5

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 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

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

ORACLE

Obtaining 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.

Types of Joins

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

- Cross joins
- Natural joins
- `USING` clause
- Full (or two-sided) outer joins
- Arbitrary join conditions for outer joins

Types of Joins

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

Note: Before the Oracle9*i* release, the join syntax was different from the ANSI standards. 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 proprietary join syntax, see Appendix C.

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

Defining Joins

In the syntax:

table1.column denotes the table and column from which data is retrieved

NATURAL JOIN joins two tables based on the same column name

JOIN *table* USING *column_name* performs an equijoin based on the column name

JOIN *table ON table1.column_name* performs an equijoin based on the condition in the ON clause, = *table2.column_name*

LEFT/RIGHT/FULL OUTER is used to perform outer joins

CROSS JOIN returns a Cartesian product from the two tables

For more information, see "SELECT" in *Oracle SQL Reference*.

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.

Creating Natural Joins

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

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, 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 ;
```

	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

ORACLE

Retrieving Records with Natural Joins

In the example in the slide, the `LOCATIONS` table is joined to the `DEPARTMENT` table by the `LOCATION_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, natural join can be applied by 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.
- Do not use a table name or alias in the referenced columns.
- The NATURAL JOIN and USING clauses are mutually exclusive.

ORACLE

USING Clause

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. 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 following 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 also applies to NATURAL joins. Therefore, columns that have the same name in both tables must be used without any qualifiers.

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
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)

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

Foreign key **Primary key**

ORACLE

The USING Clause for 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 the USING Clause

```
SELECT employees.employee_id, employees.last_name,
       departments.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	205	Higgins	1700	110
5	206	Gietz	1700	110
6	100	King	1700	90
7	101	Kochhar	1700	90
8	102	De Haan	1700	90
9	103	Hunold	1400	60
10	104	Ernst	1400	60

...

Retrieving Records with the USING Clause

The slide example joins the DEPARTMENT_ID column in the EMPLOYEES and DEPARTMENTS tables, and thus shows the location where an employee works.

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.
- Do not use aliases on columns that are identified in the `USING` clause and listed elsewhere in the SQL statement.

ORACLE

Qualifying Ambiguous Column Names

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:

```
SELECT employees.employee_id, employees.last_name,
       departments.department_id, departments.location_id
FROM   employees JOIN departments
ON     employees.department_id = departments.department_id;
```

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.

Note: 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.

Using Table Aliases

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

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

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

ON Clause

Use the ON clause to specify a join condition. This lets you 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	205	Higgins	110	110	1700
5	206	Gietz	110	110	1700
6	100	King	90	90	1700
7	101	Kochhar	90	90	1700
8	102	De Haan	90	90	1700
9	103	Hunold	60	60	1400
10	104	Ernst	60	60	1400

...

ORACLE

Creating Joins with the ON Clause

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.

You can also use the ON clause to join columns that have different names.

Self-Joins Using the ON Clause

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, or perform a self join. For example, to find the name of Lorentz's manager, you need to:

- 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 with 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 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 e.last_name emp, m.last_name mgr
FROM   employees e JOIN employees m
ON    (e.manager_id = m.employee_id);
```

	EMP	MGR
1	Abel	Zlotkey
2	Davies	Mourgos
3	De Haan	King
4	Ernst	Hunold
5	Fay	Hartstein
6	Gietz	Higgins
7	Grant	Zlotkey
8	Hartstein	King

...

ORACLE

Joining a Table to Itself (continued)

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.

Applying Additional Conditions to a Join

```
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 ;
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	174	Abel	80	80	2500
2	176	Taylor	80	80	2500

ORACLE

Applying Additional Conditions to a Join

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:

```
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;
```

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

...

ORACLE

Three-Way Joins

A three-way join is a join of three tables. In SQL:1999–compliant syntax, joins are performed from left to right. So, 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.

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.

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 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

...

ORACLE

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 EMPLOYEES and DEPARTMENTS tables, department ID 190 does not appear because there are no employees with that department ID recorded in the EMPLOYEES table. Instead of seeing 20 employees in the result set, you see 19 records.

To return the department record that does not have any employees, 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) tables 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

INNER Versus OUTER Joins

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	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Higgins	110	Accounting

...

18	Abel	80	Sales
19	Taylor	80	Sales
20	Grant	(null)	(null)

ORACLE

Example of LEFT OUTER JOIN

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

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

...

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

ORACLE

Example of RIGHT OUTER JOIN

This query retrieves all rows in the DEPARTMENTS table, which is the table on 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	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
...			
18	Abel	80	Sales
19	Taylor	80	Sales
20	Grant	(null)	(null)
21	(null)	190	Contracting

Example of FULL OUTER JOIN

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.

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.

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 shows 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.

Creating Cross Joins

- The CROSS JOIN clause produces the cross-product of two tables.
- This is also called a Cartesian product between the two tables.

```
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

...

159	Whalen	Contracting
160	Zlotkey	Contracting

Creating Cross Joins

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

Summary

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

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

Summary

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 a 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.
- Table aliases can help to keep SQL code smaller by conserving memory.

Practice 5: Overview

This practice covers the following topics:

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

Practice 5: Overview

This practice is intended to give you practical experience in extracting data from multiple tables using the SQL:1999–compliant joins.

Practice 5

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. Use a `NATURAL JOIN` to produce the results.

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

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

Practice 5 (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 last name and employee number of employees 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 `lab_05_04.sql`.

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

Practice 5 (continued)

5. Modify `lab_05_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_05_05.sql`. Run the query in `lab_05_05.sql`.

	Employee	EMP#	Manager	Mgr#
1	Whalen	200	Kochhar	101
2	Hartstein	201	King	100
3	Fay	202	Hartstein	201
4	Higgins	205	Kochhar	101
5	Gietz	206	Higgins	205
6	King	100	(null)	(null)
7	Kochhar	101	King	100
8	De Haan	102	King	100
9	Hunold	103	De Haan	102
10	Ernst	104	Hunold	103

▪ ▪ ▪

20	Grant	178	Zlotkey	149

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_05_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
6	50	Davies	Vargas
7	50	Matos	Davies
8	50	Matos	Mourgos
9	50	Matos	Rajs
10	50	Matos	Vargas
11	50	Mourgos	Davies
12	50	Mourgos	Matos
13	50	Mourgos	Rajs
14	50	Mourgos	Vargas

▪ ▪ ▪

42	110	Higgins	Gietz

Practice 5 (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. Then create a query that displays the name, job, department name, salary, and grade for all employees.

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

3 rows selected
```

	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 5 (continued)

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

8. The HR department wants to determine the names of all employees who were 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 names and hire dates for all employees who were hired before their managers, along with their managers' names and hire dates. Save the script to a file named `lab5_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	Rajs	17-OCT-95	Mourgos	16-NOV-99
4	Davies	29-JAN-97	Mourgos	16-NOV-99
5	Matos	15-MAR-98	Mourgos	16-NOV-99
6	Vargas	09-JUL-98	Mourgos	16-NOV-99
7	Abel	11-MAY-96	Zlotkey	29-JAN-00
8	Taylor	24-MAR-98	Zlotkey	29-JAN-00
9	Grant	24-MAY-99	Zlotkey	29-JAN-00