

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 displays 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.

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, e.department-id, d.department-name  
FROM employee 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, l.location, l.city FROM employee e JOIN departments d ON e.department-id = d.department-id JOIN locations l ON d.location-id = l.location-id WHERE e.commission-perc > 0;
```

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

```
SELECT DISTINCT e.job-id, l.location, l.city FROM employee e JOIN departments d ON e.department-id = d.department-id JOIN locations l ON d.location-id = l.location-id WHERE department-id = 80;
```

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

```
SELECT e.last-name, d.department-name FROM employee e JOIN departments d ON e.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, e.job-id, e.department-id, d.department-name FROM employee 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;
```

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

```
SELECT e.last_name AS employee, e.employee_id AS Emp #, m.last_name AS Manager,
       m.employee_id AS Mgr# FROM employees
    m ON e.manager_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 e1.last_name AS employee, e1.department_id AS Dept-ID, e2.last_name AS colleague
  FROM employee e1 JOIN employees e2 ON
e1.department_id = e2.department_id;
```

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

```
DESCRIBE job_grades;
SELECT e.last_name, e.job_id, d.department_name, e.salary, j.grade
  FROM employees e JOIN departments d ON
e.department_id BETWEEN j.lowest AND j.highest;
```

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

```
SELECT e.last_name, e.hire_date FROM
employees e WHERE e.hire_date > (SELECT hire_
date FROM employees WHERE last_name =
'Davies');
```

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.
WHERE e.hire_date < m.hire_
date;
```

Evaluation Procedure	Marks awarded
Query(5)	5
Execution (5)	5
Viva(5)	5
Total (15)	15
Faculty Signature	R Pn 9/9/25