# ORACLE® ACADEMY

# Database Programming with SQL

**10-3**
**Multiple-Row Subqueries**

# Objectives

This lesson covers the following objectives:

- Correctly use the comparison operators IN, ANY, and ALL in multiple-row subqueries

- Construct and execute a multiple-row subquery in the WHERE clause or HAVING clause

- Describe what happens if a multiple-row subquery returns a null value

- Understand when multiple-row subqueries should be used, and when it is safe to use a single-row subquery

# Objectives

This lesson covers the following objectives:

- Distinguish between pair-wise and non-pair-wise subqueries

- Create a query using the EXISTS and NOT EXISTS operators to test for returned rows from the subquery

# Purpose

- A subquery is designed to find information you don't know so that you can find information you want to know.

- However, single-row subqueries can return only one row. What if you need to find information based on several rows and several values?

- The subquery will need to return several rows.

- We achieve this using multiple-row subqueries and the three comparison operators: IN, ANY, and ALL.

# Query Comparison

- Whose salary is equal to the salary of an employee in department 20 ?

| LAST_NAME | DEPT_ID | SALARY |
|-----------|---------|--------|
| Hartstein | 20 | 13000 |
| Fay | 20 | 6000 |

- This example returns an error because more than one employee exists in department 20, the subquery returns multiple rows.

- We call this a multiple-row subquery.

```
SELECT first_name, last_name
FROM employees
WHERE salary =
    (SELECT salary
     FROM employees
     WHERE department_id = 20);
```

**ORA-01427:  single-row subquery returns more than one row**

ORACLE® ACADEMY

# Query Comparison

- The problem is the equal sign (=) in the WHERE clause of the outer query.

- How can one value be equal to (or not equal to) more than one value at a time?

- It's a silly question, isn't it?

```
SELECT first_name, last_name
FROM employees
WHERE salary =
    (SELECT salary
     FROM employees
     WHERE department_id = 20);
```

**ORA-01427:  single-row subquery returns more than one row**

# IN, ANY, and ALL

- Subqueries that return more than one value are called multiple-row subqueries.

- Because we cannot use the single-row comparison operators (=, <, and so on), we need different comparison operators for multiple-row subqueries.

- The multiple-row operators are:
  - IN,
  - ANY
  - ALL

- The NOT operator can be used with any of these three operators.

8

**ORACLE** **ACADEMY**

# IN

- The IN operator is used within the outer query WHERE clause to select only those rows which are IN the list of values returned from the inner query.

- For example, we are interested in all the employees that were hired the same year as an employee in department 90.

```
SELECT last_name, hire_date
FROM employees
WHERE EXTRACT(YEAR FROM hire_date) IN
    (SELECT EXTRACT(YEAR FROM hire_date)
     FROM employees
     WHERE department_id=90);
```

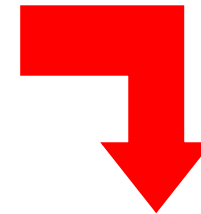| LAST_NAME | HIRE_DATE |
|-----------|-----------|
| King | 17-Jun-1987 |
| Kochhar | 21-Sep-1989 |
| De Haan | 13-Jan-1993 |
| Whalen | 17-Sep-1987 |

ORACLE ACADEMY

# IN

- The inner query will return a list of the years that employees in department 90 were hired.

- The outer query will then return any employee that was hired the same year as any year in the inner query list.

```
SELECT last_name, hire_date
FROM employees
WHERE EXTRACT(YEAR FROM hire_date) IN
   (SELECT EXTRACT(YEAR FROM hire_date)
    FROM employees
    WHERE department_id=90);
```

| LAST_NAME | HIRE_DATE |
|-----------|-----------|
| King | 17-Jun-1987 |
| Kochhar | 21-Sep-1989 |
| De Haan | 13-Jan-1993 |
| Whalen | 17-Sep-1987 |

# ANY

- The ANY operator is used when we want the outer-query WHERE clause to select the rows which match the criteria (<, >, =, etc.) of **at least** one value in the subquery result set.

- The example shown will return any employee whose year hired is less than at least one year hired of employees in department 90.

| Year Hired |
|---|
| 1987 |
| 1989 |
| 1993 |

```
SELECT last_name, hire_date
FROM employees
WHERE EXTRACT(YEAR FROM hire_date) < ANY
  (SELECT EXTRACT(YEAR FROM hire_date)
   FROM employees
   WHERE department_id=90);
```

| LAST_NAME | HIRE_DATE |
|---|---|
| King | 17-Jun-1987 |
| Kochhar | 21-Sep-1989 |
| Whalen | 17-Sep-1987 |
| Hunold | 03-Jan-1990 |
| Ernst | 21-May-1991 |

# ALL

- The ALL operator is used when we want the outer-query WHERE clause to select the rows which match the criteria ( <, >, =, etc.) of **all** of the values in the subquery result set.

- The ALL operator compares a value to every value returned by the inner query.

- As no employee was hired before 1987, no rows are returned.

```
SELECT last_name, hire_date
FROM employees
WHERE EXTRACT(YEAR FROM hire_date) < ALL
  (SELECT EXTRACT(YEAR FROM hire_date)
   FROM employees
   WHERE department_id=90);
```
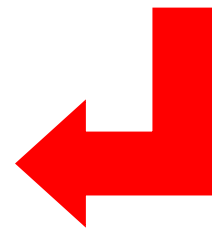
| Year Hired |
|------------|
| 1987 |
| 1989 |
| 1993 |

no data found

# NULL Values

- Suppose that one of the values returned by a multiple-row subquery is null, but other values are not.

- If IN or ANY are used, the outer query will return rows which match the non-null values.

```
SELECT last_name, employee_id
FROM employees
WHERE employee_id IN
    (SELECT manager_id
     FROM employees);
```

| MANAGER_ID |
|------------|
| -          |
| 100        |
| 100        |
| 101        |
| 101        |
| 205        |
| 100        |
| …          |

**Result of subquery**

| LAST_NAME | EMPLOYEE_ID |
|-----------|-------------|
| King      | 100         |
| Kochhar   | 101         |
| De Haan   | 102         |
| Higgins   | 205         |
| …         |             |

ORACLE® ACADEMY

# NULL Values

- If ALL is used, the outer query returns no rows because ALL compares the outer query row with every value returned by the subquery, including the null.

- And comparing anything with null results in null.

```
SELECT last_name, employee_id
FROM employees
WHERE employee_id <= ALL
    (SELECT manager_id
     FROM employees);
```

no data found

# GROUP BY and HAVING

- As you might suspect, the GROUP BY clause and the HAVING clause can also be used with multiple-row subqueries.

- What if you wanted to find the departments whose minimum salary is less than the salary of any employee who works in department 10 or 20?

| LAST_NAME | DEPT_ID | SALARY |
|-----------|---------|--------|
| Whalen | 10 | 4400 |
| Hartstein | 20 | 13000 |
| Fay | 20 | 6000 |

| DEPARTMENT_ID | MIN(SALARY) |
|---------------|-------------|
| 10 | 4400 |
| 20 | 6000 |
| 50 | 2500 |
| 60 | 4200 |
| 80 | 8600 |
| 110 | 8300 |
| (null) | 7000 |

# GROUP BY and HAVING

- We need a multiple-row subquery which returns the salaries of employees in departments 10 and 20.

- The outer query will use a group function (MIN) so we need to GROUP the outer query BY department_id.

| LAST_NAME | DEPT_ID | SALARY |
|-----------|---------|--------|
| Whalen | 10 | 4400 |
| Hartstein | 20 | 13000 |
| Fay | 20 | 6000 |

| DEPARTMENT_ID | MIN(SALARY) |
|---------------|-------------|
| 10 | 4400 |
| 20 | 6000 |
| 50 | 2500 |
| 60 | 4200 |
| 80 | 8600 |
| 110 | 8300 |
| (null) | 7000 |

ORACLE® **ACADEMY**

# GROUP BY and HAVING

- Here is the SQL statement:

```
SELECT department_id, MIN(salary)
FROM employees
GROUP BY department_id
HAVING MIN(salary) < ANY
   (SELECT salary
    FROM employees
    WHERE department_id IN (10,20))
ORDER BY department_id;
```

| DEPARTMENT_ID | MIN(SALARY) |
|---|---|
| 10 | 4400 |
| 20 | 6000 |
| 50 | 2500 |
| 60 | 4200 |
| 80 | 8600 |
| 110 | 8300 |
| - | 7000 |

| LAST_NAME | DEPT_ID | SALARY |
|---|---|---|
| Whalen | 10 | 4400 |
| Hartstein | 20 | 13000 |
| Fay | 20 | 6000 |

**Result of subquery**

# 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

- The example below 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 |

ORACLE **ACADEMY**

# Multiple-Column Subqueries

- 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
                 (149,174))
AND     department_id IN
        (SELECT  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            |

ORACLE® ACADEMY

# Multiple-Column Subqueries

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

- The example on the right shows a multiple-column non-pair-wise subquery with the subqueries highlighted in red.

```
SELECT   employee_id,
         manager_id,
         department_id
FROM     employees
WHERE    manager_id IN
         (SELECT   manager_id
          FROM     employees
          WHERE    employee_id IN
                   (149,174))
AND      department_id IN
         (SELECT   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

- This query is listing the employees who have the same manager_id and department_id as employees 149 or 174.

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

**Result of 1st subquery**

| MANAGER_ID |
|------------|
| 100 |
| 149 |

**Result of 2nd subquery**

| DEPARTMENT_ID |
|---------------|
| 80 |
| 80 |

| EMPLOYEE_ID | MANAGER_ID | DEPARTMENT_ID |
|-------------|------------|---------------|
| 176 | 149 | 80 |

ORACLE® ACADEMY

# 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?"

**ORACLE® ACADEMY**

# EXISTS & NOT EXISTS in Subqueries

- In this example, the subquery is selecting the employees that are managers.

- The outer query then returns the rows from the employee table that do NOT EXIST in the subquery.

```
SELECT last_name AS "Not a Manager"
FROM    employees emp
WHERE NOT EXISTS
 (SELECT *
  FROM employees mgr
  WHERE  mgr.manager_id = emp.employee_id);
```

| Not a Manager |
|---|
| Whalen |
| Gietz |
| Abel |
| Taylor |
| Grant |
| Rajs |
| Davies |
| Matos |
| Vargas |
| Ernst |
| ... |

# EXISTS & NOT EXISTS in Subqueries

- 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.

```
SELECT last_name AS "Not a Manager"
FROM    employees emp
WHERE emp.employee_id NOT IN
  (SELECT mgr.manager_id
   FROM employees mgr);
```

no data found

# EXISTS & NOT EXISTS in Subqueries

- 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 last_name AS "Not a Manager"
FROM    employees emp
WHERE emp.employee_id NOT IN
  (SELECT mgr.manager_id
   FROM employees mgr);
```

no data found

# EXISTS & NOT EXISTS in Subqueries

- 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 emp.manager_id  IS NOT NULL  or use NOT EXISTS  to be safe.

# One Last Point About Subqueries

- Some subqueries may return a single row or multiple rows, depending on the data values in the rows.

- If even the slightest possibility exists of returning multiple rows, make sure you write a multiple-row subquery.

```
SELECT first_name, last_name, job_id
FROM employees
WHERE job_id =
    (SELECT job_id
     FROM employees
     WHERE last_name = 'Ernst');
```

| FIRST_NAME | LAST_NAME | JOB_ID |
|------------|-----------|--------|
| Alexander | Hunold | IT_PROG |
| Bruce | Ernst | IT_PROG |
| Diana | Lorentz | IT_PROG |

| FIRST_NAME | LAST_NAME | JOB_ID |
|------------|-----------|--------|
| Bruce | Ernst | IT_PROG |

**Result of subquery**

# One Last Point About Subqueries

- For example: Who has the same job_id as Ernst?

- This single-row subquery works correctly because there is only one Ernst in the table.

- But what if later, the business hires a new employee named Susan Ernst?

```
SELECT first_name, last_name, job_id
FROM employees
WHERE job_id =
    (SELECT job_id
     FROM employees
     WHERE last_name = 'Ernst');
```

| FIRST_NAME | LAST_NAME | JOB_ID |
|------------|-----------|--------|
| Alexander | Hunold | IT_PROG |
| Bruce | Ernst | IT_PROG |
| Diana | Lorentz | IT_PROG |

| FIRST_NAME | LAST_NAME | JOB_ID |
|------------|-----------|--------|
| Bruce | Ernst | IT_PROG |

**Result of subquery**

# One Last Point About Subqueries

- It would be better to write a multiple-row subquery.

- The multiple-row subquery syntax will still work even if the subquery returns a single row.

- If in doubt, write a multiple-row subquery!

```
SELECT first_name, last_name, job_id
FROM employees
WHERE job_id IN
    (SELECT job_id
     FROM employees
     WHERE last_name = 'Ernst');
```

| FIRST_NAME | LAST_NAME | JOB_ID |
|---|---|---|
| Alexander | Hunold | IT_PROG |
| Bruce | Ernst | IT_PROG |
| Diana | Lorentz | IT_PROG |

| FIRST_NAME | LAST_NAME | JOB_ID |
|---|---|---|
| Bruce | Ernst | IT_PROG |

**Result of subquery**
**There are 2 people with last name 'Ernst'**

# Terminology

Key terms used in this lesson included:

- EXIST and NOT EXIST

- Non-pair-wise multiple column subquery

- Pair-wise multiple column subquery

# Summary

In this lesson, you should have learned how to:

- Correctly use the comparison operators IN, ANY, and ALL in multiple-row subqueries

- Construct and execute a multiple-row subquery in the WHERE clause or HAVING clause

- Describe what happens if a multiple-row subquery returns a null value

- Understand when multiple-row subqueries should be used, and when it is safe to use a single-row subquery

# Summary

In this lesson, you should have learned how to:

- Distinguish between pair-wise and non-pair-wise subqueries
- Create a query using the EXISTS and NOT EXISTS operators to test for returned rows from the subquery