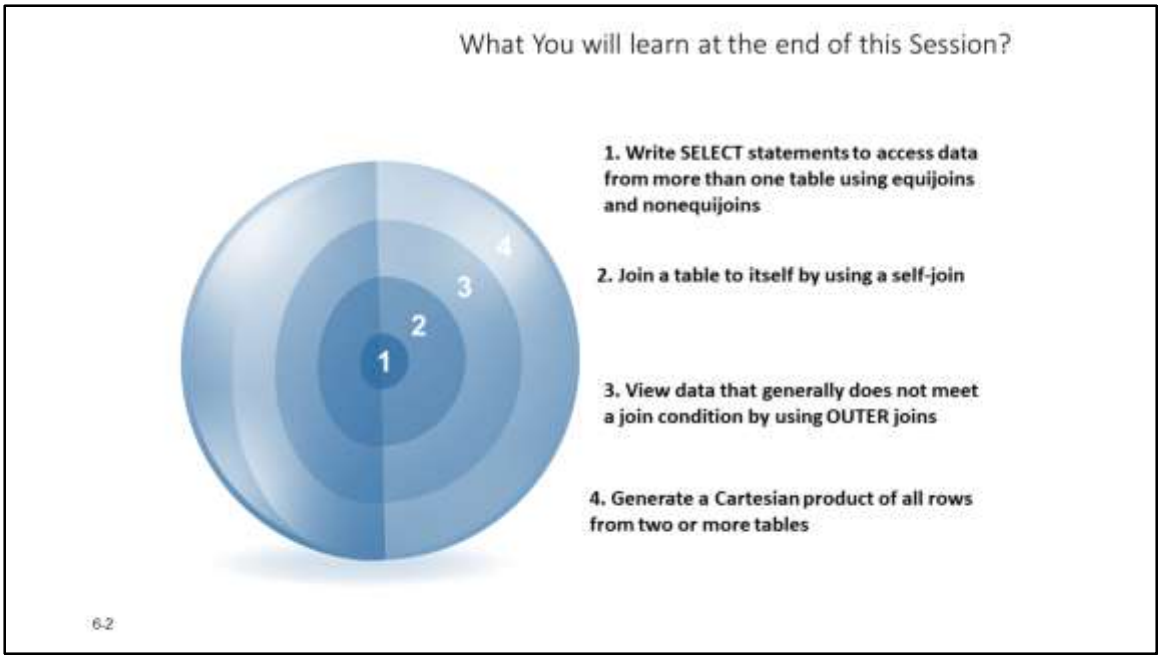
Lesson 6

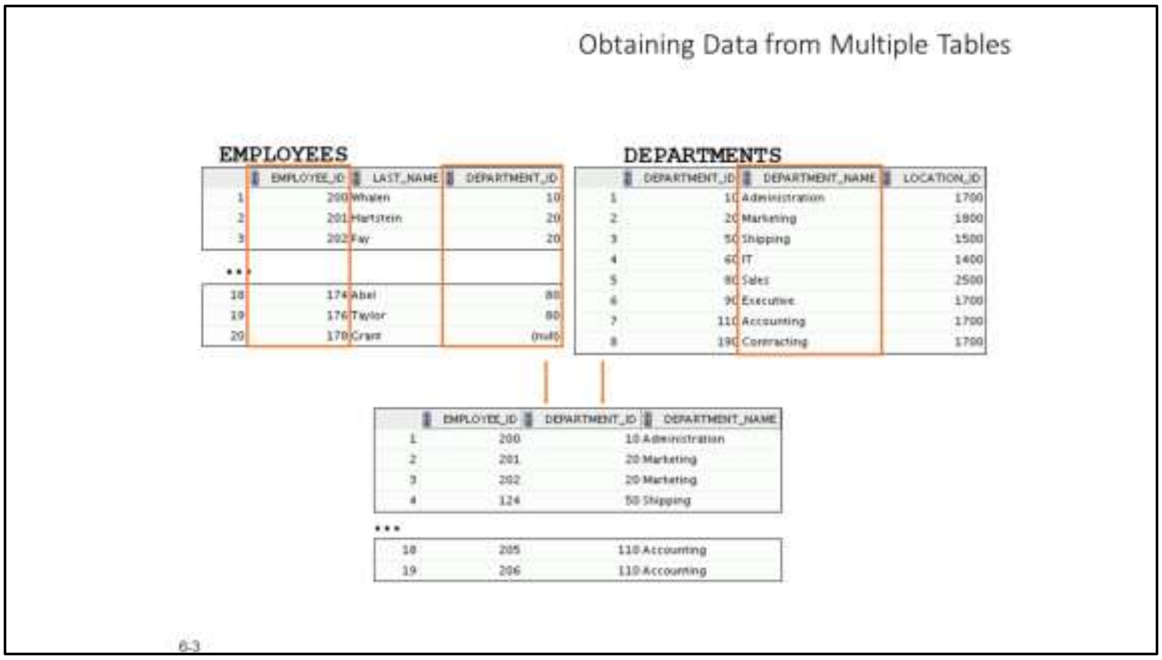
# Displaying Data
from Multiple Tables Using Joins



What You will learn at the end of this Session?

1. Write SELECT statements to access data from more than one table using equijoins and nonequijoins
2. Join a table to itself by using a self-join
3. View data that generally does not meet a join condition by using OUTER joins
4. Generate a Cartesian product of all rows from two or more tables

What You will learn at the end of this Session?
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 10*g* or 11*g* database.



Obtaining Data from Multiple Tables

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
...			
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	200	10	Administration
2	201	20	Marketing
3	202	20	Marketing
4	124	50	Shipping
...			
18	205	110	Accounting
19	206	110	Accounting

6-3

Obtaining Data from Multiple Tables

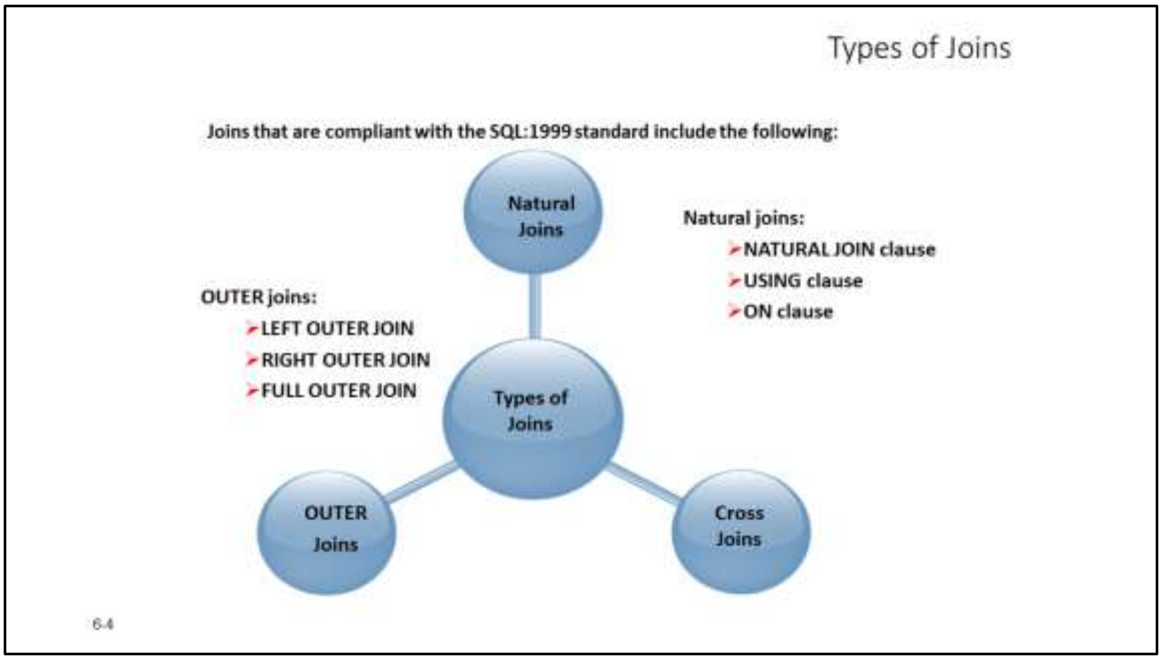
Sometimes you need to use data from more than one table. In the example in the slide, the report displays data from two separate tables:

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:

Natural joins:
- NATURAL JOIN clause
- USING clause
- ON clause

OUTER joins:
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

Natural Joins
Types of Joins
OUTER Joins
Cross Joins
6-4

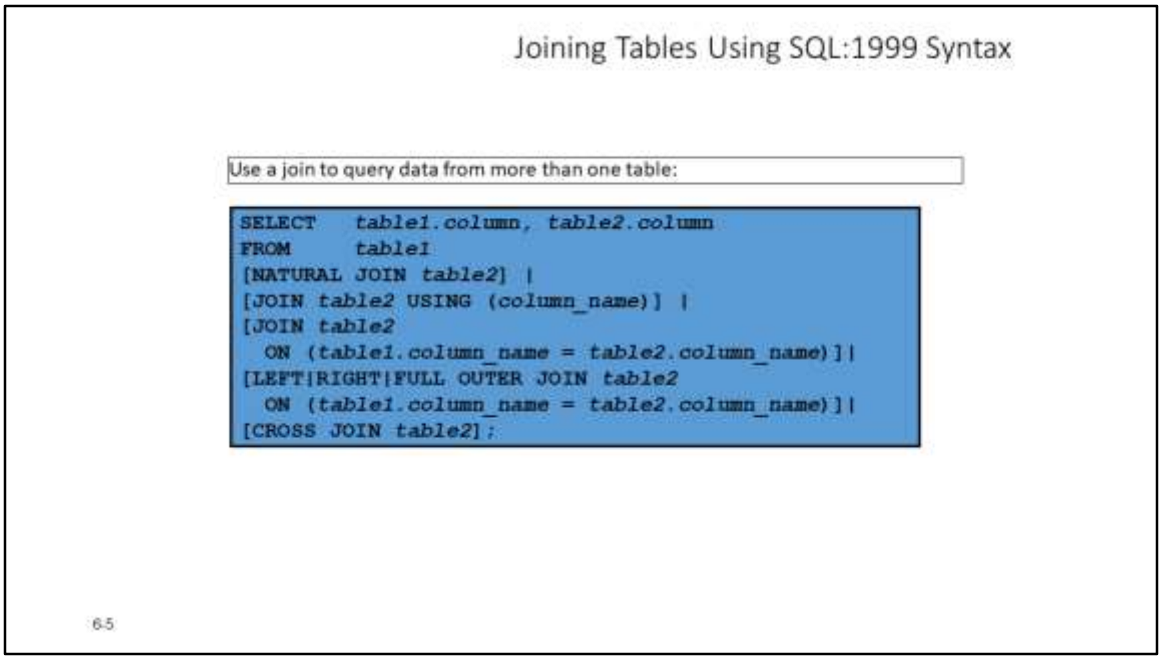
Types of Joins

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

Note

Before the Oracle9*i* release, the 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. For detailed information about the proprietary join syntax, see Appendix F: Oracle Join Syntax.

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];
```

6-5

Joining Tables Using SQL:1999 Syntax

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

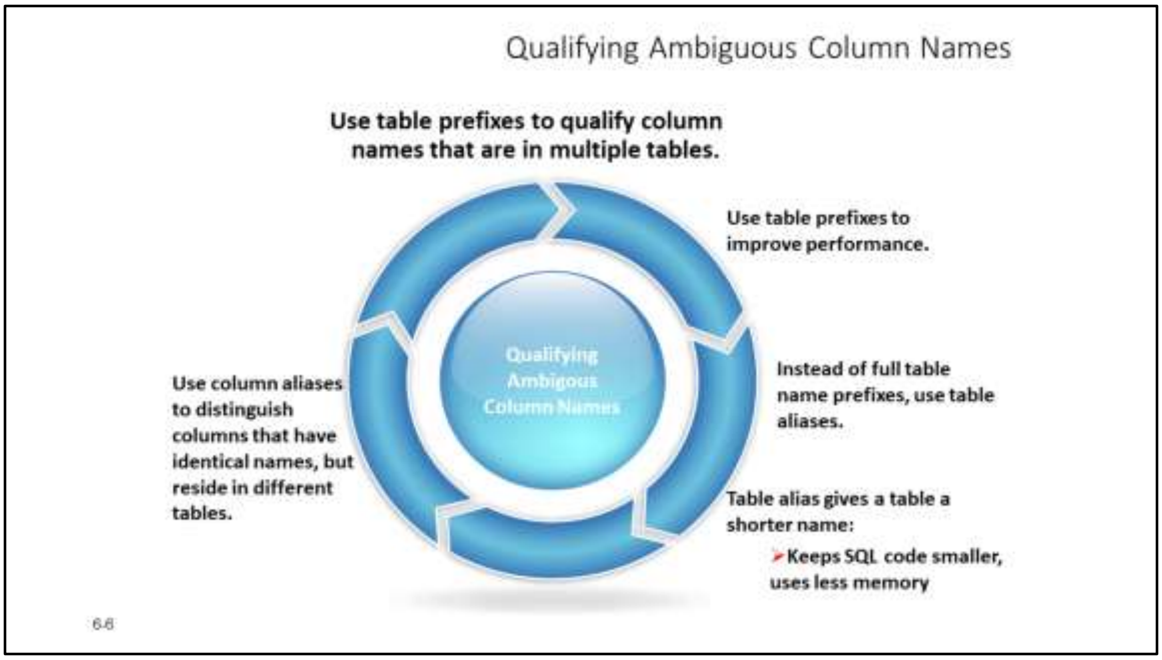
`JOIN table2 USING column_name` performs an equijoin based on the column name

`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 10*g* or 11*g* database.



Qualifying Ambiguous Column Names

Use table prefixes to qualify column names that are in multiple tables.

Use table prefixes to improve performance.

Instead of full table name prefixes, use table aliases.

Table alias gives a table a shorter name:

- Keeps SQL code smaller, uses less memory

Use column aliases to distinguish columns that have identical names, but reside in different tables.

Qualifying Ambiguous Column Names

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 improves performance, 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`.

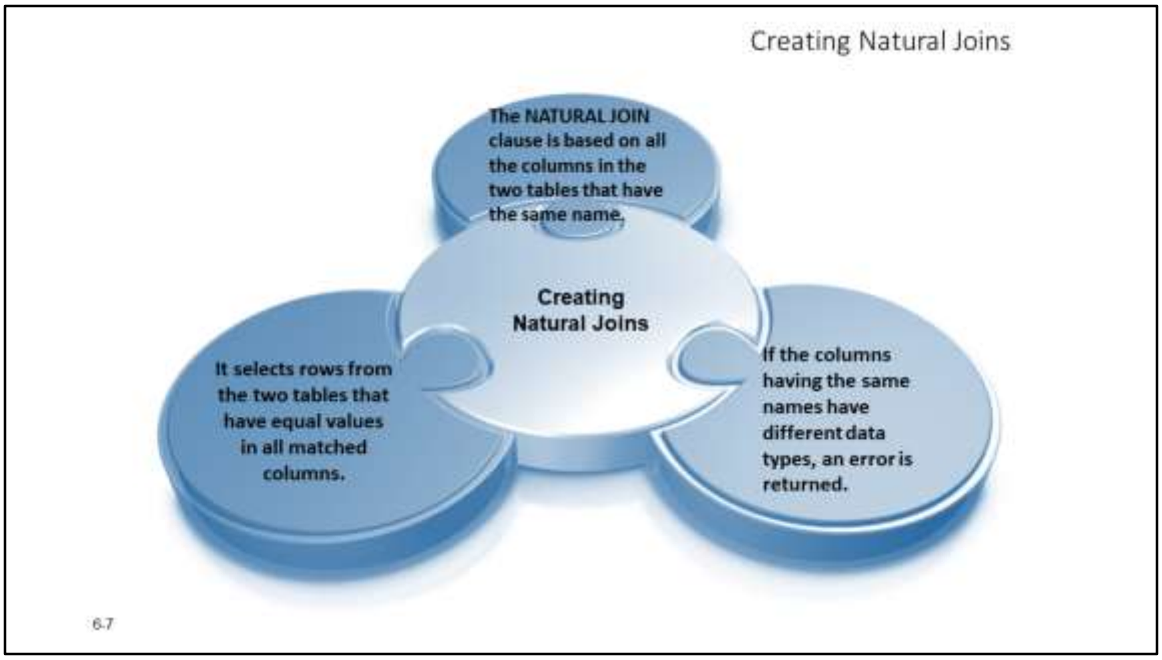
Guidelines

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.



Creating Natural Joins

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

Creating Natural Joins

It selects rows from the two tables that have equal values in all matched columns.

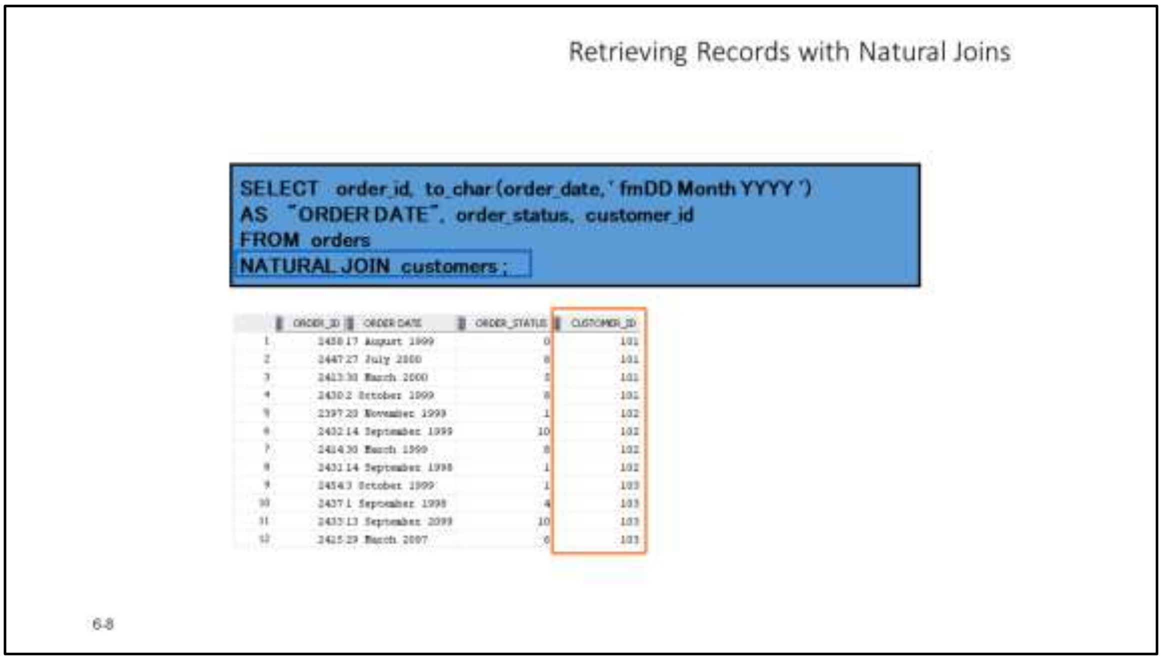
If the columns having the same names have different data types, an error is returned.

6-7

Creating Natural Joins

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  order_id, to_char(order_date,' fmDD Month YYYY ')
AS  "ORDER DATE", order_status, customer_id
FROM  orders
NATURAL JOIN  customers ;
```

	ORDER_ID	ORDER DATE	ORDER_STATUS	CUSTOMER_ID
1	2458	17 August 1999	0	101
2	2447	27 July 2000	8	101
3	2413	30 March 2000	5	101
4	2430	2 October 1999	8	101
5	2397	20 November 1999	1	102
6	2432	14 September 1999	10	102
7	2414	30 March 1999	8	102
8	2431	14 September 1998	1	102
9	2454	3 October 1999	1	103
10	2437	1 September 1998	4	103
11	2433	13 September 1999	10	103
12	2425	29 March 2007	6	103

6-8

Retrieving Records with Natural Joins

In the example in the slide, the `CUSTOMERS` table is joined to the ORDERS table by the CUSTOMER_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 an Order status equal to 0 or 1:

```
SELECT  order_id, order_date,
        order_status, customer_id
FROM    orders
NATURAL JOIN customers
WHERE   order_status IN (0, 1);
```