4-Displaying Data from Multiple Tables

Objectives

After completing this lesson, you should 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

Obtaining Data from Multiple Tables

EMPLOYEES DEPARTMENTS

Sometimes you need to use data from more than one table. In the example, the report displays data fromtwo separate tables.
• Employee IDs exist in the EMPLOYEES table.
• Department IDs exist in both the EMPLOYEES and DEPARTMENTS tables.
• Location IDs exist in the DEPARTMENTS table.
To produce the report, you need to link the EMPLOYEES and DEPARTMENTS tables and access data from both of them.

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.
Cartesian Products

Generating a Cartesian Product

EMPLOYEES
(20 rows)
DEPARTMENTS
(8 rows)
Cartesian product:
20x8=160 rows

SELECT last_name, department_name dept_name

FROM employees, departments;

Types of Joins

Oracle Proprietary SQL: 1999
Compliant Joins: Joins (8i and prior):

• Equijoin
•Cross joins
•Nonequijoin
•Natural joins
•Outer join
Using clause

•Self join
•Full or two sided outer joins
• Arbitrary join conditions for outer joins

Joining Tables Using Oracle Syntax

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

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**.**

What Is an Equijoin?

EMPLOYEES DEPARTMENTS

Foreign key Primary key

Equijoins

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*

.

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500
205	Higgins	110	110	1700
206	Gietz	110	110	1700

19 rows selected.

Retrieving Records with Equijoins

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

EMPLOYEES DEPARTMENTS

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

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

EMPLOYEES
DEPARTMENTS
LOCATIONS
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.

```
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;
```

Nonequijoins

EMPLOYEES JOB_GRADES
Salary in the EMPLOYEES
table must be between
lowest salary and highest
salary in the JOB_GRADES table.

Retrieving Records with Nonequijoins

```
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

DEPARTMENTS EMPLOYEES
There are no employees in
department 190.

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

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(+);
```

Using Outer Joins

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

Self Joins

EMPLOYEES (WORKER) EMPLOYEES (MANAGER)
MANAGER_ID in the WORKER table is equal to
EMPLOYEE_ID in the MANAGER
table.

Joining a Table to Itself

```
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 )];
```

Creating Cross Joins

•The CROSS JOIN clause produces the cross-
product of two tables.
• This is the same as a Cartesian product between
the two tables.

```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments;
```

Creating Cross Joins

The example in the slide gives the same results as the following:
SELECT last_name, department_name FROM employees, departments;

Creating Natural Joins

• The NATURAL JOIN
clause is based on all 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, then an error is returned.
Note: The join can happen only on columns having the same names and data types in both the tables. If the columns have the same name, but different data types, then the

NATURAL JOIN syntax causes an error.

Retrieving Records with Natural Joins

```
SELECT department_id, department_name,
location_id, city
FROM   departments
NATURAL JOIN locations;

SELECT department_id, department_name,
departments.location_id, city
FROM   departments, locations
WHERE  departments.location_id = locations.location_id;

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.
Note: 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.

For example, this statement is valid:

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

This statement is invalid because the LOCATION_ID is qualified in the where clause:

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

ORA-25154: column part of USING clause cannot have qualifier
The same restriction applies to NATURAL joins also. Therefore columns that have the same name in both

tables have to be used without any qualifiers.

Retrieving Records with the USING Clause

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

```
SELECT employee_id, last_name,
employees.department_id, location_id
FROM   employees, departments
WHERE  employees.department_id = departments.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.
•Separates the join condition from other search conditions.
•The ON clause makes code easy to understand.

```
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.department_id);
```

INNER versus OUTER Joins

• In SQL: 1999, the join of two tables returning only matched rows is an inner join.
• 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.
Joins: Comparing SQL: 1999 to Oracle Syntax

LEFT OUTER JOIN

```
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

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);

Example of

RIGHT OUTER JOIN
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 (+);

FULL OUTER JOIN

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);

Additional Conditions

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.department_id)
AND e.manager_id = 149;

Higgins	110	Accounting
Gietz	110	Accounting

19 rows selected.

Practice 3, Part 1

1. Write a query to display the last name, department number, and department name for all employees.
2. Create a unique listing of all jobs that are in department 30. Include the location of department 90 in the output.
3. Write a query to display the employee last name, department name, location ID, and city of all employees who earn a commission.

4. Display the employee last name and department name for all employees who have an *a* (lowercase) in their last names. Place your SQL statement in a text file named lab4_4.sql.

5. Write a query to display the last name, job, department number, and department name for all employees who work in 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.
Place your SQL statement in a text file named lab4_6.sql.

7. Modify lab4_6.sql to display all employees including King, who has no manager. Order the results by the employee number. Place your SQL statement in a text file named lab4_7.sql . Run the query in lab4_7.sql.
If you have time, complete the following exercises:
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.

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.
If you want an extra challenge, complete the following exercises:
10. Create a query to display the name and hire date of any employee hired after employee Davies.
11. Display the names and hire dates for all employees who were hire d before their managers, along with their manager's names and hire dates. Label the columns Employee , Emp Hired , Manager , and Mgr Hired , respectively.