Oracle Join Syntax

ORACLE

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
- Use outer joins to view data that generally does not meet a join condition
- Join a table to itself by using a self-join

ORACLE

Objectives

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 on joins is found in "SQL Queries and Subqueries: Joins" in *Oracle SQL Reference*.

Obtaining Data from Multiple Tables

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110
...			
18	174	Abel	80
19	176	Taylor	80
20	178	Grant	(null)

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	100	90	Executive
2	101	90	Executive
...			
17	202	20	Marketing
18	205	110	Accounting
19	206	110	Accounting

Data from Multiple Tables

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

- Employee IDs exist in the `EMPLOYEES` table.
- Department IDs exist in both the `EMPLOYEES` and `DEPARTMENTS` tables.
- Department names 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.

ORACLE

Cartesian Products

When a join condition is invalid or omitted completely, the result is a *Cartesian product,* in which all combinations of rows are displayed. All rows in the first table are joined to all rows in the second table.

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 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
...			
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	100	10	1700
2	101	10	1700
...			
156	200	190	1700
157	201	190	1700
158	202	190	1700
159	205	190	1700
160	206	190	1700

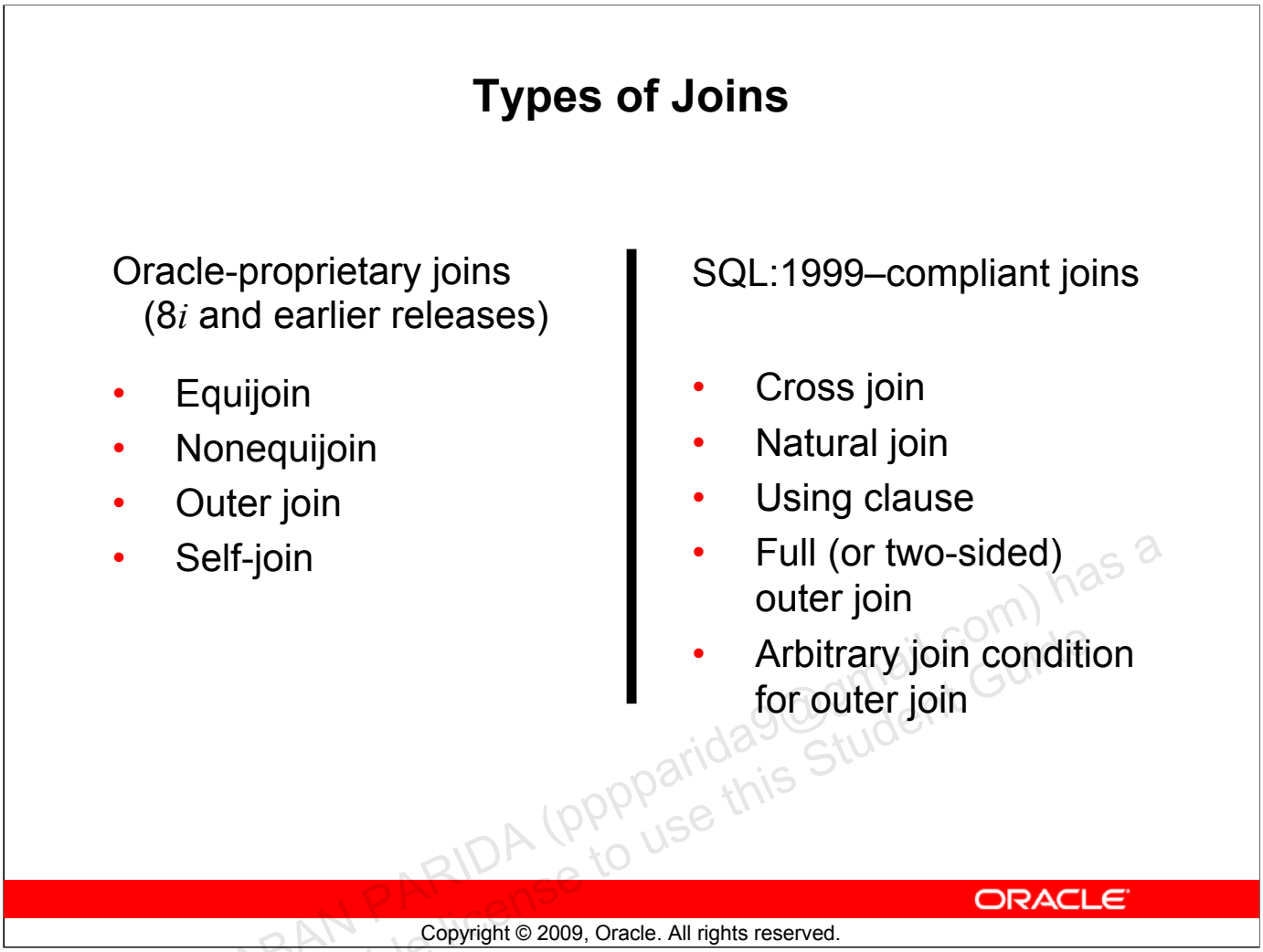
ORACLE

Cartesian Products (continued)

A Cartesian product is generated if a join condition is omitted. The example in the slide displays employee last name and department name from the EMPLOYEES and DEPARTMENTS tables. Because no join condition has been 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.

```
SELECT last_name, department_name dept_name
FROM   employees, departments;
```

	LAST_NAME	DEPT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
...		
160	Zlotkey	Contracting



Types of Joins

Before the release of Oracle9*i* Database, the join syntax was proprietary.

Note: The SQL:1999–compliant join syntax does not offer any performance benefits over the Oracle-proprietary join syntax that existed in prior releases. For detailed information about the SQL:1999–compliant join syntax, see Lesson 5.

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.

ORACLE

Defining Joins

When data from more than one table in the database is required, a *join* condition is used. Rows in one table can be joined to rows in another table according to common values that exist in corresponding columns (that is, usually primary and foreign key columns).

To display data from two or more related tables, write a simple join condition in the `WHERE` clause.

In the syntax:

table1.column	denotes the table and column from which data is retrieved
table1.column1 = table2.column2	is the condition that joins (or relates) the tables together

Guidelines

- 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.

Equijoins

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
11	107	60
12	124	50
13	141	50
14	142	50
15	143	50
16	144	50
17	149	80
18	174	80
19	176	80
20	178	(null)

Foreign key

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	20	Marketing
4	50	Shipping
5	50	Shipping
6	50	Shipping
7	50	Shipping
8	50	Shipping
9	60	T
10	60	T
11	60	T
12	80	Sales
13	80	Sales
14	80	Sales
15	90	Executive
16	90	Executive
17	90	Executive
18	110	Accounting
19	110	Accounting

Primary key

ORACLE

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 in 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*.

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;
```

	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	205	Higgins	110	110	1700
...					
18	174	Abel	80	80	2500
19	176	Taylor	80	80	2500

ORACLE

Retrieving Records with Equijoins

In the slide example:

- **The `SELECT` clause specifies the column names to retrieve:**
  - Employee last name, employee number, and department number, which are columns in the `EMPLOYEES` table
  - Department number, department name, and location ID, which are columns in the `DEPARTMENTS` table
- **The `FROM` clause specifies the two tables that the database must access:**
  - `EMPLOYEES` table
  - `DEPARTMENTS` table
- **The `WHERE` clause specifies how the tables are to be joined:**

```
EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID
```

Because the `DEPARTMENT_ID` column is common to both tables, it must be prefixed by the table name to avoid ambiguity.

Additional Search Conditions Using the AND Operator

EMPLOYEES

	LAST_NAME	DEPARTMENT_ID
1	Whalen	10
2	Hartstein	20
3	Fay	20
4	Vargas	50
5	Matos	50
6	Davies	50
7	Rajs	50
8	Mourgos	50
9	Hunold	60
10	Ernst	60

...

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	20	Marketing
4	50	Shipping
5	50	Shipping
6	50	Shipping
7	50	Shipping
8	50	Shipping
9	60	IT
10	60	IT

...

Additional Search Conditions

In addition to the join, you may have criteria for your WHERE clause to restrict the rows under consideration for one or more tables in the join. For example, to display employee Matos's 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';
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Matos	50	Shipping

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to improve performance.
- Use column aliases to distinguish columns that have identical names but reside in different tables.

ORACLE

Qualifying Ambiguous Column Names

You need to qualify the names of the columns in the `WHERE` clause with the table name to avoid ambiguity. Without the table prefixes, the `DEPARTMENT_ID` column 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 improves performance, because you tell the Oracle server exactly where to find the columns.

The requirement to qualify ambiguous column names is also applicable to columns that may be ambiguous in other clauses, such as the `SELECT` clause or the `ORDER BY` clause.

Using Table Aliases

- Use table aliases to simplify queries.
- Use table prefixes to improve performance.

```
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;
```

ORACLE

Using Table Aliases

Qualifying column names with table names can be very time consuming, particularly if table names are lengthy. You can use *table aliases* instead of table names. 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.

Notice how table aliases are identified in the `FROM` clause in the example. The table name is specified in full, followed by a space and then the table alias. The `EMPLOYEES` table has been given an alias of `e`, and the `DEPARTMENTS` table has 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, then 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.

Joining More Than Two Tables

EMPLOYEES

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

...

DEPARTMENTS

	DEPARTMENT_ID	LOCATION_ID
1	10	1700
2	20	1800
3	50	1500
4	60	1400
5	80	2500
6	90	1700
7	110	1700
8	190	1700

LOCATIONS

	LOCATION_ID	CITY
1	1400	Southlake
2	1500	South San Francisco
3	1700	Seattle
4	1800	Toronto
5	2500	Oxford

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.

ORACLE

Additional Search Conditions

Sometimes you may need to join more than two tables. For example, to display the last name, the department name, and the city for each employee, you have to join the `EMPLOYEES`, `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;
```

	LAST_NAME	DEPARTMENT_NAME	CITY
1	Whalen	Administration	Seattle
2	Hartstein	Marketing	Toronto
3	Fay	Marketing	Toronto
4	Higgins	Accounting	Seattle
5	Gietz	Accounting	Seattle
6	King	Executive	Seattle
7	Kochhar	Executive	Seattle

...

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

...

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

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

ORACLE

Nonequijoins

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. 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 operator other than equality (=).

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;
```

	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

...

Nonequijoins (continued)

The slide example 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 grade 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 that are 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 `BETWEEN`.

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

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

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

...

There are no employees in department 190.

Returning Records with No Direct Match with Outer Joins

If a row does not satisfy a join condition, the row does not appear in the query result. For example, in the equijoin condition of the EMPLOYEES and DEPARTMENTS tables, employee Grant does not appear because there is no department ID recorded for her in the EMPLOYEES table. Instead of seeing 20 employees in the result set, you see 19 records.

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

Outer Joins Syntax

- You use an outer join to 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(+);
```


ORACLE

Using Outer Joins to Return Records with No Direct Match

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.

In the syntax:

`table1.column =` is the condition that joins (or relates) the tables together

`table2.column (+)` is the outer join symbol, which can be placed on either side of the `WHERE` clause condition, but not on both sides. (Place the outer join symbol following the name of the column in the table without the matching rows.)

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 ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Higgins	110	Accounting
...			
18	Abel	80	Sales
19	Taylor	80	Sales
20	(null)	(null)	Contracting

ORACLE

Using Outer Joins to Return Records with No Direct Match (continued)

The slide example displays employee last names, department IDs, and department names. The Contracting department does not have any employees. The empty value is shown in the output.

Outer Join Restrictions

- 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-Joins

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

...

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

ORACLE

Joining a Table to Itself

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; this type of join is called a *self-join*.

For example, to find the name of Lorentz's manager, you need to do the following:

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

In this process, you look in the table twice. The first time you look in the table to find Lorentz 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.

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;
```

	WORKER.LAST_NAME||'WORKSFOR'||MANAGER.LAST_NAME
1	Fay works for Hartstein
2	Gietz works for Higgins
3	Zlotkey works for King
4	Mourgos works for King
5	De Haan works for King
6	Kochhar works for King

...

Joining a Table to Itself (continued)

The slide example joins the `EMPLOYEES` table to itself. To simulate two tables in the `FROM` clause, there are two aliases, namely w and m, for the same table, `EMPLOYEES`.

In this example, the `WHERE` clause contains the join that means "where a worker's manager number matches the employee number for the manager."

Summary

In this appendix, you should have learned how to use joins to display data from multiple tables by using Oracle-proprietary syntax for versions 8*i* and earlier.

ORACLE

Summary

There are multiple ways to join tables.

Types of Joins

- Cartesian products
- Equijoins
- Nonequijoins
- Outer joins
- Self-joins

Cartesian Products

A Cartesian product results in a display of all combinations of rows. This is done by omitting the `WHERE` clause.

Table Aliases

- Table aliases speed up database access.
- Table aliases can help to keep SQL code smaller by conserving memory.

Practice C: Overview

This practice covers writing queries to join tables using Oracle syntax.

Practice C: Overview

This practice is designed to give you a variety of exercises that join tables using the Oracle syntax that was covered in this appendix.

Practice C

1. Write a query for the HR department to produce the addresses of all the departments. Use the `LOCATIONS` and `COUNTRIES` tables. Show the location ID, street address, city, state or province, and country in the output.

	LOCATION_ID	STREET_ADDRESS	CITY	STATE_PROVINCE	COUNTRY_NAME
1	1400	2014 Jabberwocky Rd	Southlake	Texas	United States of America
2	1500	2011 Interiors Blvd	South San Francisco	California	United States of America
3	1700	2004 Charade Rd	Seattle	Washington	United States of America
4	1800	460 Bloor St. W.	Toronto	Ontario	Canada
5	2500	Magdalen Centre, The Oxford Science Park	Oxford	Oxford	United Kingdom

2. The HR department needs a report of all employees. Write a query to display the last name, department number, and department name for all employees.

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Higgins	110	Accounting
5	Gietz	110	Accounting
6	King	90	Executive
7	Kochhar	90	Executive
8	De Haan	90	Executive
9	Hunold	60	IT
10	Ernst	60	IT
11	Lorentz	60	IT
12	Mourgos	50	Shipping
13	Rajs	50	Shipping
14	Davies	50	Shipping
15	Matos	50	Shipping
16	Vargas	50	Shipping
17	Zlotkey	80	Sales
18	Abel	80	Sales
19	Taylor	80	Sales

Practice C (continued)

3. The HR department needs a report of employees in Toronto. Display the last name, job, department number, and department name for all employees who work in Toronto.

	LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	Hartstein	MK_MAN	20	Marketing
2	Fay	MK_REP	20	Marketing

4. Create a report to display the employee last name and employee number along with the employee's 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 `lab_c_04.sql`.

	Employee	EMP#	Manager	Mgr#
1	Fay	202	Hartstein	201
2	Gietz	206	Higgins	205
3	Zlotkey	149	King	100
4	Mourgos	124	King	100
5	De Haan	102	King	100
6	Kochhar	101	King	100
7	Hartstein	201	King	100
8	Higgins	205	Kochhar	101
9	Whalen	200	Kochhar	101
10	Hunold	103	De Haan	102
11	Lorentz	107	Hunold	103
12	Ernst	104	Hunold	103
13	Vargas	144	Mourgos	124
14	Matos	143	Mourgos	124
15	Davies	142	Mourgos	124
16	Rajs	141	Mourgos	124
17	Grant	178	Zlotkey	149
18	Taylor	176	Zlotkey	149
19	Abel	174	Zlotkey	149

Practice C (continued)

5. Modify `lab_c_04.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 `lab_c_05.sql`. Run the query in `lab_c_05.sql`.

	Employee	EMP#	Manager	Mgr#
1	Fay	202	Hartstein	201
2	Gietz	206	Higgins	205
3	Zlotkey	149	King	100
4	Mourgos	124	King	100
5	De Haan	102	King	100
6	Kochhar	101	King	100

▪ ▪ ▪

19	Abel	174	Zlotkey	149
20	King	100	(null)	(null)

6. Create a report for the HR department 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. Save the script to a file named `lab_c_06.sql`.

	DEPARTMENT	EMPLOYEE	COLLEAGUE
1	20	Fay	Hartstein
2	20	Hartstein	Fay
3	50	Davies	Matos
4	50	Davies	Mourgos
5	50	Davies	Rajs

▪ ▪ ▪

41	110	Gietz	Higgins
42	110	Higgins	Gietz

Practice C (continued)

7. The HR department needs a report on job grades and salaries. To familiarize yourself with the JOB_GRADES table, first show the structure of the JOB_GRADES table. Second, create a query that displays the last name, job, department name, salary, and grade for all employees.

```
Name                           Null     Type
------------------------------ -------- ------------
GRADE_LEVEL                             VARCHAR2(3)
LOWEST_SAL                              NUMBER
HIGHEST_SAL                             NUMBER
```

	LAST_NAME	JOB_ID	DEPARTMENT_NAME	SALARY	GRADE_LEVEL
1	Vargas	ST_CLERK	Shipping	2500	A
2	Matos	ST_CLERK	Shipping	2600	A
3	Davies	ST_CLERK	Shipping	3100	B
4	Rajs	ST_CLERK	Shipping	3500	B
5	Lorentz	IT_PROG	IT	4200	B
6	Whalen	AD_ASST	Administration	4400	B
7	Mourgos	ST_MAN	Shipping	5800	B
8	Ernst	IT_PROG	IT	6000	C
9	Fay	MK_REP	Marketing	6000	C
10	Gietz	AC_ACCOUNT	Accounting	8300	C
11	Taylor	SA_REP	Sales	8600	C
12	Hunold	IT_PROG	IT	9000	C
13	Zlotkey	SA_MAN	Sales	10500	D
14	Abel	SA_REP	Sales	11000	D
15	Higgins	AC_MGR	Accounting	12000	D
16	Hartstein	MK_MAN	Marketing	13000	D
17	De Haan	AD_VP	Executive	17000	E
18	Kochhar	AD_VP	Executive	17000	E
19	King	AD_PRES	Executive	24000	E

Practice C (continued)

If you want an extra challenge, complete the following exercises:

8. The HR department wants to determine the names of all employees hired after Davies. Create a query to display the name and hire date of any employee hired after employee Davies.

	LAST_NAME	HIRE_DATE
1	Fay	17-AUG-97
2	Lorentz	07-FEB-99
3	Mourgos	16-NOV-99
4	Matos	15-MAR-98
5	Vargas	09-JUL-98
6	Zlotkey	29-JAN-00
7	Taylor	24-MAR-98
8	Grant	24-MAY-99

9. The HR department needs to find the name and hire date for all employees who were hired before their managers, along with their manager's name and hire date. Label the columns `Employee`, `Emp Hired`, `Manager`, and `Mgr Hired`, respectively. Save the script to a file named `lab_c_09.sql`.

	LAST_NAME	HIRE_DATE	LAST_NAME_1	HIRE_DATE_1
1	Whalen	17-SEP-87	Kochhar	21-SEP-89
2	Hunold	03-JAN-90	De Haan	13-JAN-93
3	Vargas	09-JUL-98	Mourgos	16-NOV-99
4	Matos	15-MAR-98	Mourgos	16-NOV-99
5	Davies	29-JAN-97	Mourgos	16-NOV-99
6	Rajs	17-OCT-95	Mourgos	16-NOV-99
7	Grant	24-MAY-99	Zlotkey	29-JAN-00
8	Taylor	24-MAR-98	Zlotkey	29-JAN-00
9	Abel	11-MAY-96	Zlotkey	29-JAN-00