Database Programming

Fundamentals of Subqueries

Objectives

This lesson covers the following objectives:

- Define and explain the purpose of subqueries for retrieving data
- Construct and execute a single-row subquery in the WHERE clause
- Distinguish between single-row and multiple-row subqueries
- Distinguish between pair-wise and non-pair-wise subqueries
- Use the EXISTS and NOT EXISTS operators in a query

Purpose

Has a friend asked you to go to a movie, but before you could answer "yes" or "no“, you first had to check with your parents? Has someone asked you the answer to a math problem, but before you can give the answer, you had to do the problem yourself?

Asking parents, or doing the math problem, are examples of subqueries. In SQL, subqueries enable us to find the information we need so that we can get the information we want.

Subquery Overview

Throughout this course, you have written queries to extract data from a database. What if you wanted to write a query, only to find out you didn't have all the information you needed to construct it?

You can solve this problem by nesting queries—placing one query inside the other query. The inner query is called the "subquery." The subquery executes to find the information you don’t know. The outer query uses that information to find out what you need to know.

Subquery Overview (cont.)

Being able to combine two queries into one can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself.

Subquery Example

A subquery is a SELECT statement that is embedded in a clause of another SELECT statement. A subquery executes once before the main query. The result of the subquery is used by the main or outer query.

The subquery syntax is:

```
SELECT select_list
FROM table
WHERE expression operator
(SELECT select_list
FROM table);
```

The SELECT statement in parentheses is the inner query or 'subquery'. It executes first, before the outer query.

Subquery Example (cont.)

Subqueries can be placed in a number of SQL clauses, including the WHERE clause, the HAVING clause, and the FROM clause.

The subquery syntax is:

```
SELECT select_list
FROM table
WHERE expression operator
(SELECT select_list
FROM table);
```

The SELECT statement in parentheses is the inner query or 'subquery'. It executes first, before the outer query.

Guidelines for Using Subqueries

Guidelines:

- The subquery is enclosed in parentheses.
- The subquery is placed on the right side of the comparison condition.
- The outer and inner queries can get data from different tables.
- Only one ORDER BY clause can be used for a SELECT statement; if used, it must be the last clause in the outer query. A subquery cannot have its own ORDER BY clause.

Guidelines for Using Subqueries

Guidelines:

- The only limit on the number of subqueries is the buffer size the query uses.

Two Types of Subqueries

The two types of subqueries are:

- Single-row subqueries that use single-row operators (>, =, >=, <, <>, <=) and return only one row from the inner query.
- Multiple-row subqueries that use multiple-row operators (IN, ANY, ALL) and return more than one row from the inner query.

Subquery Example

What if you wanted to find out the names of the Global Fast Foods staff members that were born after Monique Tuttle? The first thing you need to know is the answer to the question, “When was Monique born?” Once you know her birth date, then you can select those staff members whose birth dates are after hers.

```
SELECT staff_id,
first_name, last_name,
birth_date
FROM f_staffs
WHERE birth_date >=
     (SELECT birth_date
   FROM f_staffs
   WHERE last_name =
‘Tuttle’);
```

STAFF_ID	FIRST_NAME	LAST_NAME	BIRTH_DATE
1000	Roger	Morgan	17-JUN-1987
1189	Nancy	Vickers	21-SEP-1989
1007	Monique	Tuttle	13-JAN-1987
1354	Alex	Hunter	03-JAN-1990
1423	Kathryn	Bassman	08-AUG-1991

Subquery and Null

If a subquery returns a null value or no rows, the outer query takes the results of the subquery (null) and uses this result in its WHERE clause.

The outer query will then return no rows, because comparing any value with a null always yields a null.

```
SELECT last_name
FROM employees
WHERE department_id =
    (SELECT department_id
     FROM employees
     WHERE last_name =
'Grant');
```

No data found.

Subquery and Null (cont.)

Who works in the same department as Grant? Grant's department_id is null. Thus, the outer query does not even return Grant's row because comparing anything with a null returns a null.

```
SELECT last_name
FROM employees
WHERE department_id =
    (SELECT department_id
     FROM employees
     WHERE last_name =
'Grant');
```

No data found.

Multiple-Column Subqueries

Subqueries can use one or more columns. If they use more than one column, they are called multiple-column subqueries. A multiple-column subquery can be either pair-wise comparisons or non-pair-wise comparisons.

```
SELECT   employee_id,
         manager_id,
         department_id
FROM     employees
WHERE
(manager_id,department_id) IN
(SELECT manager_id,department_id
           FROM   employees
           WHERE  employee_id IN
(149,174))
AND employee_id NOT IN (149,174)
```

EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
176	149	80

Multiple-Column Subqueries (cont.)

The example on the right shows a multiple-column pair-wise subquery with the subquery highlighted in red and the result in the table below. The query lists the employees whose manager and departments are the same as the manager and department of employees 149 or 174.

```
SELECT   employee_id,
         manager_id,
         department_id
FROM     employees
WHERE
(manager_id,department_id) IN
(SELECT manager_id,department_id
           FROM   employees
           WHERE  employee_id IN
(149,174))
AND employee_id NOT IN (149,174)
```

EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
176	149	80

Multiple-Column Subqueries (cont.)

A non-pair-wise multiple-column subquery also uses more than one column in the subquery, but it compares them one at a time, so the comparisons take place in different subqueries.

```
SELECT  employee_id,
        manager_id,
        department_id
FROM    employees
WHERE  manager_id IN
      (SELECT  manager_id
       FROM    employees
       WHERE   employee_id IN
                 (174,199))
AND    department_id IN
       (SELECT  department_id
        FROM    employees
        WHERE   employee_id IN
                  (174,199))
AND  employee_id NOT IN(174,199);
```

EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
176	149	80
149	100	80

Multiple-Column Subqueries (cont.)

You will need to write one subquery per column you want to compare against when performing non-pair-wise multiple column subqueries.

```
SELECT  employee_id,
        manager_id,
        department_id
FROM    employees
WHERE  manager_id IN
      (SELECT  manager_id
       FROM    employees
       WHERE   employee_id IN
                 (174,199))
AND    department_id IN
       (SELECT  department_id
        FROM    employees
        WHERE   employee_id IN
                  (174,199))
AND  employee_id NOT IN(174,199);
```

EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
176	149	80
149	100	80

Multiple-Column Subqueries (cont.)

The example on the right shows a multiple-column non-pair-wise subquery with the subqueries highlighted in red. This query is listing the employees who have either a manager_id or a department_id in common with employees 174 or 199.

```
SELECT  employee_id,
        manager_id,
        department_id
FROM    employees
WHERE  manager_id IN
      (SELECT  manager_id
       FROM    employees
       WHERE   employee_id IN
                 (174,199))
AND    department_id IN
       (SELECT  department_id
        FROM    employees
        WHERE   employee_id IN
                  (174,199))
AND  employee_id NOT IN(174,199);
```

EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
176	149	80
149	100	80

EXISTS & NOT EXISTS in Subqueries

EXISTS, and its opposite NOT EXISTS, are two clauses that can be used when testing for matches in subqueries. EXISTS tests for a TRUE, or a matching result in the subquery.

To answer the question, “Which employees are not managers?”, you first have to ask, “Who are the managers?” and then ask, “Who does NOT EXIST on the managers list?”

EXISTS & NOT EXISTS in Subqueries (cont.)

In this example, the subquery is selecting a NULL value (no columns) to emphasize the fact that we do not wish to view any column data—we are testing for a quantity of rows in the subquery. This query will work whether we select one column, all (*) columns, or no columns.

```
SELECT count(*)
FROM   employees t1
WHERE  NOT EXISTS (SELECT NULL
                   FROM   employees t2
                   WHERE  t2.manager_id = t1.employee_id );
```

COUNT(*)
12

EXISTS & NOT EXISTS in Subqueries (cont.)

If the same query is executed with a NOT IN instead of NOT EXISTS, the result is very different. The result of this query suggests there are no employees who are also not managers, so all employees are managers, which we already know is not true. What is causing this result?

```
SELECT count(*)
FROM   employees t1
WHERE  t1.employee_id NOT IN (SELECT t2.manager_id
                              FROM   employees t2 );
```

COUNT(*)
0

EXISTS & NOT EXISTS in Subqueries (cont.)

The cause of the strange result is due to the NULL value returned by the subquery. One of the rows in the employees table does not have a manager, and this makes the entire result wrong. Subqueries can return three values: TRUE, FALSE, and UNKNOWN. A NULL in the subquery result set will return an UNKNOWN, which Oracle cannot evaluate, so it doesn't.

```
SELECT count(*)
FROM   employees t1
WHERE  t1.employee_id NOT IN (SELECT t2.manager_id
                              FROM   employees t2 );
```

COUNT(*)
0

EXISTS & NOT EXISTS in Subqueries (cont.)

BEWARE of NULLS in subqueries when using IN or NOT IN.

If you are unsure whether or not a subquery will include a null value, either eliminate the null by using IS NOT NULL in a WHERE clause (for example, “WHERE t2.manager_id IS NOT NULL) or use NOT EXISTS to be safe.

EXISTS & NOT EXISTS in Subqueries (cont.)

The way EXISTS and NOT EXISTS are executed by Oracle is very different from the way IN and NOT IN are executed.

When the database is executing an IN with a Subquery, it evaluates the Subquery, typically Distincts that query, and then joins the result to the outer query to get the result.

EXISTS & NOT EXISTS in Subqueries (cont.)

When an EXISTS is executed, it performs a Full Table Scan of the outer table, and then Oracle loops through the Subquery result set row by row to see if the condition is true. It executes like this:

```
FOR  x IN ( SELECT * FROM employees t1 ) LOOP
     IF ( EXISTS ( SELECT NULL FROM employees t2 WHERE
t1.emplolyee_id = t2.manager_id ) THEN
          OUTPUT THE RECORD
     END IF
END LOOP
```

EXISTS & NOT EXISTS in Subqueries (cont.)

One final note about EXISTS and IN. IN will generally execute faster, as it can use any existing indexes on the outer table. EXISTS cannot use indexes on the outer table. The size of the table is also an important factor and affects which of the expressions will run faster.

A small outer table joined to a very big inner table can still execute very fast using EXISTS, perhaps even faster than the same statement using IN.

Remember: be careful when handling NULLs.

Terminology

Key terms used in this lesson included:

- EXIST and NOT EXIST
- Inner query
- Multiple-row subquery
- Non-pair-wise multiple column subquery
- Outer subquery
- Pair-wise multiple column subquery
- Single-row subquery
- Subquery

Summary

In this lesson, you should have learned how to:

- Define and explain the purpose of subqueries for retrieving data
- Construct and execute a single-row subquery in the WHERE clause
- Distinguish between single-row and multiple-row subqueries
- Distinguish between pair-wise and non-pair-wise subqueries
- Use the EXIST and NOT EXISTS operators in a query