VI.I Multiple Table Part I

Lesson Objectives (Part 1 & 2)

After completing this lesson, you should be able to do the following:

- Write SELECT statement to access data from more than one table using equijoins and nonequijoins
- 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

Lesson Agenda

- Type of JOINS and its syntax
- INNER joins:
  - NATURAL JOIN clause
  - USING clause
  - ON clause

Lesson Agenda

- **Type of JOINS and its syntax**
- INNER joins:
  - NATURAL JOIN clause
  - USING clause
  - ON clause

Obtaining Data from Multiple Tables

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
...			
18	202	Fay	20
19	205	Higgins	110
20	206	Gietz	110

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
5	144	50	Shipping
...			
18	205	110	Accounting
19	206	110	Accounting

Types of Joins

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

- `INNER` joins:
  - `NATURAL JOIN` clause
  - `USING` clause
  - `ON` clause
- `OUTER` joins:
  - `LEFT OUTER JOIN`
  - `RIGHT OUTER JOIN`
  - `FULL OUTER JOIN`

Joining Tables Using SQL Syntax

```
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)];
```


Joining Tables Using SQL 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 JOIN` is used to perform `OUTER` join

Lesson Agenda

- Type of JOINS and its syntax
- **INNER joins:**
  - **NATURAL JOIN clause**
  - **USING clause**
  - **ON clause**

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.

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 NATURAL JOIN the syntax causes an error.

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.

```
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	30	Purchasing	1700	Seattle
5	90	Executive	1700	Seattle
6	100	Finance	1700	Seattle

Retrieving Records with Natural Joins

PRACTICE:

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.

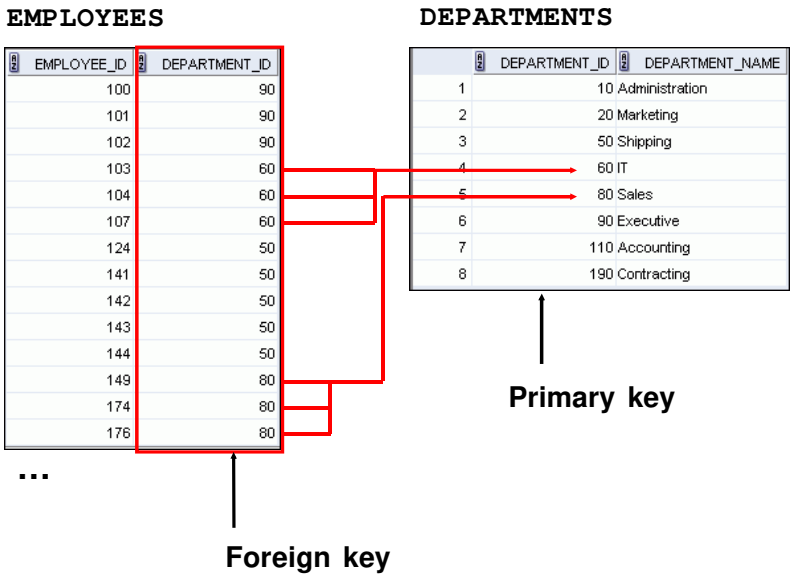
	LOCATION_ID	STREET_ADDRESS	CITY	STATE_PROVINCE	COUNTRY_NAME
1	2200	12-98 Victoria Street	Sydney	New South Wales	Australia
2	2800	Rua Frei Caneca 1360	Sao Paulo	Sao Paulo	Brazil
3	1800	147 Spadina Ave	Toronto	Ontario	Canada
4	1900	6092 Boxwood St	Whitehorse	Yukon	Canada

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 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.
- The NATURAL JOIN and USING clauses are mutually exclusive.

Note: 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



Retrieving Records with the USING Clause

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.

```
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	100	King	1700	90
2	101	Kochhar	1700	90
3	102	De Haan	1700	90
4	103	Hunold	1400	60

Retrieving Records with the USING Clause

PRACTICE:

Create a query to display the last name, department number, and department name for all the employees.

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	King	90	Executive
2	Kochhar	90	Executive
3	De Haan	90	Executive
4	Hunold	60	IT
5	Ernst	60	IT

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

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.

Using Table Aliases with the USING Clause

- Do not qualify a column that is used in the 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: 8
```


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.

Note: 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.

Joining Tables Using Oracle Syntax

- Use a join to query data from more than one table.
- 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.

```
SELECT   t1.column, t2.column
FROM     table1 t1 JOIN table2 t2
ON       t1.column1 = t2.column2;
```

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

Retrieving Records 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.

```
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	100	King	90	90	1700
2	101	Kochhar	90	90	1700
3	102	De Haan	90	90	1700
4	103	Hunold	60	60	1400
5	104	Ernst	60	60	1400

Creating Three-Way Joins with the ON Clause

- 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.
- **Note:** The code example in the slide can also be accomplished with the USING clause:

Creating Three-Way Joins with the ON Clause

```
SELECT   employee_id, city, department_name
FROM     employees e JOIN departments d
ON       e.department_id = d.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	105	Southlake	IT
7	106	Southlake	IT
8	107	Southlake	IT
9	108	Seattle	Finance
10	109	Seattle	Finance

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;
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	174	Abel	80	80	2500
2	175	Hutton	80	80	2500
3	176	Taylor	80	80	2500
4	177	Livingston	80	80	2500
5	179	Johnson	80	80	2500

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
WHERE     e.manager_id = 149;
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	174	Abel	80	80	2500
2	175	Hutton	80	80	2500
3	176	Taylor	80	80	2500
4	177	Livingston	80	80	2500
5	179	Johnson	80	80	2500

Applying Additional Conditions to a Join

PRACTICE:

Additional restrictions on a 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:

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
1	20	Marketing	1800	Toronto
2	50	Shipping	1500	South San Francisco

Summary

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

- Equijoins
- INNER joins

Table Aliases:

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