

# 3

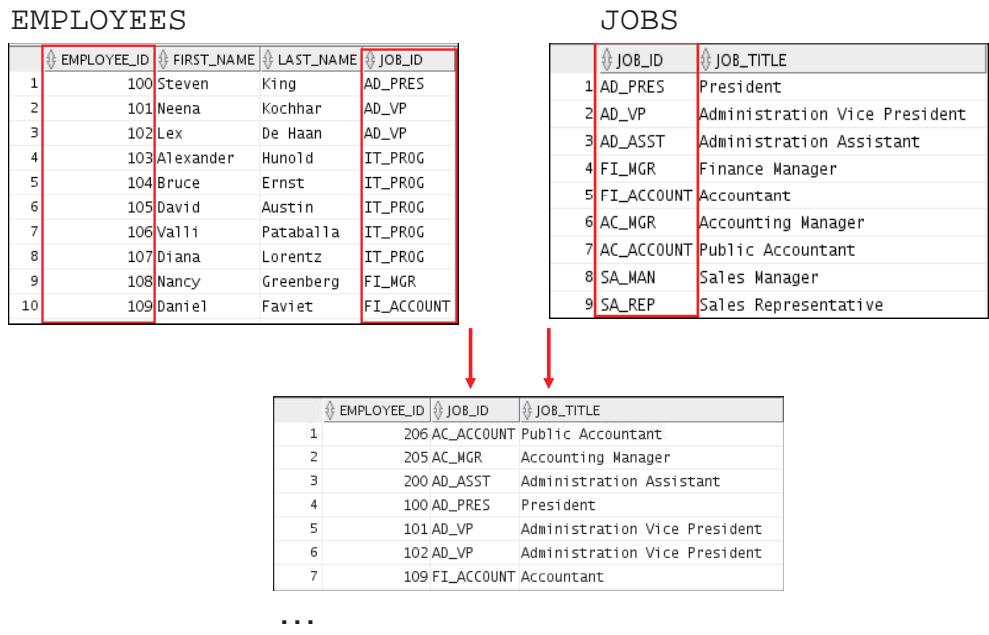
## Displaying Data from Multiple Tables Using Joins

### 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

# Obtaining Data from Multiple Tables



3 - 3

## Types of Joins

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

- Natural join with the NATURAL JOIN clause
- Join with the USING clause
- Join with the ON clause
- OUTER joins:
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN
- Cross joins

3 - 4

## 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];
```

3 - 5

## Creating Natural Joins

- The NATURAL JOIN clause is based on all the columns that have the same name in two tables.
- 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.

```
SELECT * FROM table1 NATURAL JOIN table2;
```

3 - 6

## Retrieving Records with Natural Joins

```
SELECT employee_id, first_name, job_id, job_title  
from employees NATURAL JOIN jobs;
```

	EMPLOYEE_ID	FIRST_NAME	JOB_ID	JOB_TITLE
1	100 Steven		AD_PRES	President
2	101 Neena		AD_VP	Administration Vice President
3	102 Lex		AD_VP	Administration Vice President
4	103 Alexander		IT_PROG	Programmer
5	104 Bruce		IT_PROG	Programmer
6	105 David		IT_PROG	Programmer
7	106 Valli		IT_PROG	Programmer
8	107 Diana		IT_PROG	Programmer
9	108 Nancy		FI_MGR	Finance Manager
10	109 Daniel		FI_ACCOUNT	Accountant
11	110 John		FI_ACCOUNT	Accountant

...

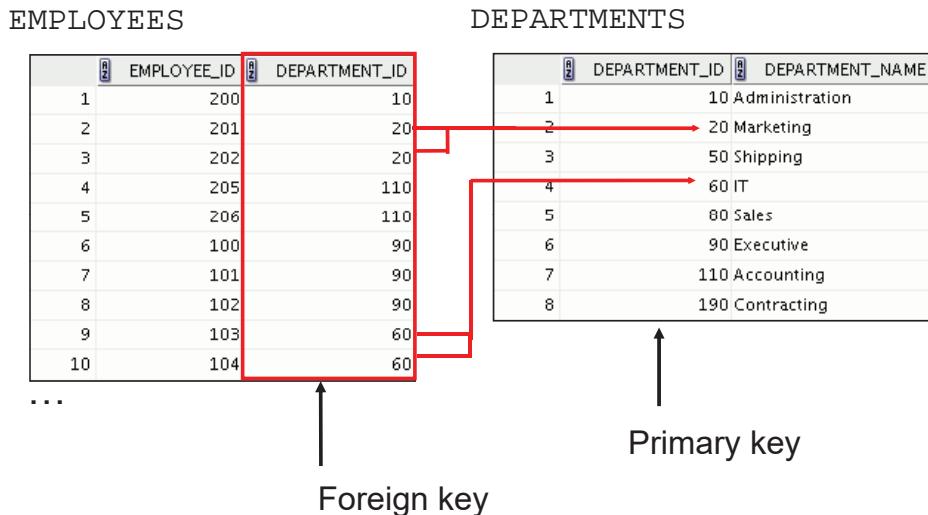
3 - 7

## Creating Joins with the USING Clause

- If several columns have the same names but the data types do not match, use the `USING` clause to specify the columns for the equijoin.
- Use the `USING` clause to match only one column when more than one column matches.

3 - 8

## Joining Column Names



3 - 9

## 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	144	Vargas	1500	50
5	143	Matos	1500	50
6	142	Davies	1500	50
7	141	Rajs	1500	50
8	124	Mourgos	1500	50
...				
18	206	Gietz	1700	110
19	205	Higgins	1700	110

3 - 10

## Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to increase the speed of parsing of the statement.
- 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.

3 - 11

## Using Table Aliases with the USING Clause

- Do not qualify a column that is used in the NATURAL join or a join with a USING clause.
- If the same column is used elsewhere 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;
```

ORA-25154: column part of USING clause cannot have qualifier  
25154. 00000 - "column part of USING clause cannot have qualifier"  
\*Cause: Columns that are used for a named-join (either a NATURAL join  
or a join with a USING clause) cannot have an explicit qualifier.  
\*Action: Remove the qualifier.  
Error at Line: 4 Column: 6

3 - 12

## 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.

3 - 13

## 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
 WHERE 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
11	103	Hunold	60	60	1400
...					

3 - 14

## Creating Three-Way Joins

```
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
9	142	South San Francisco	Shipping

...

3 - 15

## Applying Additional Conditions to a Join

Use the AND clause or the WHERE clause to apply additional conditions:

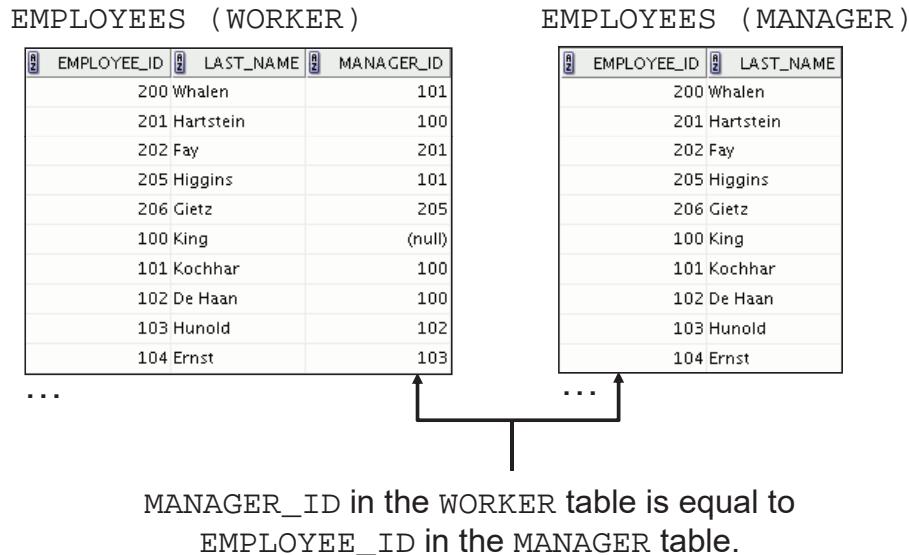
```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
  FROM employees e JOIN departments d
    ON (e.department_id = d.department_id)
   AND e.manager_id = 149 ;
```

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 ;
```

3 - 16

## Joining a Table to Itself



3 - 17

## 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

...

3 - 18

## Nonequijoins

EMPLOYEES

	LAST_NAME	SALARY
1	Whalen	4400
2	Hartstein	13000
3	Fay	6000
4	Higgins	12000
5	Gietz	8300
6	King	24000
7	Kochhar	17000
8	De Haan	17000
9	Hunold	9000
10	Ernst	6000
...		
19	Taylor	8600
20	Grant	7000

JOB\_GRADES

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

The JOB\_GRADES table defines the LOWEST\_SAL and HIGHEST\_SAL range of values for each GRADE\_LEVEL. Therefore, the GRADE\_LEVEL column can be used to assign grades to each employee.

3 - 19

## Retrieving Records with Nonequijoins

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

	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Mourgos	5800	B
8	Ernst	6000	C
9	Fay	6000	C
10	Grant	7000	C
...			

3 - 20

## Returning Records with No Direct Match Using OUTER Joins

DEPARTMENTS

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

Equijoin with EMPLOYEES

	DEPARTMENT_ID	LAST_NAME
1	10	Whalen
2	20	Hartstein
3	20	Fay
4	110	Higgins
5	110	Gietz
6	90	King
7	90	Kochhar
8	90	De Haan
9	60	Hunold
10	60	Ernst
...		
18	80	Abel
19	80	Taylor

There are no employees in department 190.

Employee "Grant" has not been assigned a department ID.

3 - 21

## 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
...		
16 Kochhar	90	Executive
17 King	90	Executive
18 Gietz	110	Accounting
19 Higgins	110	Accounting
20 Grant	(null)	(null)

3 - 22

## RIGHT OUTER JOIN

```
SELECT e.last_name, d.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 Davies	50	Shipping
5 Vargas	50	Shipping
6 Raji	50	Shipping
7 Mourgos	50	Shipping
8 Matos	50	Shipping
...		
18 Higgins	110	Accounting
19 Gietz	110	Accounting
20 (null)	190	Contracting

3 - 23

## 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 King	90	Executive
2 Kochhar	90	Executive
3 De Haan	90	Executive
4 Hunold	60	IT
...		
15 Grant	(null)	(null)
16 Whalen	10	Administration
17 Hartstein	20	Marketing
18 Fay	20	Marketing
19 Higgins	110	Accounting
20 Gietz	110	Accounting
21 (null)	190	Contracting

3 - 24

## Cartesian Products

- Cartesian product is a join of every row of one table to every row of another table.
- A Cartesian product generates a large number of rows and the result is rarely useful.

3 - 25

## Generating a Cartesian Product

EMPLOYEES (20 rows)

#	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110
...			
19	176	Taylor	80
20	178	Grant	(null)

DEPARTMENTS (8 rows)

#	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

Cartesian product:

$20 \times 8 = 160$  rows

#	EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	200	10	1700
2	201	20	1700
...			
21	200	10	1800
22	201	20	1800
...			
159	176	80	1700
160	178	(null)	1700

3 - 26

## Creating Cross Joins

- A CROSS JOIN is a JOIN operation that produces the Cartesian product of two tables.
- To create a Cartesian product, specify the CROSS JOIN in your SELECT statement.

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

	LAST_NAME	DEPARTMENT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
5	Fay	Administration
...		
158	Vargas	Contracting
159	Whalen	Contracting
160	Zlotkey	Contracting

3 - 27

## Summary

In this lesson, you should have learned how to :

- 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

3 - 28