sam        | Snedden   | 48000.00 |
| John       | James     | 81000.00 |
| Nandita    | Ball      | 62000.00 |
| Bob        | B         | 96000.00 |
| Kate       | King      | 44000.00 |
| James      | Nestor    | 54000.00 |
| Ken        | Numoto    | 54000.00 |
| James      | Borg      | 55000.00 |
| Rick       | Brown     | 54000.00 |
+------------+-----------+----------+
22 rows in set (0.00 sec)
```

Using ALL and ANY with a Multiple-Row Subquery

The ANY and ALL operators can be used with various mathematical operators to specify how the results of the subquery should be evaluated. The word ALL, which must follow a comparison operator, means "return TRUE if the comparison is TRUE for ALL of the values in the column that the subquery returns.". ALL and ANY Operator Combinations

Operator	**Description**
> ALL	More than the highest value returned by the subquery
< ALL	Less than the lowest value returned by the subquery
< ANY	Less than the highest value returned by the subquery
> ANY	More than the lowest value returned by the subquery
= ANY	Equal to any value returned by the subquery (same as IN)

Example 5: Find the names of **all employees** whose salary is greater than **all** the employees in the department number 5.

```
mysql> SELECT first_name, last_name, salary
       FROM employee
       WHERE SALARY > ALL (SELECT SALARY
                           FROM employee
                           WHERE dept_number = 5);
+------------+-----------+----------+
| first_name | last_name | salary   |
+------------+-----------+----------+
| Jared      | James     | 85000.00 |
| Jon        | Jones     | 45000.00 |
| Brad       | Knight    | 44000.00 |
| Evan       | Wallis    | 92000.00 |
| Josh       | Zell      | 56000.00 |
| Andy       | Vile      | 53000.00 |
| Tom        | Brand     | 62500.00 |
| Jenny      | Vos       | 61000.00 |
| Chris      | Carter    | 43000.00 |
| Kim        | Grace     | 79000.00 |
| Jeff       | Chase     | 44000.00 |
| Alex       | Freed     | 89000.00 |
| Bonnie     | Bays      | 70000.00 |
| Alec       | Best      | 60000.00 |
| Sam        | Snedden   | 48000.00 |
| John       | James     | 81000.00 |
| Nandita    | Ball      | 62000.00 |
| Bob        | B         | 96000.00 |
| Kate       | King      | 44000.00 |
| Lyle       | Leslie    | 41000.00 |
| James      | Nestor    | 54000.00 |
| Ken        | Numoto    | 54000.00 |
| James      | Borg      | 55000.00 |
| Rick       | Brown     | 54000.00 |
| Jennifer   | Wallace   | 43000.00 |
+------------+-----------+----------+
25 rows in set (0.00 sec)
```

2.2. Single-Row Subquery in a HAVING Clause

You use the HAVING clause to filter groups of rows. You can place a subquery in the HAVING clause of an outer query. This allows you to filter groups of rows based on the result returned by a subquery.

2.3. Single-Row Subquery in a SELECT Clause

A single-row subquery can also be nested in the outer query's SELECT clause. In this case, the value the subquery returns is available for every row of output the outer query generates.

A subquery in the attribute list must return one single value; otherwise, an error code is raised.

Example 6: Find all departments along with the number of employees in each department.

```
mysql> SELECT dept_name
              , (SELECT COUNT(*) FROM employee
                 WHERE department.Dept_number = employee.Dept_number)
              AS num_employees
       FROM department;
+----------------+---------------+
| dept_name      | num_employees |
+----------------+---------------+
| Administration |             2 |
| Hardware       |             9 |
| Headquarters   |             0 |
| InfoSystems    |             1 |
| Marketing      |             0 |
| Production     |             1 |
| Research       |             3 |
| Sales          |             5 |
| Software       |             6 |
+----------------+---------------+
9 rows in set (0.00 sec)
```

3. Multiple-Row Subqueries

A multiple-row subquery can return more than one row of results but only one column of output. The most commonly used multiple-row operator is the IN comparison operator. Also, the ANY and ALL operators can be used with various mathematical operators to specify how the results of the subquery should be evaluated.

3.1. Using IN with a Multiple-Row Subquery

Used for comparison of value with a set of values.

Example 7: Retrieve the name of each employee who has a dependent with the same first name or same gender as the employee**.**

```
mysql> SELECT e.first_name,e.last_name
       FROM employee e
       WHERE e.ssn IN (SELECT emp_ssn
                       FROM dependent
                       WHERE  e.first_name = dependent_name
                       OR e.gender = gender);
+----------+-------+
| fname    | lname |
+----------+-------+
| John     | Smith |
| Franklin | Wong  |
| Alex     | Freed |
| Alec     | Best  |
+----------+-------+
4 rows in set (0.01 sec)
```

3.2. Using NOT IN with a Multiple-Row Subquery

We use the **not in the** construct in a way similar to the in the construct.

Example 8: For each department that has less than five employees, retrieve the number of its employees who are making more than $40,000.

```
mysql> SELECT count(*)
       FROM department d, employee e
       WHERE d.dept_number = e.dept_number
       AND salary > 40000
       AND e.dept_number IN (SELECT dept_number
                             FROM employee
                             GROUP BY dept_number
                             HAVING count(*) < 5);
+------------+
| count( * ) |
+------------+
|          4 |
+------------+
```

Example 9: Retrieve the names of employees who do not work on any project.

```
mysql> SELECT last_name, first_name
       FROM employee
       WHERE SSN
       NOT IN (SELECT emp_ssn FROM assignment);
+-----------+------------+
| last_name | first_name |
+-----------+------------+
| B         | Bob        |
| King      | Kate       |
| Nestor    | James      |
| Numoto    | Ken        |
| Brown     | Rick       |
+-----------+------------+
5 rows in set (0.02 sec)
```

4. Multiple-Column Subqueries

A multiple-column subquery is a subquery that can return several columns, as well as several rows, in its results. When a multiple-column subquery is used in a **FROM** clause, it is referred to an **inline view**. When using a multiple-column subquery for comparison purposes, the column list specified in the subquery must be in the same order as the column list specified in the WHERE or HAVING clause. The column list included in the WHERE or HAVING clause must be enclosed in parentheses.

1.2.1 Multiple-Column Subquery in a FROM Clause (Inline Views)

You can place a subquery in the FROM clause of an outer query. These types of subqueries are also known as **inline views** because the subquery provides data in line with the FROM clause.

Example 10: Find the average employees' salaries of those departments where the average salary is less than $42,000.

```
mysql> SELECT AVG_SALARY
       FROM (SELECT dept_number, AVG( SALARY ) AS AVG_SALARY
             FROM employee
             GROUP BY dept_number) empl
       WHERE AVG_SALARY < 42000;
+--------------+
| AVG_SALARY   |
+--------------+
| 31000.000000 |
| 33250.000000 |
+--------------+
2 rows in set (0.00 sec)
```

A subquery cannot contain an ORDER BY clause. Instead, any ordering must be done in the outer query.

5. Correlated Subqueries

A correlated subquery references one or more columns in the outer SQL statement. These are called correlated subqueries because they are related to the outer SQL statement through the same columns.

The syntax of a subquery that uses the EXISTS operator

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

A correlated subquery is a subquery that executes once for each row in the outer query. The relational DBMS uses the same sequence to produce correlated subquery results:

1. It initiates the outer query.
2. For each row of the outer query result set, it executes the inner query by passing the outer row to the inner query.

That process is the opposite of the subqueries you have seen so far. The query is called a **correlated subquery** because the inner query is related to the outer query because the inner query references a column of the outer subquery.

5.1. Using EXISTS and NOT EXISTS with a Correlated Subquery

You use the EXISTS operator to check for the existence of rows returned by a subquery. Although you can use EXISTS with non-correlated subqueries, EXISTS is typically used with correlated subqueries. NOT EXISTS does the logical opposite of EXISTS. NOT EXISTS checks if rows do not exist in the results returned by a subquery.

5.2. Using EXISTS with a Correlated Subquery

The EXISTS function in SQL is used to check whether the result of a correlated nested query is empty (contains no tuples) or not. The result of EXISTS is a Boolean value TRUE if the nested query result contains at least one tuple, or FALSE if the nested query result contains no tuples.

Example 11: Display the names of managers who have at least one dependent.

```
mysql> SELECT first_name,last_name
       FROM employee
       WHERE EXISTS (SELECT * FROM department
                     WHERE ssn = mgr_ssn)
       AND EXISTS (SELECT * FROM dependent
                   WHERE ssn=emp_ssn);
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| John       | Smith     |
| Franklin   | Wong      |
| Alex       | Freed     |
| Jennifer   | Wallace   |
+------------+-----------+
4 rows in set (0.08 sec)
```

5.3. Using NOT EXISTS with a Correlated Subquery

We can test for the nonexistence of tuples in a subquery by using the not exists construct. We can use the not exists construct to simulate the set containment (that is, superset) operation:

EXISTS and NOT EXISTS are typically used in conjunction with a correlated nested query.

Example 12: Retrieve the names of employees who have no dependents.

```
mysql> SELECT first_name,last_name
       FROM employee
       WHERE NOT EXISTS (SELECT *
                         FROM dependent
                         WHERE ssn=emp_ssn);
+---------+---------+
| fname   | lname   |
+---------+---------+
| Jared   | James   |
| Jon     | Jones   |
+---------+---------+
. . .  rows omitted
27 rows in set (0.00 sec)
```

5.4. EXISTS and NOT EXISTS Versus IN and NOT IN

The IN operator checks if a value is contained in a list of values. EXISTS is different from IN. EXISTS checks for the existence of rows, whereas IN checks for actual values.

There is an important difference between NOT EXISTS and NOT IN: When a list of values contains a null value, NOT EXISTS returns true, but NOT IN returns false.

6. Writing UPDATE and DELETE Statements Containing Subqueries

You can place subqueries inside UPDATE and DELETE statements. In a UPDATE statement, you can set a column to the result returned by a single-row subquery.

7. Writing a DELETE Statement Containing a Subquery

You can use the rows returned by a subquery in the WHERE clause of a DELETE statement.

Example 13:

```
mysql> UPDATE employee
       SET salary = (SELECT AVG(salary)
                     FROM employee)
       WHERE SSN = 111111100;
```

Example 14: The following DELETE statement removes the employee whose salary is greater than the average salary returned by a subquery.

```
mysql> DELETE FROM employee
       WHERE salary > (SELECT AVG (salary)
                       FROM employee);
```

Example 15: Display employee Kate King's department name.

```
mysql> SELECT dept_name
       FROM department
       WHERE dept_number = (SELECT dept_number
                            FROM employee
                            WHERE Last_name = 'king'
                            AND First_name = 'kate');
+-----------+
| dept_name |
+-----------+
| Sales     |
+-----------+
1 row in set (0.02 sec)
```

Example 16: Find employees with minimum salary in their own department with the use of correlated subquery.

```
mysql> SELECT last_name, first_name, salary, dept_number
       FROM employee e
       WHERE salary = (SELECT MIN(Salary)
                       FROM employee
                       WHERE Dept_number = e.Dept_number
                       GROUP BY dept_number);
+-----------+------------+----------+-------------+
| Mark      | Justin     | 40000.00 |           6 |
| Carter    | Chris      | 43000.00 |           7 |
| English   | Joyce      | 25000.00 |           5 |
| Jarvis    | Jill       | 36000.00 |           8 |
| Numoto    | Ken        | 54000.00 |           3 |
| Borg      | James      | 55000.00 |           1 |
| Brown     | Rick       | 54000.00 |           9 |
| Jabbar    | Ahmad      | 25000.00 |           4 |
| Zelaya    | Alicia     | 25000.00 |           4 |
+-----------+------------+----------+-------------+
9 rows in set (0.00 sec)
```

Example 17: Which department has more employees than department 6?

```
mysql> SELECT Dept_number, COUNT(*)
       FROM employee
       GROUP BY dept_number
       HAVING COUNT(*) >(SELECT COUNT(*)
                         FROM employee
                         WHERE Dept_number = 6);
+-------------+----------+
| dept_number | COUNT(*) |
+-------------+----------+
|           7 |       10 |
+-------------+----------+
1 row in set (0.00 sec)
```

Example 18: Write a subquery that finds average salary by each department. Check to find if employee 444444400's salary satisfies =ANY, <ANY, >ANY, <ALL, or >ALL condition against those departmental average salaries.

```
mysql> SELECT Last_name, First_name, Salary
       FROM employee
       WHERE ssn = 444444400
       AND Salary > ANY
       (SELECT AVG(Salary) FROM employee GROUP BY dept_number);
+-----------+------------+----------+
| Last_name | First_name | Salary   |
+-----------+------------+----------+
| Freed     | Alex       | 89000.00 |
+-----------+------------+----------+
1 row in set (0.00 sec)
```

Example 19: Retrieve the names of all employees who work in the department that has the employee with the highest salary among all employees.

```
mysql> SELECT last_name
       FROM employee
       WHERE dept_number = (SELECT dept_number
                            FROM employee
                            WHERE salary = (SELECT MAX(salary)
                                            FROM employee) )

+-----------+
| last_name |
+-----------+
| James     |
| B         |
| Jarvis    |
| King      |
| Leslie    |
| Nestor    |
+-----------+
6 rows in set (0.00 sec)
```

Example 20: Retrieve the names of all employees who have two or more dependents.

```
mysql> SELECT last_name,first_name
       FROM employee
       WHERE(SELECT count(*) FROM dependent WHERE ssn = emp_ssn) >=2;
+-----------+------------+
| last_name | first_name |
+-----------+------------+
| Smith     | John       |
| Wong      | Franklin   |
| Freed     | Alex       |
+-----------+------------+
3 rows in set (0.00 sec)
```

Example 21: Retrieve the names of employees who make at least $10,000 more than the employee who is paid the least in the company.

```
mysql> SELECT last_name
       FROM employee
       WHERE salary >= 10000 + (SELECT MIN(salary) FROM employee);
+-----------+
| last_name |
+-----------+
| James     |
| Jones     |
| Mark      |
| Knight    |
+-----------+
. . .  rows omitted
29 rows in set (0.00 sec)
```

Example 22: Write a query to display the last names of the employees who earn less than the average salary in their departments.

```
mysql> SELECT last_name
       FROM employee o
       WHERE o.salary < (SELECT AVG(i.salary)
                         FROM employee i
                         WHERE i.dept_number = o.dept_number);
+-----------+
| last_name |
+-----------+
| Jones     |
| Mark      |
| Knight    |
| Smith     |
| Zell      |
| Vile      |
| Brand     |
| Vos       |
| Carter    |
| Chase     |
| Best      |
| Snedden   |
| English   |
| Jarvis    |
+-----------+
. . .  rows omitted
19 rows in set (0.01 sec)
```

Key Terms

- **Correlated subquery:** A subquery that references a column in the outer query. The outer query executes the subquery once for every row in the outer query.
- **Multiple-column subquery**: A nested query that returns more than one column of results to the outer query. It can be listed in the FROM, WHERE, or HAVING clause.
- **Multiple-row subquery**: Nested queries that return more than one row of results to the parent query. They are most commonly used in WHERE and HAVING clauses and require multiple-row operators.
- **Single value**: The output of a single-row subquery.
- **Single-row subquery**: A nested subquery that can return to the outer query only one row of results that consists of only one column. The output of a single-row subquery is a single value.
- **Uncorrelated subquery:** A subquery that follows this method of processing: the subquery is executed, then the results of the subquery are passed to the outer query, and finally the outer query is executed.
- **subquery:** The inner query in nested queries