

Date: 9/8/25

EXERCISE-7

Displaying data from multiple tables

Objective

After the completion of this exercise, the students will be able to do the following:

- Write SELECT statements to access data from more than one table using equality and nonequality joins
- View data that generally does not meet a join condition by using outer joins
- Join a table to itself by using a self join

Sometimes you need to use data from more than one table.

Cartesian Products

- A Cartesian product is formed when:
 - A join condition is omitted
 - A join condition is invalid
 - All rows in the first table are joined to all rows in the second table
 - To avoid a Cartesian product, always include a valid join condition in a WHERE clause.
- A Cartesian product tends to generate a large number of rows, and the result is rarely useful. You should always include a valid join condition in a WHERE clause, unless you have a specific need to combine all rows from all tables.

Cartesian products are useful for some tests when you need to generate a large number of rows to simulate a reasonable amount of data.

Example:

To display employee last name and department name from the EMPLOYEES and DEPARTMENTS tables.

```
SELECT last_name, department_name dept_name  
FROM employees, departments;
```

Types of Joins

- Equijoin
- Non-equijoin
- Outer join
- Self join
- Cross joins
- Natural joins
- Using clause
- Full or two sided outer joins
- Arbitrary join conditions for outer joins

Joining Tables Using Oracle Syntax

```
SELECT table1.column, table2.column  
FROM table1, table2  
WHERE table1.column1 = table2.column2;
```

Write the join condition in the WHERE clause.

- Prefix the column name with the table name when the same column name appears in more than one table.

Guidelines

- When writing a SELECT statement that joins tables, precede the column name with the table name for clarity and to enhance database access.
- If the same column name appears in more than one table, the column name must be prefixed with the table name.
- To join n tables together, you need a minimum of n-1 join conditions. For example, to join four tables, a minimum of three joins is required. This rule may not apply if your table has a concatenated primary key, in which case more than one column is required to uniquely identify each row

What is an Equijoin?

To determine an employee's department name, you compare the value in the DEPARTMENT_ID column in the EMPLOYEES table with the DEPARTMENT_ID values in the DEPARTMENTS table.

The relationship between the EMPLOYEES and DEPARTMENTS tables is an equijoin—that is, values

in the DEPARTMENT_ID column on both tables must be equal. Frequently, this type of join involves

primary and foreign key complements.

Note: Equijoins are also called simple joins or inner joins

```
SELECT employees.employee_id, employees.last_name, employees.department_id,
```

```
departments.department_id, departments.location_id
```

```
FROM employees, departments
```

```
WHERE employees.department_id = departments.department_id;
```

Additional Search Conditions

Using the AND Operator

Example:

To display employee Matos's department number and department name, you need an additional condition in the WHERE clause.

```
SELECT last_name, employees.department_id,
```

```
department_name
```

```
FROM employees, departments
```

```
WHERE employees.department_id = departments.department_id AND last_name = 'Matos';
```

Qualifying Ambiguous

Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Improve performance by using table prefixes.
- Distinguish columns that have identical names but reside in different tables by using column aliases.

Using Table Aliases

- Simplify queries by using table aliases.
- Improve performance by using table prefixes

Example:

```
SELECT e.employee_id, e.last_name, e.department_id,
```

```
d.department_id, d.location_id
```

```
FROM employees e, departments d
```

```
WHERE e.department_id = d.department_id;
```

Joining More than Two Tables

To join n tables together, you need a minimum of n-1 join conditions. For example, to join three

tables, a minimum of two joins is required.

Example:

To display the last name, the department name, and the city for each employee, you have to join the EMPLOYEES, DEPARTMENTS, and LOCATIONS tables.

```
SELECT e.last_name, d.department_name, l.city
FROM   employees e, departments d, locations l
WHERE  e.department_id = d.department_id
AND    d.location_id = l.location_id;
```

Non-Equijoins

A non-equijoin is a join condition containing something other than an equality operator. The relationship between the EMPLOYEES table and the JOB_GRADES table has an example of a non-equijoin. A relationship between the two tables is that the SALARY column in the EMPLOYEES table must be between the values in the LOWEST_SALARY and HIGHEST_SALARY columns of the JOB_GRADES table. The relationship is obtained using an operator other than equals (=).

Example:

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  e.salary
BETWEEN j.lowest_sal AND j.highest_sal;
```

Outer Joins

Syntax

- You use an outer join to also see rows that do not meet the join condition.
- The Outer join operator is the plus sign (+).

```
SELECT table1.column, table2.column
FROM   table1, table2
WHERE  table1.column(+) = table2.column;
SELECT table1.column, table2.column
FROM   table1, table2
WHERE  table1.column = table2.column(+);
```

The missing rows can be returned if an outer join operator is used in the join condition. The operator is a plus sign enclosed in parentheses (+), and it is placed on the "side" of the join that is deficient in information. This operator has the effect of creating one or more null rows, to which one or more

rows from the nondeficient table can be joined.

Example:

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  e.department_id(+) = d.department_id ;
```

Outer Join Restrictions

- The outer join operator can appear on only one side of the expression—the side that has information missing. It returns those rows from one table that have no direct match in the other table.
- A condition involving an outer join cannot use the IN operator or be linked to another condition by the OR operator

This query retrieves all rows in the EMPLOYEES table, even if there is no match in the DEPARTMENTS table. It also retrieves all rows in the DEPARTMENTS table, even if there is no match in the EMPLOYEES table.

Find the Solution for the following:

1. Write a query to display the last name, department number, and department name for all employees.

```
SELECT e.last_name AS "Last name", e.department_id
AS "Dept #", d.department_name AS "Department"
FROM employees e JOIN departments d ON e.department_id = d.department_id;
```

2. Create a unique listing of all jobs that are in department 80. Include the location of the department in the output.

```
SELECT DISTINCT e.job_id AS "Job", d.department_name
AS "Department", d.location_id AS "Location ID"
FROM employees e JOIN departments d ON
e.department_id = d.department_id WHERE e.department_id = 80;
```

3. Write a query to display the employee last name, department name, location ID, and city of all employees who earn a commission

```
SELECT e.last_name AS "Last Name", d.department_name
AS "Department", d.location_id AS "Location ID",
e.city AS "City" FROM employees e JOIN departments d
ON e.department_id = d.department_id JOIN locations l
ON d.location_id = l.location_id WHERE e.commission_pct
```

8. Display the employee last name and department name for all employees who have an 'a' (lowercase) in their last names. P

```
SELECT e.last_name AS "Last Name", d.department_name AS "Department" FROM
employees e JOIN departments d ON e.department_id = d.department_id
WHERE e.last_name LIKE '%a%';
```

5. Write a query to display the last name, job, department number, and department name for all employees who work in Toronto.

```
SELECT e.last_name AS "Last Name", e.job_id AS "Job",
d.department_id AS "Dept #", d.department_name AS "Department" FROM employees e JOIN departments d ON
e.department_id = d.department_id JOIN locations l ON
d.location_id = l.location_id WHERE l.city = 'Toronto';
```

6. Display the employee last name and employee number along with their manager's last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, Respectively

```
SELECT e.last_name AS "Employee", e.employee_id
AS "Emp#", m.last_name AS "Manager", m.employee_id
AS "Mgr#" FROM employees e LEFT JOIN employees m
ON e.manager_id = m.employee_id ORDER BY e.employee_id;
```

7. Modify lab4_6.sql to display all employees including King, who has no manager. Order the results by the employee number.

```
SELECT e.employee_id AS "Emp#", e.last_name  
AS "Employee", m.last_name AS "Manager" FROM employees  
e LEFT JOIN employees m ON e.manager_id = m.  
employee_id ORDER BY e.employee_id;
```

8. Create a query that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label

```
SELECT e.last_name AS "Employee", e.department_id  
AS "Dept#", c.last_name AS "Colleague" FROM employees  
e JOIN employees c ON e.department_id = c.department  
-id ORDER BY e.department_id, e.last_name;
```

9. Show the structure of the JOB_GRADES table. Create a query that displays the name, job, department name, salary, and grade for all employees

```
SELECT e.last_name AS "Employee", e.job_id AS "Job", d.  
department_name AS "Department", e.salary AS "Salary", g.  
grade_level AS "Grade" FROM employees e JOIN departments d  
ON e.department_id = d.department_id ORDER BY e.last_name;
```

10. Create a query to display the name and hire date of any employee hired after employee Davies.

```
SELECT last_name AS "Employee", hire_date AS "Hire Date"  
WHERE hire_date > (SELECT hire_date FROM  
employees WHERE last_name = "Davies") ORDER  
BY hire_date;
```

11. Display the names and hire dates for all employees who were hired before their managers, along with their manager's names and hire dates. Label the columns Employee, Emp Hired, Manager, and Mgr Hired, respectively.

```
SELECT e.last_name AS "Employee", e.hire_date AS  
"Emp Hired", m.last_name AS "Manager", m.hire_date  
AS "Mgr Hired" FROM employees e JOIN employees m  
ON e.manager_id = m.manager_id WHERE e.hire_date  
< m.hire_date ORDER BY e.hire_date;
```

Evaluation Procedure	Marks awarded
Query(5)	5
Execution (5)	5
Viva(5)	5
Total (15)	15
Faculty Signature	T. B. M.