

7 Displaying Data from Multiple Tables Using Joins

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

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

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This lesson explains how to obtain data from more than one table. A *join* is used to view information from multiple tables. Therefore, you can *join* tables together to view information from more than one table.

Note: Information about joins is found in the “SQL Queries and Subqueries: Joins” section in *Oracle Database SQL Language Reference* for 12c database.

Lesson Agenda

- Types of JOINS and their syntax
- Natural join
- Join with the USING clause
- Join with the ON clause
- Self-join
- Nonequijoins
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Obtaining Data from Multiple Tables

EMPLOYEES

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	JOB_ID
1	100	Steven	King	AD_PRES
2	101	Neena	Kochhar	AD_VP
3	102	Lex	De Haan	AD_VP
4	103	Alexander	Hunold	IT_PROG
5	104	Bruce	Ernst	IT_PROG
6	105	David	Austin	IT_PROG
7	106	Valli	Pataballa	IT_PROG
8	107	Diana	Lorentz	IT_PROG
9	108	Nancy	Greenberg	FI_MGR
10	109	Daniel	Faviet	FI_ACCOUNT

JOBS

	JOB_ID	JOB_TITLE
1	AD_PRES	President
2	AD_VP	Administration Vice President
3	AD_ASST	Administration Assistant
4	FI_MGR	Finance Manager
5	FI_ACCOUNT	Accountant
6	AC_MGR	Accounting Manager
7	AC_ACCOUNT	Public Accountant
8	SA_MAN	Sales Manager
9	SA_REP	Sales Representative

↓ ↓

	EMPLOYEE_ID	JOB_ID	JOB_TITLE
1	206	AC_ACCOUNT	Public Accountant
2	205	AC_MGR	Accounting Manager
3	200	AD_ASST	Administration Assistant
4	100	AD_PRES	President
5	101	AD_VP	Administration Vice President
6	102	AD_VP	Administration Vice President
7	109	FI_ACCOUNT	Accountant

...

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Sometimes you need to use data from more than one table. In the example in the slide, the report displays data from two separate tables:

- Employees IDs exist in the EMPLOYEES table.
- Job IDs exist in both the EMPLOYEES and JOBS tables.
- Job title exist in the JOBS table.

To produce the report, you need to link the EMPLOYEES and JOBS tables, and access data from both of them.

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

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To join tables, you can use a join syntax that is compliant with the SQL:1999 standard.

Note

- Before the Oracle9i release, the Oracle join syntax was different from the American National Standards Institute (ANSI) standards. The SQL:1999–compliant join syntax does not offer any performance benefits over the Oracle-proprietary join syntax that existed in the prior releases.
- The following slide discusses the SQL:1999 join syntax.

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];
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the syntax:

- `table1.column` denotes the table and the column from which data is retrieved
- `NATURAL JOIN` joins two tables based on the same column name
- `JOIN table2 USING column_name` performs an equijoin based on the column name
- `JOIN table2 ON table1.column_name = table2.column_name` performs an equijoin based on the condition in the `ON` clause
- `LEFT/RIGHT/FULL OUTER` is used to perform OUTER joins
- `CROSS JOIN` returns a Cartesian product from the two tables

For more information, see the section titled “SELECT” in *Oracle Database SQL Language Reference* for 12c database.

Lesson Agenda

- Types of JOINS and their syntax
- Natural join
- Join with the USING clause
- Join with the ON clause
- Self-join
- Nonequijoins
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

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;
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can join tables automatically based on the columns in the two tables that have matching data types and names. You do this by using the `NATURAL JOIN` keywords.

Note: The join can happen on only those columns that have the same names and data types in both tables. If the columns have the same name but different data types, the `NATURAL JOIN` syntax causes an error.

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

...

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the example in the slide, the JOBS table is joined to the EMPLOYEES table by the JOB_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.

Natural Joins with a WHERE Clause

Additional restrictions on a natural join are implemented by using a WHERE clause. The following example limits the rows of output to those with a DEPARTMENT_ID equal to 20 or 50:

```
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, 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.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Natural joins use all columns with matching names and data types to join the tables. The `USING` clause can be used to specify only those columns that should be used for an equijoin.

Joining Column Names

EMPLOYEES

	EMPLOYEE_ID	DEPARTMENT_ID
1	200	10
2	201	20
3	202	20
4	205	110
5	206	110
6	100	90
7	101	90
8	102	90
9	103	60
10	104	60

...

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

Foreign key

Primary key

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

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 in both the tables must be equal. Frequently, this type of join involves primary and foreign key complements.

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

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the example in the slide, the DEPARTMENT_ID columns in the EMPLOYEES and DEPARTMENTS tables are joined and thus the LOCATION_ID of the department where an employee works is shown.

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.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When joining two or more tables, you need to qualify the names of the columns with the table name to avoid ambiguity. Without the table prefixes, the `DEPARTMENT_ID` column in the `SELECT` list could be from either the `DEPARTMENTS` table or the `EMPLOYEES` table. It is necessary to add the table prefix to execute your query. If there are no common column names between the two tables, there is no need to qualify the columns. However, using the table prefix increases the speed of parsing of the statement, because you tell the Oracle server exactly where to find the columns.

However, qualifying column names with table names can be time consuming, particularly if the table names are lengthy. Instead, you can use *table aliases*. Just as a column alias gives a column another name, a table alias gives a table another name. Table aliases help to keep SQL code smaller, therefore, using less memory.

The table name is specified in full, followed by a space, and then the table alias. For example, the `EMPLOYEES` table can be given an alias of `e`, and the `DEPARTMENTS` table an alias of `d`.

Guidelines

- Table aliases can be up to 30 characters in length, but shorter aliases are better than longer ones.
- If a table alias is used for a particular table name in the `FROM` clause, that table alias must be substituted for the table name throughout the `SELECT` statement.
- Table aliases should be meaningful.
- The table alias is valid for only the current `SELECT` statement.

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
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

When joining with the USING clause, you cannot qualify a column that is used in the USING clause itself. Furthermore, if that column is used anywhere in the SQL statement, you cannot alias it. For example, in the query mentioned in the slide, you should not alias the `location_id` column in the WHERE clause because the column is used in the USING clause. The columns that are referenced in the USING clause should not have a qualifier (table name or alias) anywhere in the SQL statement. For example, the following statement is valid:

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

The columns that are common in both the tables, but not used in the USING clause, must be prefixed with a table alias; otherwise, you get the “column ambiguously defined” error.

In the following statement, `manager_id` is present in both the `employees` and `departments` table; if `manager_id` is not prefixed with a table alias, it gives a “column ambiguously defined” error.

The following statement is valid:

```
SELECT first_name, e.department_name, d.manager_id
FROM   employees e JOIN departments d USING (department_id)
WHERE  department_id = 50;
```

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.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Use the `ON` clause to specify a join condition. With this, you can specify join conditions separate from any search or filter conditions in the `WHERE` clause.

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
11	103	Hunold	60	60	1400

...

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In this example, the DEPARTMENT_ID columns in the EMPLOYEES and DEPARTMENTS table are joined using the ON clause. Wherever a department ID in the EMPLOYEES table equals a department ID in the DEPARTMENTS table, the row is returned. The table alias is necessary to qualify the matching column_names.

You can also use the ON clause to join columns that have different names. The parentheses around the joined columns, as in the example in the slide, (e.department_id = d.department_id) is optional. So, even ON e.department_id = d.department_id will work.

Note: When you use the Execute Statement icon to run the query, SQL Developer suffixes a '_1' to differentiate between the two department_ids.

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

...

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A three-way join is a join of three tables. Here, the first join to be performed is `EMPLOYEES JOIN DEPARTMENTS`. The first join condition can reference columns in `EMPLOYEES` and `DEPARTMENTS` but cannot reference columns in `LOCATIONS`. The second join condition can reference columns from all three tables.

Note: The code example in the slide can also be accomplished with the `USING` clause:

```
SELECT e.employee_id, l.city, d.department_name
FROM   employees e
JOIN   departments d
USING (department_id)
JOIN   locations l
USING (location_id);
```

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 ;
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You can apply additional conditions to the join.

The example shown performs a join on the **EMPLOYEES** and **DEPARTMENTS** tables and, in addition, displays only employees who have a manager ID of 149. To add additional conditions to the **ON** clause, you can add **AND** clauses. Alternatively, you can use a **WHERE** clause to apply additional conditions.

Both the queries produce the same output.

Lesson Agenda

- Types of JOINS and their syntax
- Natural join
- Join with the USING clause
- Join with the ON clause
- **Self-join**
- Nonequijoins
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Joining a Table to Itself

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
200	Whalen	101
201	Hartstein	100
202	Fay	201
205	Higgins	101
206	Gietz	205
100	King	(null)
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
200	Whalen
201	Hartstein
202	Fay
205	Higgins
206	Gietz
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst

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

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Sometimes you need to join a table to itself. To find the name of each employee's manager, you need to join the `EMPLOYEES` table to itself, or perform a self-join. For example, to find the name of Ernst's manager, you need to:

- Find Ernst in the `EMPLOYEES` table by looking at the `LAST_NAME` column
- Find the manager number for Ernst by looking at the `MANAGER_ID` column. Ernst's manager number is 103.
- Find the name of the manager with `EMPLOYEE_ID` 103 by looking at the `LAST_NAME` column. Hunold's employee number is 103, so Hunold is Ernst's manager.

In this process, you look in the table twice. The first time you look in the table to find Ernst in the `LAST_NAME` column and the `MANAGER_ID` value of 103. The second time you look in the `EMPLOYEE_ID` column to find 103 and the `LAST_NAME` column to find Hunold.

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

...

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The ON clause can also be used to join columns that have different names, within the same table or in a different table.

The example shown is a self-join of the EMPLOYEES table, based on the EMPLOYEE_ID and MANAGER_ID columns.

Lesson Agenda

- Types of JOINS and their syntax
- Natural join
- Join with the USING clause
- Join with the ON clause
- Self-join
- **Nonequijoins**
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

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.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A nonequijoin is a join condition containing something other than an equality operator.

The relationship between the EMPLOYEES table and the JOB_GRADES table is an example of a nonequijoin. The SALARY column in the EMPLOYEES table ranges between the values in the LOWEST_SAL and HIGHEST_SAL columns of the JOB_GRADES table. Therefore, each employee can be graded based on their salary. The relationship is obtained using an operator other than the equality (=) operator.

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

...

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The example in the slide creates a nonequijoin to evaluate an employee's salary grade. The salary must be *between* any pair of the low and high salary ranges.

It is important to note that all employees appear exactly once when this query is executed. No employee is repeated in the list. There are two reasons for this:

- None of the rows in the `JOB_GRADES` table contain grades that overlap. That is, the salary value for an employee can lie only between the low-salary and high-salary values of one of the rows in the salary grade table.
- All of the employees' salaries lie within the limits provided by the job grade table. That is, no employee earns less than the lowest value contained in the `LOWEST_SAL` column or more than the highest value contained in the `HIGHEST_SAL` column.

Note: Other conditions (such as `<=` and `>=`) can be used, but `BETWEEN` is the simplest.

Remember to specify the low value first and the high value last when using the `BETWEEN` condition. The Oracle server translates the `BETWEEN` condition to a pair of `AND` conditions.

Therefore, using `BETWEEN` has no performance benefits, but should be used only for logical simplicity.

Table aliases have been specified in the slide example for performance reasons, not because of possible ambiguity.

Lesson Agenda

- Types of JOINS and their syntax
- Natural join
- Join with the USING clause
- Join with the ON clause
- Self-join
- Nonequi joins
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

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

There are no employees in department 190.

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

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

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

If a row does not satisfy a join condition, the row does not appear in the query result.

In the slide example, a simple equijoin condition is used on the EMPLOYEES and DEPARTMENTS tables to return the result on the right. The result set does not contain the following:

- Department ID 190, because there are no employees with that department ID recorded in the EMPLOYEES table
- The employee with the last name of Grant, because this employee has not been assigned a department ID

To return the department record that does not have any employees, or employees that do not have an assigned department, you can use an OUTER join.

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.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Joining tables with the `NATURAL JOIN`, `USING`, or `ON` clauses results in an `INNER` join. Any unmatched rows are not displayed in the output. To return the unmatched rows, you can use an `OUTER` join. An `OUTER` join returns all rows that satisfy the join condition and also returns some or all of those rows from one table for which no rows from the other table satisfy the join condition.

There are three types of `OUTER` joins:

- `LEFT OUTER`
- `RIGHT OUTER`
- `FULL OUTER`

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)

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This query retrieves all the rows in the EMPLOYEES table, which is the left table, even if there is no match in the DEPARTMENTS table.

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	Rajs	50	Shipping
7	Mourgos	50	Shipping
8	Matos	50	Shipping

...

18	Higgins	110	Accounting
19	Gietz	110	Accounting
20	(null)	190	Contracting

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This query retrieves all the rows in the DEPARTMENTS table, which is the table at the right, even if there is no match in the EMPLOYEES table.

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

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

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.

Lesson Agenda

- Types of JOINS and their syntax
- Natural join
- Join with the USING clause
- Join with the ON clause
- Self-join
- Nonequijoins
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

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.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A Cartesian product tends to generate a large number of rows and the result is rarely useful. You should, therefore, always include a valid join condition 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.

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 x 8 = 160 rows

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

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A Cartesian product is generated if a join condition is omitted. The example in the slide displays the employee last name and the department name from the EMPLOYEES and DEPARTMENTS tables. Because no join condition was specified, all rows (20 rows) from the EMPLOYEES table are joined with all rows (8 rows) in the DEPARTMENTS table, thereby generating 160 rows in the output.

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

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The example in the slide produces a Cartesian product of the `EMPLOYEES` and `DEPARTMENTS` tables.

It is a good practice to explicitly state `CROSS JOIN` in your `SELECT` when you intend to create a Cartesian product. Therefore, it is very clear that you intend for this to happen and it is not the result of missing joins.

Quiz

If you join a table to itself, what kind of join are you using?

- a. Nonequijoin
- b. Left OUTER join
- c. Right OUTER join
- d. Full OUTER join
- e. Self-join
- f. Natural join
- g. Cartesian products

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Answer: e

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

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

There are multiple ways to join tables.

Types of Joins

- Equijoins
- Nonequijoins
- `OUTER` joins
- Self-joins
- Cross joins
- Natural joins
- Full (or two-sided) `OUTER` joins

Cartesian Products

A Cartesian product results in the display of all combinations of rows. This is done by either omitting the `WHERE` clause or specifying the `CROSS JOIN` clause.

Table Aliases

- Table aliases speed up database access.
- They can help to keep SQL code smaller by conserving memory.
- They are sometimes mandatory to avoid column ambiguity.

Practice 7: Overview

This practice covers the following topics:

- Joining tables using an equijoin
- Performing outer and self-joins
- Adding conditions

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This practice is intended to give you experience in extracting data from more than one table using the SQL:1999–compliant joins.

YASINEB MAZOUZ (z.m.malak@hotmail.com) has a non-transferable
license to use this Student Guide.