7

Multiple-Column Subqueries

ORACLE®

Schedule:	Timing	Topic
	20 minutes	Lecture
	20 minutes	Practice
	40 minutes	Total

Objectives

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

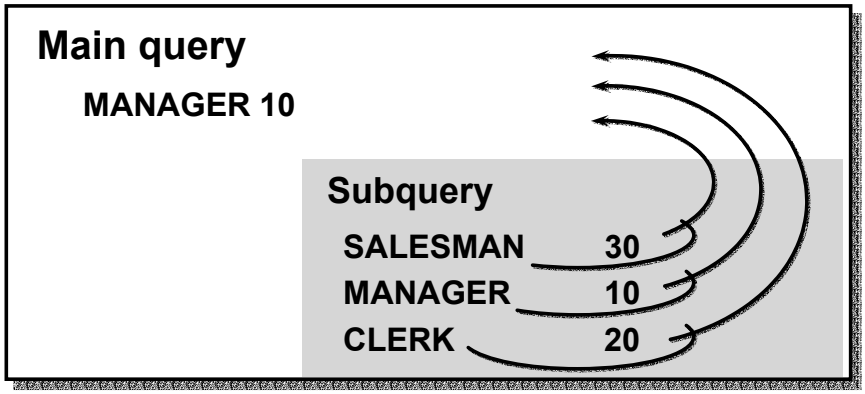
- **Write a multiple-column subquery**
- **Describe and explain the behavior of subqueries when null values are retrieved**
- **Write a subquery in a FROM clause**

ORACLE®

Lesson Aim

In this lesson, you will learn how to write multiple-column subqueries and subqueries in the FROM clause of a SELECT statement.

Multiple-Column Subqueries



Main query compares **to** **Values from a multiple-row and multiple-column subquery**

MANAGER 10

SALESMAN 30
MANAGER 10
CLERK 20

ORACLE®

Multiple-Column Subqueries

So far you have written single-row subqueries and multiple-row subqueries where only one column was compared in the WHERE clause or HAVING clause of the SELECT statement. If you want to compare two or more columns, you must write a compound WHERE clause using logical operators. Multiple-column subqueries enable you to combine duplicate WHERE conditions into a single WHERE clause.

Syntax

```
SELECT   column, column, ...
FROM     table
WHERE    (column, column, ...) IN
                              (SELECT column, column, ...
                                FROM   table
                                WHERE  condition);
```

Using Multiple-Column Subqueries

Display the order id, product id, and quantity of items in the item table that match both the product id and quantity of an item in order 605.

```
SQL> SELECT   ordid, prodid, qty
  2  FROM     item
  3  WHERE    (prodid, qty) IN
  4                          (SELECT prodid, qty
  5                           FROM   item
  6                           WHERE  ordid = 605)
  7  AND      ordid <> 605;
```


ORACLE®

Using Multiple-Column Subqueries

The example on the slide is that of a multiple-column subquery because the subquery returns more than one column. It compares the values in the PRODID column and the QTY column of each candidate row in the ITEM table to the values in the PRODID column and QTY column for items in order 605.

First, execute the subquery to see the PRODID and QTY values for each item in order 605.

```
SQL> SELECT prodid, qty
  2  FROM   item
  3  WHERE  ordid = 605;

   PRODID        QTY
---------- ---------
    100861       100
    100870       500
    100890         5
    101860        50
    101863       100
    102130        10

6 rows selected.
```

Using Multiple-Column Subqueries

Display the order number, product number, and quantity of any item in which the product number and quantity match both the product number and quantity of an item in order 605.

```
SQL> SELECT   ordid, prodid, qty
  2  FROM     item
  3  WHERE    (prodid, qty) IN
  4                          (SELECT prodid, qty
  5                           FROM   item
  6                           WHERE  ordid = 605)
  7  AND      ordid <> 605;
```


ORACLE®

Using Multiple-Column Subqueries (continued)

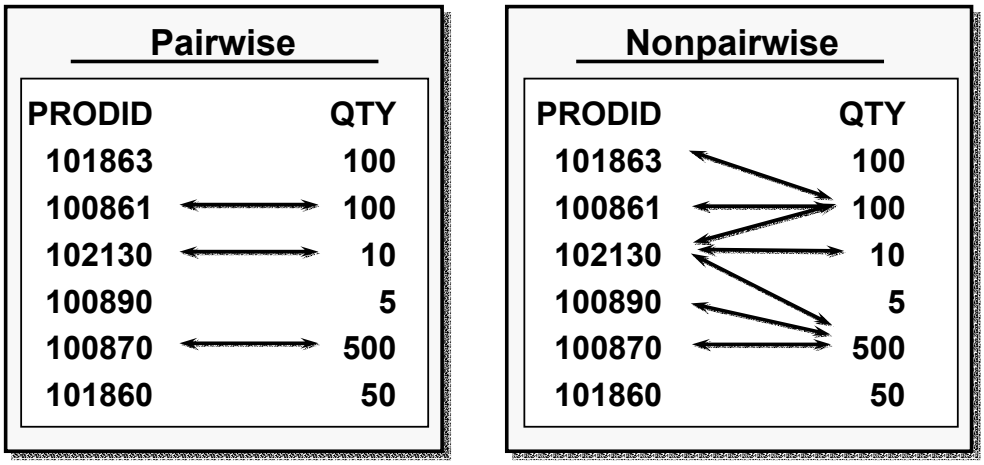
When the SQL statement on the slide is executed, the Oracle server compares the values in both the PRODID and QTY columns and returns those orders where the product number and quantity for *that* product match *both* the product number and quantity for an item in order 605.

The output of the SQL statement is:

```
    ORDID     PRODID       QTY
--------- --------- ---------
      617    100861       100
      617    100870       500
      616    102130        10
```

The output shows that there are three items in other orders that contain the same product number and quantity as an item in order 605. For example, order 617 has ordered a quantity 500 of product 100870. Order 605 has also ordered a quantity 500 of product 100870. Therefore, these candidate rows are part of the output.

Column Comparisons



ORACLE®

Pairwise Versus Nonpairwise Comparisons

Column comparisons in a multiple-column subquery can be pairwise comparisons or nonpairwise comparisons.

The slide shows the product numbers and quantities of the items in order 605.

In the example on the previous slide, a pairwise comparison was executed in the WHERE clause. Each candidate row in the SELECT statement must have *both* the same product number and same quantity as an item in order 605. This is illustrated on the left side of the slide above. The arrows indicate that both the product number and quantity in a candidate row match a product number and quantity of an item in order 605.

A multiple-column subquery can also be a nonpairwise comparison. If you want a nonpairwise comparison (a cross product), you must use a WHERE clause with multiple conditions. A candidate row must match the multiple conditions in the WHERE clause but the values are compared individually. A candidate row must match some product number in order 605 as well as some quantity in order 605, but these values do not need to be in the same row. This is illustrated on the right side of the slide. For example, product 102130 appears in other orders, one order matching the quantity in order 605 (10), and another order having a quantity of 500. The arrows show a sampling of the various quantities ordered for a particular product.

Nonpairwise Comparison Subquery

Display the order number, product number, and quantity of any item in which the product number and quantity match any product number and any quantity of an item in order 605.

```
SQL> SELECT   ordid, prodid, qty
  2  FROM     item
  3  WHERE    prodid IN  (SELECT        prodid
  4                       FROM         item
  5                       WHERE        ordid = 605)
  6  AND      qty    IN  (SELECT        qty
  7                       FROM         item
  8                       WHERE        ordid = 605)
  9  AND      ordid <> 605;
```


ORACLE®

Nonpairwise Comparison Subquery

The slide example does a nonpairwise comparison of the columns. It displays the order number, product number, and quantity of any item in which the product number and quantity match any product number and quantity of an item in order 605. Order 605 is not included in the output.

Nonpairwise Subquery

```
    ORDID     PRODID        QTY
--------- --------- ---------
      609    100870          5
      616    100861         10
      616    102130         10
      621    100861         10
      618    100870         10
      618    100861         50
      616    100870         50
      617    100861        100
      619    102130        100
      615    100870        100
      617    101860        100
      621    100870        100
      617    102130        100
      . . .
16 rows selected.
```


ORACLE®

Nonpairwise Subquery

The results of the nonpairwise subquery are shown in the slide. Sixteen candidate rows in the ITEM table match the multiple conditions in the WHERE clause.

For example, an item from order 621 is returned from the SQL statement. A product in order 621 (product number 100861) matches a product in an item in order 605. The quantity for product 100861 in order 621 (10) matches the quantity in another item in order 605 (the quantity for product 102130).

Null Values in a Subquery

```
SQL> SELECT   employee.ename
  2  FROM     emp employee
  3  WHERE    employee.empno NOT IN
  4                          (SELECT manager.mgr
  5                           FROM   emp manager);
no rows selected.
```


ORACLE®

Returning Nulls in the Resulting Set of a Subquery

The SQL statement on the slide attempts to display all the employees who do not have any subordinates. Logically, this SQL statement should have returned eight rows. However, the SQL statement does not return any rows. One of the values returned by the inner query is a null value and hence the entire query returns no rows. The reason is that all conditions that compare a null value result in a null. So whenever null values are likely to be part of the resultant set of a subquery, do not use the NOT IN operator. The NOT IN operator is equivalent to !=ALL.

Notice that the null value as part of the resultant set of a subquery will not be a problem if you are using the IN operator. The IN operator is equivalent to =ANY. For example, to display the employees who have subordinates, use the following SQL statement:

```
SQL> SELECT    employee.ename
  2  FROM      emp employee
  3  WHERE     employee.empno IN (SELECT manager.mgr
  4                               FROM  emp manager);

ENAME
----------
KING
...
6 rows selected.
```

Using a Subquery in the FROM Clause

```
SQL> SELECT  a.ename, a.sal, a.deptno, b.salavg
  2  FROM    emp a, (SELECT   deptno, avg(sal) salavg
  3                  FROM     emp
  4                  GROUP BY deptno) b
  5  WHERE   a.deptno = b.deptno
  6  AND     a.sal > b.salavg;
```

```
ENAME             SAL     DEPTNO     SALAVG
---------- --------- --------- ----------
KING             5000         10  2916.6667
JONES            2975         20       2175
SCOTT            3000         20       2175
...
6 rows selected.
```


ORACLE®

Using a Subquery in the FROM Clause

You can use a subquery in the FROM clause of a SELECT statement, which is very similar to how views are used. A subquery in the FROM clause of a SELECT statement defines a data source for that particular SELECT statement, and only that SELECT statement. The slide example displays employee names, salaries, department numbers, and average salaries for all the employees who make more than the average salary in their department.

Summary

- A multiple-column subquery returns more than one column.
- Column comparisons in multiple-column comparisons can be pairwise or nonpairwise.
- A multiple-column subquery can also be used in the FROM clause of a SELECT statement.

ORACLE®

Summary

Multiple-column subqueries enable you to combine duplicate WHERE conditions into a single WHERE clause. Column comparisons in a multiple-column subquery can be pairwise comparisons or nonpairwise comparisons. You can use a subquery to define a table to be operated on by a containing query. You do this by placing the subquery in the FROM clause of the containing query as you would a table name.

Practice Overview

Creating multiple-column subqueries

ORACLE®

Practice Overview

In this practice, you will write multiple-value subqueries.