# Perancangan Basis Data(IFWP1007)
# 3 sks

UKRIDA
Universitas Kristen Krida Wacana

Major: Informatics
Topic: Fundamentals of Subqueries


Dosen Pengampu

**Rita Wiryasaputra, ST., M. Cs., Ph.D.**

**Email: rita.wiryasaputra@ukrida.ac.id**


## Skills

# 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

# 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 a "subquery."

# Subquery Overview

- 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.
- 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 Overview

- 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.
- 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.
  - 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 employees that were hired after Peter Vargas?
- The first thing you need to know is the answer to the question, "When was Peter Vargas hired?"
- Once you know his hire date, then you can select those employees whose hire dates are after his.

```
SELECT first_name, last_name, hire_date
FROM employees WHERE hire_date > (SELECT
hire_date FROM employees
  WHERE last_name = 'Vargas');
```

| FIRST_NAME | LAST_NAME | HIRE_DATE |
|---|---|---|
| Eleni | Zlotkey | 29-Jan-2000 |
| Kimberely | Grant | 24-May-1999 |
| Kevin | Mourgos | 16-Nov-1999 |
| Diana | Lorentz | 07-Feb-1999 |

# 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

- Who works in the same department as Grant?
- Grant's department_id is null, so the subquery returns NULL.
- The outer query then substitutes this value in the WHERE clause (WHERE department_id = NULL).
- The outer query returns no rows, 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

# Objectives

This lesson covers the following objectives:

- Construct and execute a single-row subquery in the WHERE clause or HAVING clause
- Construct and execute a SELECT statement using more than one subquery
- Construct and execute a SELECT statement using a group function in the subquery

# Purpose

- As you have probably realized, subqueries are a lot like Internet search engines.
- They are great at locating the information needed to accomplish another task.
- In this lesson, you will learn how to create even more complicated tasks for subqueries to do for you.
- Keep in mind that subqueries save time in that you can accomplish two tasks in one statement.

# Facts About Single-row Subqueries

- They:
  - Return only one row
  - Use single-row comparison operators (=, >,>=, <, <=, <>)
- Always:
  - Enclose the subquery in parentheses.
  - Place the subquery on the right hand side of the comparison condition.

# Additional Subquery Facts

- The outer and inner queries can get data from different tables.
- Only one ORDER BY clause can be used for a SELECT statement, and if specified, it must be the last clause in the main SELECT statement.
- The only limit on the number of subqueries is the buffer size that the query uses.

# Subqueries from Different Tables

- The outer and inner queries can get data from different tables.
- Who works in the Marketing department?

```
SELECT last_name, job_id, department_id FROM
employees
WHERE department_id = (SELECT department_id FROM
departments WHERE department_name = 'Marketing')
ORDER BY job_id;
```

| LAST_NAME | JOB_ID | DEPARTMENT_ID |
|---|---|---|
| Hartstein | MK_MAN | 20 |
| Fay | MK_REP | 20 |

Result of subquery

| DEPARTMENT_ID |
|---|
| 20 |

# Subqueries from Different Tables

- More than one subquery can return information to the outer query.

```
SELECT last_name, job_id, salary, department_id FROM
employees
WHERE job_id = (SELECT job_id FROM employees
    WHERE employee_id = 141) AND department_id =
   (SELECT department_id FROM departments
    WHERE location_id = 1500);
```

Result of 1st subquery

| JOB_ID |
|---|
| ST_CLERK |

Result of 2nd subquery

| DEPARTMENT_ID |
|---|
| 50 |

| LAST_NAME | JOB_ID | SALARY | DEPARTMENT_ID |
|---|---|---|---|
| Rajs | ST_CLERK | 3500 | 50 |
| Davies | ST_CLERK | 3100 | 50 |
| Matos | ST_CLERK | 2600 | 50 |
| Vargas | ST_CLERK | 2500 | 50 |

# Group Functions in Subqueries

- Group functions can be used in subqueries.
- A group function without a GROUP BY clause in the subquery returns a single row.
- The query on the next slide answers the question, "Which employees earn less than the average salary?"

# Group Functions in Subqueries

- The subquery first finds the average salary for all employees, the outer query then returns employees with a salary of less than the average.

```
SELECT last_name, salary FROM employees
WHERE salary <  (SELECT AVG(salary) FROM
    employees);
```

Result of subquery

| AVG(SALARY) |
|---|
| 8775 |

| LAST_NAME | SALARY |
|---|---|
| Whalen | 4400 |
| Gietz | 8300 |
| Taylor | 8600 |
| Grant | 7000 |
| Mourgos | 5800 |
| Rajs | 3500 |
| Davies | 3100 |
| Matos | 2600 |
| Vargas | 2500 |
| Ernst | 6000 |
| Lorentz | 4200 |
| Fay | 6000 |

# Subqueries in the HAVING Clause

- Subqueries can also be placed in the HAVING clause.
- Remember that the HAVING clause is similar to the WHERE clause, except that the HAVING clause is used to restrict groups and always includes a group function such as MIN, MAX, or AVG.
- Because the HAVING clause always includes a group function, the subquery will nearly always include a group function as well.

# Subquery Example

- Which departments have a lowest salary that is greater than the lowest salary in department 50?
- In this example, the subquery selects and returns the lowest salary in department 50.

```
SELECT department_id, MIN(salary)
FROM employees GROUP BY department_id
HAVING MIN(salary) >(SELECT
MIN(salary) FROM employees WHERE
department_id = 50);
```

| DEPARTMENT_ID | MIN(SALARY) |
|---|---|
| - | 7000 |
| 90 | 17000 |
| 20 | 6000 |
| 110 | 8300 |
| 80 | 8600 |
| 10 | 4400 |
| 60 | 4200 |

Result of subquery

| MIN(SALARY) |
|---|
| 2500 |

# Subquery Example

- The outer query uses this value to select the department ID and lowest salaries of all the departments whose lowest salary is greater than that number.
- The HAVING clause eliminated those departments whose MIN salary was less than department 50's MIN salary.

```
SELECT department_id, MIN(salary)
FROM employees GROUP BY department_id
HAVING MIN(salary) >
(SELECT MIN(salary) FROM employees
WHERE department_id = 50);
```

| DEPARTMENT_ID | MIN(SALARY) |
|---|---|
| - | 7000 |
| 90 | 17000 |
| 20 | 6000 |
| 110 | 8300 |
| 80 | 8600 |
| 10 | 4400 |
| 60 | 4200 |

| MIN(SALARY) |
|---|
| 2500 |

Result of subquery

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

# 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 ?
- 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.

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

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

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

# 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.

# 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 |

# 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.

```
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);
```

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

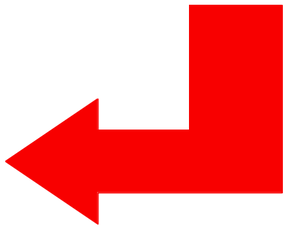
| LAST_NAME | HIRE_DATE |
|---|---|
| King | 17-Jun-1987 |
| Kochhar | 21-Sep-1989 |
| Whalen | 17-Sep-1987 |
| Hunold | 03-Jan-1990 |
| Ernst | 21-May-1991 |

# 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);
```

no data found



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

# 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 |
| … | |

# 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 |

# 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;
```

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

**Result of subquery**

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

# 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.

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

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

# 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 |

# 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
          FROMemployees 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

- 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
         FROMemployees 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
        FROMemployees 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 |

# 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
    employeesWHERE 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 |
|---|---|---|
| Bruce | Ernst | IT_PROG |

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

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

# Objectives

This lesson covers the following objectives:

- Identify when correlated subqueries are needed.
- Construct and execute correlated subqueries.
- Create a query using the EXISTS and NOT EXISTS operators to test for returned rows from the subquery
- Construct and execute named subqueries using the WITH clause.

# Purpose

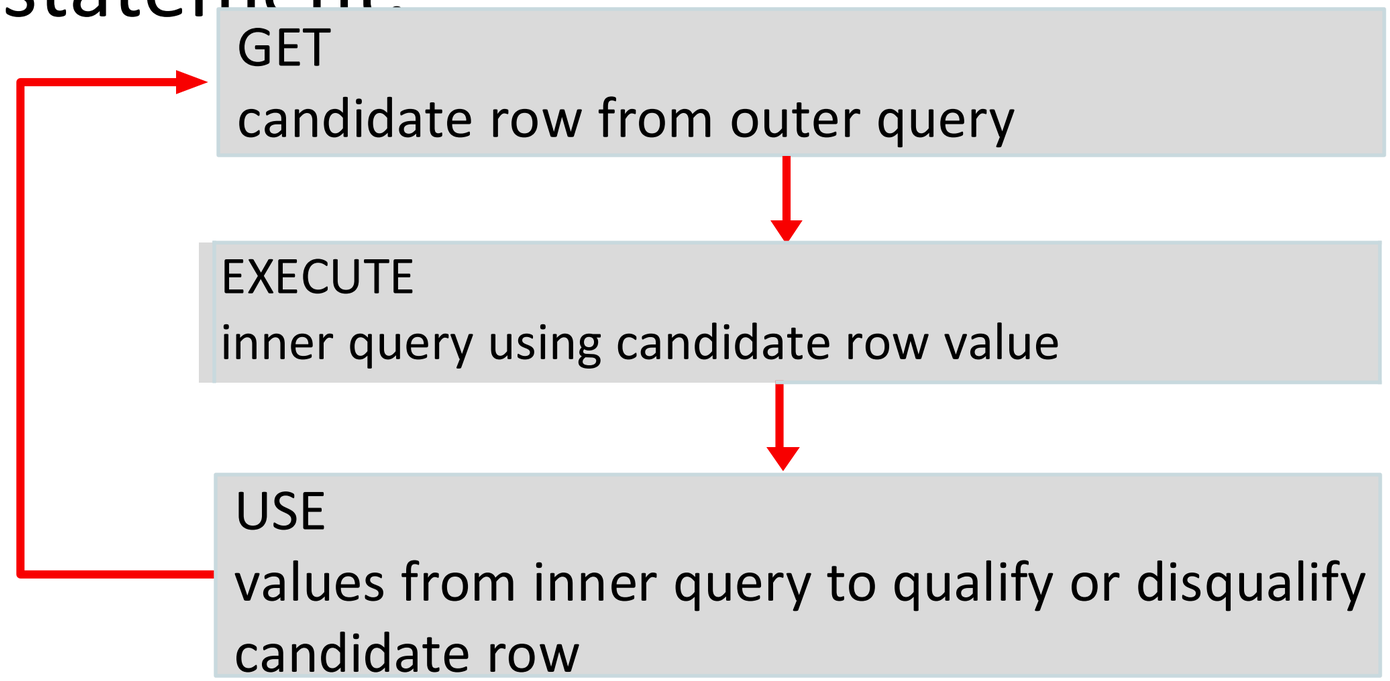
- Sometimes you have to answer more than one question in one sentence.
- Your friend might ask you if you have enough money for a cinema ticket, popcorn, and a drink.
- Before you can answer your friend, you need to know the prices of the ticket, the popcorn, and the drink.
- You also need to see how much money you have in your pocket.
- So actually, what seemed like an easy question, turns into four questions that you need answers to before you can say "Yes" or "No."

# Purpose

- In business, you might get asked to produce a report of all employees earning more than the average salary for their departments.
- So here you first have to calculate the average salary per department, and then compare the salary for each employee to the average salary of that employee's department.

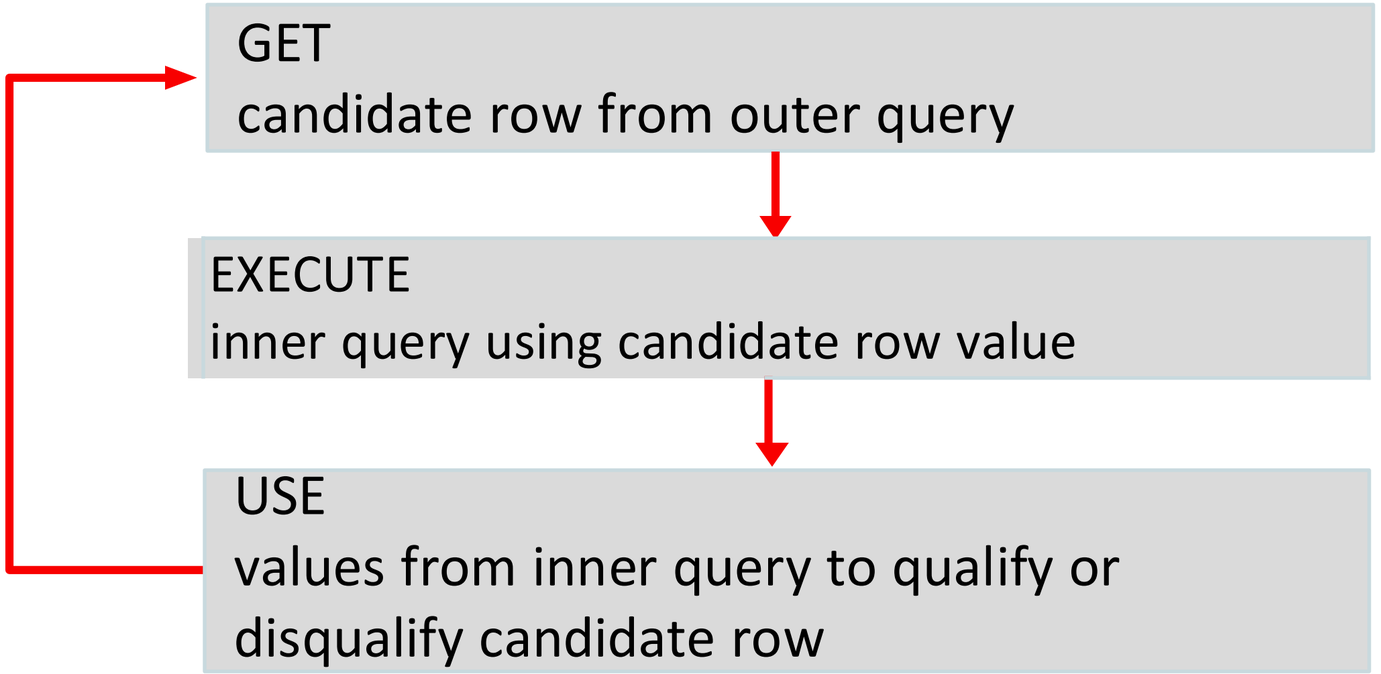
# Correlated Subqueries

- The Oracle server performs a correlated subquery when the subquery references a column from a table referred to in the parent statement.



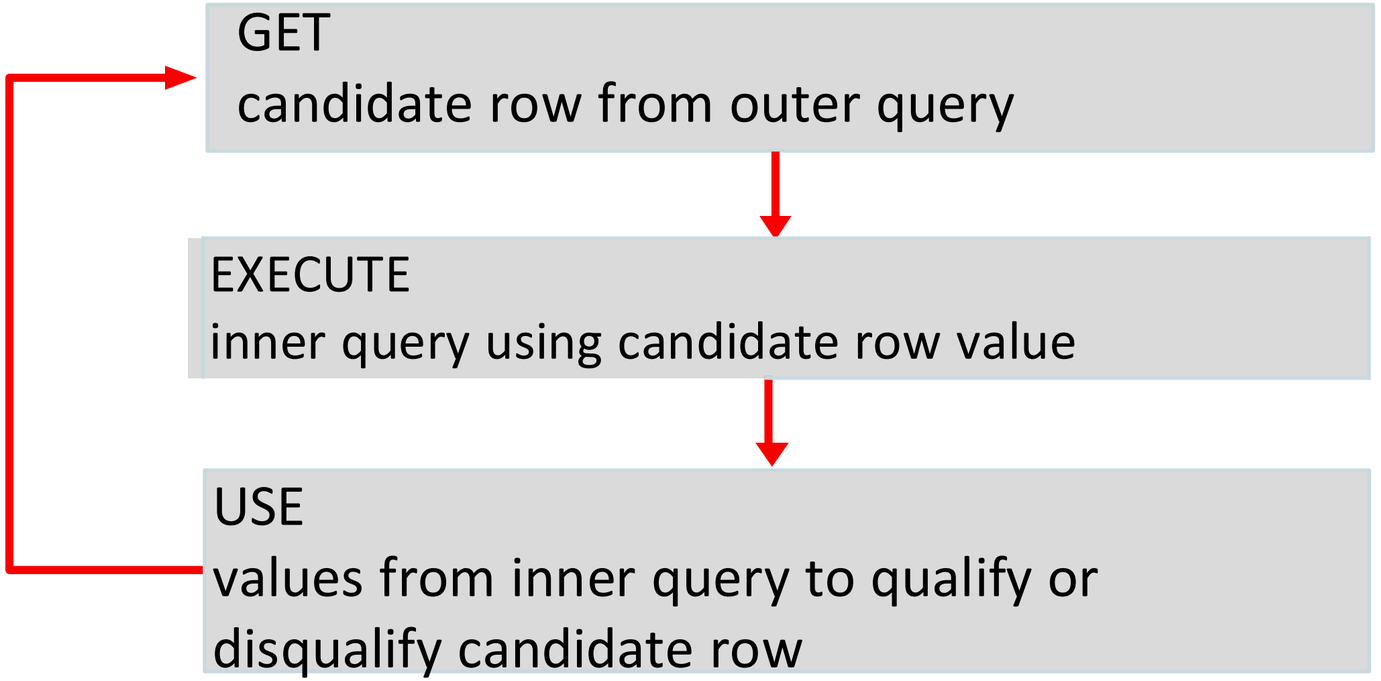
# Correlated Subqueries

- A correlated subquery is evaluated once for each row processed by the parent statement.



# Correlated Subqueries

- The parent statement can be a SELECT, UPDATE, or DELETE statement.



# Correlated Subquery Example

- Whose salary is higher than the average salary of their department?
- To answer that question, we need to write a correlated subquery.
- Correlated subqueries are used for row-by-row processing.

```
SELECT o.first_name, o.last_name,
o.salary FROM employees o WHERE
o.salary >
    (SELECT AVG(i.salary) FROM
     employees i
     WHERE i.department_id =
           o.department_id);
```

| FIRST_NAME | LAST_NAME | SALARY |
|---|---|---|
| Steven | King | 24000 |
| Shelley | Higgins | 12000 |
| Eleni | Zlotkey | 10500 |
| Ellen | Abel | 11000 |
| Kevin | Mourgos | 5800 |
| Alexander | Hunold | 9000 |
| Michael | Hartstein | 13000 |

# Correlated Subquery Example

- Each subquery is executed once for every row of the outer query.
- With a normal subquery, the inner SELECT query runs first and executes once, returning values to be used by the outer query.

```
SELECT o.first_name, o.last_name, o.salary
FROM employees o WHERE o.salary >
    (SELECT AVG(i.salary) FROM employees i
     WHERE i.department_id = o.department_id);
```

| FIRST_NAME | LAST_NAME | SALARY |
|---|---|---|
| Steven | King | 24000 |
| Shelley | Higgins | 12000 |
| Eleni | Zlotkey | 10500 |
| Ellen | Abel | 11000 |
| Kevin | Mourgos | 5800 |
| Alexander | Hunold | 9000 |
| Michael | Hartstein | 13000 |

# Correlated Subquery Example

- A correlated subquery, however, executes once for each row considered by the outer query.
- In other words, the inner query is driven by the outer query.
- The correlated subquery in this example is marked in red.

```
SELECT o.first_name,
       o.last_name, o.salary
FROM employees o WHERE
o.salary >
   (SELECT AVG(i.salary) FROM
    employees i
    WHERE i.department_id =
          o.department_id);
```

| FIRST_NAME | LAST_NAME | SALARY |
|---|---|---|
| Steven | King | 24000 |
| Shelley | Higgins | 12000 |
| Eleni | Zlotkey | 10500 |
| Ellen | Abel | 11000 |
| Kevin | Mourgos | 5800 |
| Alexander | Hunold | 9000 |
| Michael | Hartstein | 13000 |

# 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

- 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.

# WITH Clause

- If you have to write a very complex query with joins and aggregations used many times, you can write the different parts of the statement as query blocks and then use those same query blocks in a SELECT statement.
- Oracle allows you to write named subqueries in one single statement, as long as you start your statement with the keyword WITH.
- The WITH clause retrieves the results of one or more query blocks and stores those results for the user who runs the query.

# WITH Clause

- The WITH clause improves performance.
- The WITH clause makes the query easier to read.
- The syntax for the WITH clause is as follows:

```
WITH subquery-name AS (subquery), subquery-name AS (subquery) SELECT
        column-list
    FROM    {table | subquery-name | view} WHERE  condition is true;
```

# WITH Clause

- Write the query for the following requirement:
  - Display a list of employee last names that are not managers.
- To construct this query, you will first need to get a list of manager_ids from the employee table, then return the names of the employees whose employee id is not on the managers list.
- We can create a named subquery using the WITH clause to retrieve the manager_id from the employees table, then the outer query will return the employees that do not appear on that list.

# WITH Clause

```
WITH managers AS
    (SELECT DISTINCT manager_id FROM
     employees
     WHERE manager_id IS NOT NULL)

SELECT last_name AS "Not a manager" FROM
employees
WHERE employee_id NOT IN
    (SELECT *
     FROM managers);
```

| Not a manager |
|---|
| Whalen |
| Gietz |
| Abel |
| Taylor |
| Grant |
| Rajs |
| Davies |
| Vargas |
| Ernst |
| … |

# Terminology

Key terms used in this lesson included:

- Subquery
- Inner query
- Outer query
- Single-row subquery
- Multiple-row subquery

# Summary

In this lesson, you should have learned how to:

- Construct and execute a single-row subquery in the WHERE clause or HAVING clause
- Construct and execute a SELECT statement using more than one subquery
- Construct and execute a SELECT statement using a group function in the 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

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

# Summary

In this lesson, you should have learned how to:

- Identify when correlated subqueries are needed.
- Construct and execute correlated subqueries.
- Create a query using the EXISTS and NOT EXISTS operators to test for returned rows from the subquery
- Construct and execute named subqueries using the WITH clause.

# Thank You

ukrida.ac.id

UKRIDA
Universitas Kristen Krida Wacana