# 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.

• 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?**

Todetermine 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

```
SELECTemployees.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'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**
• Usetable prefixesto 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 queriesbyusing 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;
```

~~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 ~~EMPLOY~~EES, 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 operatorother 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 ;

_____

• 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

**Self Join** Sometimes you need to join a table to itself. **Example:**

To find the name of each employee's manager, you need to join the EMPLOYEES table to itself, or perform a self join. SELECT worker.last_name || ' works for ' || manager.last_name FROM employees worker, employees manager WHERE worker.manager_id = manager.employee_id ;

**Use a join to query data from more than one table.**

SELECT table1.column, table2.column FROM table1 [CROSS JOIN table2] | [NATURAL JOIN table2] | [JOIN table2 USING (column_name)] | [JOIN table2 ON(table1.column_name = table2.column_name)] | [LEFT|RIGHT|FULL OUTER JOIN table2 ON (table1.column_name = table2.column_name)]; In the syntax: table1.column Denotes the table and column from which data is retrieved

CROSS JOIN Returns a Cartesian product from the two tables
NATURAL JOIN Joins two tables based on the same column name
JOIN table USING column_name Performs an equijoin based on the column name
JOIN table ON table1.column_name Performs an equijoin based on the condition in the ON clause
= table2.column_name

**LEFT/RIGHT/FULL OUTER**

**Creating Cross Joins**

• The CROSSJOIN clauseproduces the crossproduct of two tables.
• This is the same as a Cartesian product between the two tables.

**Example:**
SELECT last_name, department_name
FROM employees
CROSS JOIN departments ;
SELECT last_name, department_name
FROM employees, departments;

• The NATURALJOIN clause is based onall columns in the two tables that have the same name. • It selects rows from the two tables that have equal values in all matched columns. • If the columns having the same names have different data types, an error is returned.

**Example:**

SELECT department_id, department_name, location_id, city FROM departments NATURAL JOIN locations ; LOCATIONS table is joined to the DEPARTMENT table by the LOCATION_ID column, which is the only column of the same name in both tables. If other common columns were present, the join would have used them all. **Example:** SELECT department_id, department_name, location_id, city FROM departments NATURAL JOIN locations WHERE department_id IN (20, 50);

## Creating Joins with the USING Clause

• If several columns have the same names but the data types do not match, the NATURAL JOIN clause can be modified with the USING clause to specify the columns that should be used for an equijoin. • Use the USING clause to match only one column when more than one column matches. • Do not use a table name or alias in the referenced columns. • The NATURAL JOIN and USING clauses are mutually exclusive.

**Example:**

SELECT l.city, d.department_name FROM locations l JOIN departments d USING (location_id) WHERE location_id = 1400; EXAMPLE:

_____

_____

SELECT e.employee_id, e.last_name, d.location_id
FROM employees e JOIN departments d
USING (department_id) ;

## Creating Joins with the ON Clause

• The join condition for the natural join is basically an equijoin of all columns with the same name.
• To specify arbitrary conditions or specify columns to join, the ON clause is used.
• The join condition is separated from other searchconditions.
• The ON clause makes code easy to understand.

**Example:**

SELECT e.employee_id, e.last_name, e.department_id,
d.department_id, d.location_id
FROM employees e JOIN departments d

ON   (e.department_id = d.departbent_id);
EXAMPLE:

FROM employees e JOIN employees m
ON    (e.manager_id = m.employee_id);
INNER Versus OUTER Joins


 • A join between two tables that returns the results of the inner join as well as unmatched rows left (or
right) tables is a left (or right) outer join.
 • A join between two tables that returns the results of an inner join as well as the results of a left and
right join is a full outer join.

**LEFT OUTER JOIN**
**Example:**
**SELECT e.last_name,e.department_id,d.department_name**
_____


FROM employees e
LEFT OUTER JOIN departments d
ON (e.department_id = d.department_id) ;
Example of LEFT OUTER JOIN
This query retrieves all rows in the EMPLOYEES table, which is the left table even if there is no
match in the DEPARTMENTS table.
This query was completed in earlier releases as follows:
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE d.department_id (+) = e.department_id;


 **RIGHT OUTER JOIN**
**Example:**

 SELECT e.last_name, e.department_id, d.department_name
FROM employees e
RIGHT OUTER JOIN departments d
ON (e.department_id = d.department_id) ;
This query retrieves all rows in the DEPARTMENTS table, which is the right table even if there is
no
match in the EMPLOYEES table.
This query was completed in earlier releases as follows:
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE d.department_id = e.department_id (+);

**FULLOUTERJOIN Example:** SELECT e.last_name, e.department_id, d.department_name FROM employees e FULL OUTER JOIN departments d ON (e.department_id = d.department_id) ; This query retrieves all rows in the EMPLOYEES table, even if there is no match in the DEPARTMENTS table. It alslso 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.dept_name
AS department_name
FROM employees e
JOIN department d
ON e.department_id = d.dept_id;
```

🗑 ⓘ   Download ▾   Execution time: 0.005 seconds

| | LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|---|---|---|---|
| 1 | Brown | 40 | Human Resources |

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, d.location_id
FROM employees e
JOIN department d
ON e.department_id = d.dept_id
WHERE e.department_id = 80;
```

🗑 ⓘ   Download ▾   Execution time: 0.016 seconds

| JOB_ID | LOCATION_ID |
|---|---|
| No items to display. | |

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,d.dept_name,d.location_id,l.city
FROM employees e
JOIN department d ON e.department_id = d.dept_id
JOIN locations l ON d.location_id = l.location_i
WHERE e.commission_pct IS NOT NULL;
```

🗑 ⓘ   Download ▾   Execution time: 0.06 seconds

| | LAST_NAME | DEPT_NAME | LOCATION_ID | CITY |
|---|---|---|---|---|
| 1 | Brown | Human Resources | 2400 | New York |

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

2.

```sql
SELECT e.last_name, d.dept_name
FROM employees e
JOIN department d ON e.department_id = d.dept_id
WHERE LOWER(e.last_name) LIKE '%a%';
```

Download ▾  Execution time: 0.008 seconds

| LAST_NAME | DEPT_NAME |
|---|---|
| No items to display. | |

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

```sql
SELECT e.last_name,e.job_id,e.department_id,d.dept_name
FROM employees e
JOIN department d ON e.department_id = d.dept_id
JOIN locations l ON d.location_id = l.location_id
WHERE LOWER(l.city) = 'toronto';
```

Download ▾  Execution time: 0.004 seconds

| LAST_NAME | JOB_ID | DEPARTMENT_ID | DEPT_NAME |
|---|---|---|---|
| No items to display. | | | |

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

```sql
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;
```

Download ▾  Execution time: 0.007 seconds

| | EMPLOYEE | EMP# | MANAGER | MGR# |
|---|---|---|---|---|
| 1 | Smith | 1002 | Doe | 1001 |
| 2 | Brown | 1003 | Doe | 1001 |
| 3 | White | 1004 | Brown | 1003 |
| 4 | Doe | 1001 | (null) | (null) |

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

```sql
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;
```

Download ▾  Execution time: 0.003 seconds

| | EMPLOYEE | EMP# | MANAGER | MGR# |
|---|---|---|---|---|
| 1 | Doe | 1001 | (null) | (null) |
| 2 | Smith | 1002 | Doe | 1001 |
| 3 | Brown | 1003 | Doe | 1001 |
| 4 | White | 1004 | Brown | 1003 |

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

```sql
SELECT e1.last_name AS Employee, e1.department_id AS "Dept No",
e2.last_name AS "Colleague"
FROM employees e1
JOIN employees e2 ON
e1.department_id = e2.department_id
WHERE e1.employee_id != e2.employee_id;
```

Download ▾  Execution time: 0.006 seconds

| EMPLOYEE | DEPT NO | COLLEAGUE |
|---|---|---|
| No items to display. | | |

```
DESC job_grades;
SELECT e.last_name,e.job_id,
d.dept_name,e.salary,j.grade_level
FROM employees e
JOIN department d ON
e.department_id = d.dept_id
JOIN job_grade j ON e.salary
BETWEEN j.lowest_sal AND j.highest_sal;
```

```
Name          Null?  Type
-----------   -----  -----------
GRADE_LEVEL          VARCHAR2(2)
LOWEST_SAL           NUMBER
HIGHEST_SAL          NUMBER
```

🗑 ⓘ   Download ▾   Execution time: 0.012 seconds

|   | LAST_NAME | JOB_ID | DEPT_NAME | SALARY | GRADE_LEVEL |
|---|-----------|--------|-----------|--------|-------------|
| 1 | Brown     | HR_REP | Human Resources | 6000 | B |
| 2 | Brown     | HR_REP | Human Resources | 6000 | B |

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

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

🗑 ⓘ   Download ▾   Execution time: 0.008 seconds

| EMPLOYEE_ID | LAST_NAME | HIRE_DATE |
|-------------|-----------|-----------|
| No items to display. | | |

## 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.employee_id
WHERE e.hire_date < m.hire_date;
```

🗑 ⓘ   Download ▾   Execution time: 0.006 seconds

| EMPLOYEE | EMP HIRED | MANAGER | MGR HIRED |
|----------|-----------|---------|-----------|
| No items to display. | | | |

| Evaluation Procedure | Marks awarded |
|----------------------|---------------|
| Query(5) | |
| Execution (5) | |
| Viva(5) | |
| Total (15) | |
| Faculty Signature | |