6

Displaying Data from Multiple Tables

Oracle for Base

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 equijoins and nonequijoins
- Join a table to itself by using a self-join
- View data that generally does not meet a join condition by using outer joins
- Generate a Cartesian product of all rows from two or more tables

Lesson Stanford

- Types of JOINS and its syntax
- Natural join:
  - USING clause
  - ON clause
- Self-join
- OUTER join:
  - LEFT OUTER join
  - RIGHT OUTER join
  - FULL OUTER join

Obtaining Data from Multiple Tables

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
...			
18	202	Fay	20
19	205	Higgins	110
20	206	Gietz	110

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

	EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	200	10	Administration
2	201	20	Marketing
3	202	20	Marketing
4	124	50	Shipping
5	144	50	Shipping
...			
18	205	110	Accounting
19	206	110	Accounting

Types of Joins

Joins that are compliant with the SQL:1999 standard include the following:

- Natural joins:
  - `NATURAL JOIN` clause
  - `USING` clause
  - `ON` clause
- Outer joins:
  - `LEFT OUTER JOIN`
  - `RIGHT OUTER JOIN`
  - `FULL OUTER JOIN`

Joining Tables Using SQL:1999 Syntax

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

```
SELECT   table1.column, table2.column
FROM     table1
[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)]|
[CROSS JOIN table2];
```

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to improve performance.
- Instead of full table name prefixes, use table aliases.
- Table alias gives a table a shorter name:
  - Keeps SQL code smaller, uses less memory
- Use column aliases to distinguish columns that have identical names, but reside in different tables.

Lesson Stanford

- Types of JOINS and its syntax
- Natural join:
  - USING clause
  - ON clause
- Self-join
- OUTER join:
  - LEFT OUTER join
  - RIGHT OUTER join
  - FULL OUTER join

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, an error is returned.

Retrieving Records with Natural Joins

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

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
1	60	IT	1400	Southlake
2	50	Shipping	1500	South San Francisco
3	10	Administration	1700	Seattle
4	90	Executive	1700	Seattle
5	110	Accounting	1700	Seattle
6	190	Contracting	1700	Seattle
7	20	Marketing	1800	Toronto
8	80	Sales	2500	Oxford

Creating Joins with the `USING` Clause

- If several columns have the same names but the data types do not match, natural join can be applied using 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.
- The `NATURAL JOIN` and `USING` clauses are mutually exclusive.

Joining Column Names

EMPLOYEES

EMPLOYEE_ID	DEPARTMENT_ID
100	90
101	90
102	90
103	60
104	60
107	60
124	50
141	50
142	50
143	50
144	50
149	80
174	80
176	80

...

Foreign key

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	50	Shipping
4	60	IT
5	80	Sales
6	90	Executive
7	110	Accounting
8	190	Contracting

Primary key

Retrieving Records with the USING Clause

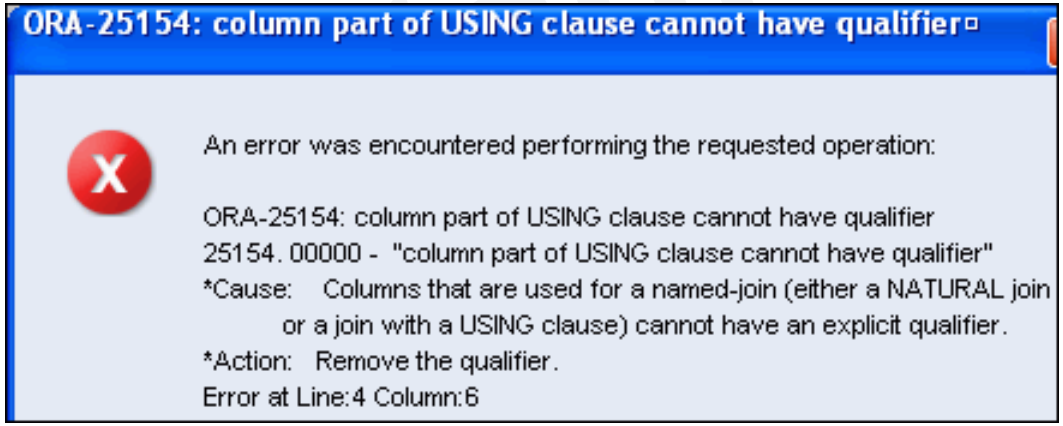
```
SELECT employee_id, last_name,
       location_id, department_id
FROM   employees JOIN departments
USING (department_id) ;
```

	EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
1	200	Whalen	1700	10
2	201	Hartstein	1800	20
3	202	Fay	1800	20
4	124	Mourgos	1500	50
5	144	Vargas	1500	50
6	143	Matos	1500	50
7	142	Davies	1500	50
8	141	Rajs	1500	50
9	107	Lorentz	1400	60
10	104	Ernst	1400	60
...				
19	205	Higgins	1700	110

Using Table Aliases with the USING Clause

- Do not qualify a column that is used in the USING clause.
- If the same column is used else where in the SQL statement, do not alias it.

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



Creating Joins with the ON Clause

- The join condition for the natural join is basically an equijoin of all columns with the same name.
- Use the ON clause to specify arbitrary conditions or specify columns to join.
- The join condition is separated from other search conditions.
- The ON clause makes code easy to understand.

Retrieving Records with the ON Clause

```
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);
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	124	Mourgos	50	50	1500
5	144	Vargas	50	50	1500
6	143	Matos	50	50	1500
7	142	Davies	50	50	1500
8	141	Rajs	50	50	1500
9	107	Lorentz	60	60	1400
10	104	Ernst	60	60	1400

...

Creating Three-Way Joins with the ON Clause

```
SELECT employee_id, city, department_name
FROM   employees e
JOIN   departments d
ON     d.department_id = e.department_id
JOIN   locations l
ON     d.location_id = l.location_id;
```

	EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100	Seattle	Executive
2	101	Seattle	Executive
3	102	Seattle	Executive
4	103	Southlake	IT
5	104	Southlake	IT
6	107	Southlake	IT
7	124	South San Francisco	Shipping
8	141	South San Francisco	Shipping

...

Applying Additional Conditions to a Join

Use the AND clause or the WHERE clause to apply 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 ;
```

Or

```
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)
WHERE   e.manager_id = 149 ;
```

Lesson Stanford

- Types of JOINS and its syntax
- Natural join:
  - USING clause
  - ON clause
- Self-join
- OUTER join:
  - LEFT OUTER join
  - RIGHT OUTER join
  - FULL OUTER join

Joining a Table to Itself

EMPLOYEES (WORKER)

	EMPLOYEE_ID	LAST_NAME	MANAGER_ID
1	100	King	(null)
2	101	Kochhar	100
3	102	De Haan	100
4	103	Hunold	102
5	104	Ernst	103
6	107	Lorentz	103
7	124	Mourgos	100
8	141	Rajs	124
9	142	Davies	124
10	143	Matos	124

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos
141	Rajs
142	Davies
143	Matos

...

MANAGER_ID in the WORKER table is equal to EMPLOYEE_ID in the MANAGER table.

Self-Joins Using the ON Clause

```
SELECT worker.last_name emp, manager.last_name mgr
FROM   employees worker JOIN employees manager
ON     (worker.manager_id = manager.employee_id);
```

	EMP	MGR
1	Hunold	De Haan
2	Fay	Hartstein
3	Gietz	Higgins
4	Lorentz	Hunold
5	Ernst	Hunold
6	Zlotkey	King
7	Mourgos	King
8	Kochhar	King
9	Hartstein	King
10	De Haan	King

...

Lesson Stanford

- Types of JOINS and its syntax
- Natural join:
  - USING clause
  - ON clause
- Self-join
- OUTER join:
  - LEFT OUTER join
  - RIGHT OUTER join
  - FULL OUTER join

Returning Records with No Direct Match with Outer Joins

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

EMPLOYEES

	DEPARTMENT_ID	LAST_NAME
1	90	King
2	90	Kochhar
3	90	De Haan
4	60	Hunold
5	60	Ernst
6	60	Lorentz
7	50	Mourgos
8	50	Rajs
9	50	Davies
10	50	Matos
...		
19	110	Higgins
20	110	Gietz

There are no employees in department 190.

INNER Versus OUTER Joins

- In SQL:1999, the join of two tables returning only matched rows is called an inner join.
- A join between two tables that returns the results of the inner join as well as the unmatched rows from the left (or right) table is called 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

```
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) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Vargas	50	Shipping
5	Matos	50	Shipping

...

17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

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) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Higgins	110	Accounting

...

19	Taylor	80	Sales
20	Grant	(null)	(null)
21	(null)	190	Contracting

FULL OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON   (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Higgins	110	Accounting

...

19	Taylor	80	Sales
20	Grant	(null)	(null)
21	(null)	190	Contracting

Summary

In this lesson, you should have learned how to use joins to display data from multiple tables by using:

- Equijoins
- Outer joins
- Self-joins
- Natural joins
- Full (or two-sided) outer joins

6

Using Subqueries to Solve Queries

Objectives

After completing this lesson, you should be able to do the following:

- Define subqueries
- Describe the types of problems that the subqueries can solve
- List the types of subqueries
- Write single-row subqueries

Lesson Stanford

- Subquery: Types, syntax, and guidelines
- Single-row subqueries:
  - Group functions in a subquery
  - `HAVING` clause with subqueries
- Null values in a subquery

Using a Subquery to Solve a Problem

Who has a salary greater than Abel's?

Main query:

Which employees have salaries greater than Abel's salary?

Subquery:

What is Abel's salary?

Subquery Syntax

```
SELECT    select_list
FROM      table
WHERE     expr operator
                        (SELECT        select_list
                        FROM           table);
```

- The subquery (inner query) executes *before* the main query (outer query).
- The result of the subquery is used by the main query.

Using a Subquery

```
SELECT last_name, salary
FROM   employees
WHERE  salary >
                (SELECT salary
                 FROM   employees
                 WHERE  last_name = 'Abel');
```

11000

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hartstein	13000
5	Higgins	12000

Guidelines for Using Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition for readability (However, the subquery can appear on either side of the comparison operator.).
- Use single-row operators with single-row subqueries and multiple-row operators with multiple-row subqueries.

Types of Subqueries

- Single-row subquery

- Multiple-row subquery

Lesson Stanford

- Subquery: Types, syntax, and guidelines
- Single-row subqueries:
  - Group functions in a subquery
  - `HAVING` clause with subqueries
- Null values in a subquery

Single-Row Subqueries

- Return only one row
- Use single-row comparison operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

Executing Single-Row Subqueries

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  last_name = 'Taylor')
AND    salary >
                (SELECT salary
                 FROM   employees
                 WHERE  last name = 'Taylor');
```

SA_REP

8600

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000

Using Group Functions in a Subquery

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary =
                (SELECT MIN(salary)
                 FROM   employees);
```

2500

	LAST_NAME	JOB_ID	SALARY
1	Vargas	ST_CLERK	2500

The HAVING Clause with Subqueries

- The Oracle server executes the subqueries first.
- The Oracle server returns results into the HAVING clause of the main query.

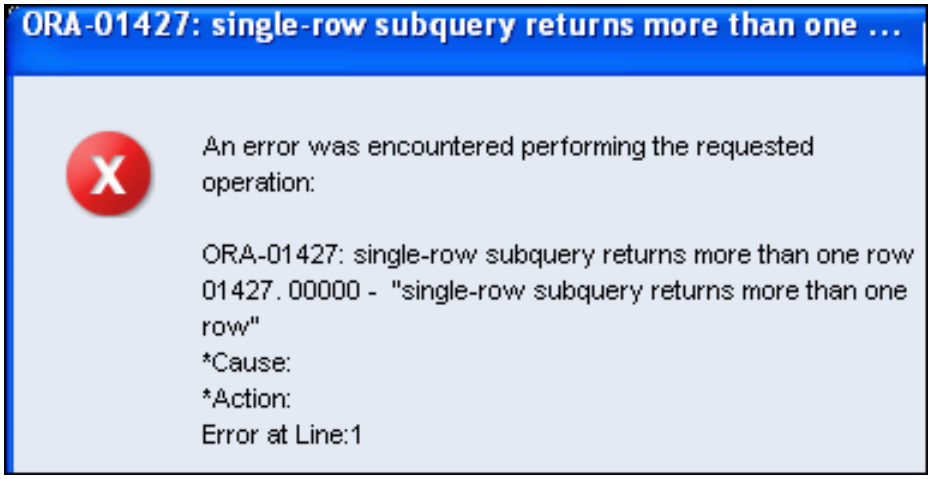
```
SELECT    department_id, MIN(salary)
FROM      employees
GROUP BY  department_id
HAVING    MIN(salary) >
                        (SELECT MIN(salary)
                         FROM   employees
                         WHERE  department_id = 50);
```

2500

	DEPARTMENT_ID	MIN(SALARY)
1	(null)	7000
2	90	17000
3	20	6000
...		
7	10	4400

What Is Wrong with This Statement?

```
SELECT employee_id, last_name
FROM   employees
WHERE  salary =
                (SELECT   MIN(salary)
                 FROM     employees
                 GROUP BY department_id);
```



Single-row operator with multiple-row subquery

No Rows Returned by the Inner Query

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  last_name = 'Haas');
```

```
0 rows selected
```

Subquery returns no rows because there is no employee named "Haas."

Lesson Stanford

- Subquery: Types, syntax, and guidelines
- Single-row subqueries:
  - Group functions in a subquery
  - `HAVING` clause with subqueries
- Null values in a subquery

Null Values in a Subquery

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
                        (SELECT mgr.manager_id
                         FROM   employees mgr);
```

```
0 rows selected
```

Summary

In this lesson, you should have learned how to:

- Identify when a subquery can help solve a problem
- Write subqueries when a query is based on unknown values

```
SELECT    select_list
FROM      table
WHERE     expr operator
                        (SELECT select_list
                        FROM     table);
```

Practice 6: Overview

This practice covers the following topics:

- Creating subqueries to query values based on unknown criteria
- Using subqueries to find out the values that exist in one set of data and not in another