Displaying Data from Multiple Tables

Chapter 4

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 equality and nonequality joins**
- **View data that generally does not meet a join condition by using outer joins**
- **Join a table to itself**

Lesson Aim

This lesson covers how to obtain data from more than one table, using the different methods available.

Cartesian Product

- 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.

Generating a Cartesian Product

```
SELECT ename, dname
FROM emp, dept;
```

ENAME	DNAME
BLAKE	ACCOUNTING
SMITH	ACCOUNTING
ALLEN	ACCOUNTING

56 Rows Selected

What Is a Join?

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

Old Syntax

Write the join condition in the WHERE clause.

```
SELECT    tablel.column,  table2. column2
FROM      table1,  table2
WHERE     tablel. columnl  =  table2. column2;
```

ANSI Syntax

Write the join condition in the ON clause.

```
SELECT    tablel.column,  table2. column2
FROM      table1 INNER JOIN  table2
ON     tablel. columnl  =  table2. column2;
```

Prefix the column name with the table name when the same column name appears in more than one table.

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 existing 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.column1 denotes the table and column from which data is retrieved

Table1. column1 = *table2. column2* is the condition that joins (or relates) the tables together.

Types of Joins

- **Equijoin**
- **Non-equijoin**
- **Outer join**
- **Self join**

Types of Joins

There are two main types of join conditions:

- Equijoins
- Non-equijoins

Additional join methods include the following

- Outerjoins
- Selfjoins
- Set Operators

Note: Set operators are not covered in this course . They are covered in another SQL course.

What Is an Equijoin?

Equijoins
To determine the name of an employee's department, you compare the value in the DEPTNO column in the EMP table with the DEPTNO values in the DEPT table.

The relationship between the EMP and DEPT table is an equijoin - that is, values in the DEPTNO column on both tables must be equal.
Frequently, this type of join involves primary and foreign key complements.

Note: Equijoins are also called simple joins or innerjoins.

Obtaining Data from Multiple Tables

```
SELECT e.empno, e.deptno, d.loc
FROM emp e, dept d
WHERE e.deptno = d.deptno;
```

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.

- EMPNO exists in the EMP table
- DEPTNO exists in both the EMP and DEPT the Tables.
- LOC exists in the DEPT table.

To prodıce the report. you need to link EMP and DEPT tables and access data from both of them.

ANSI Syntax

```
SELECT e.empno, e.deptno, d.loc
FROM emp e inner join dept d
on e.deptno = d.deptno;
```

Retrieving Records with Equijoins

```
SELECT EMP.EMPNO, EMP.ENAME, EMP.DEPTNO,
DEPT.DEPTNO, DEPT.LOC
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
```

EMPNO	ENAME	DEPTNO	DEPTNO	LOC
7698	BLAKE	30	30	CHICAGO
7369	SMITH	20	20	DALLAS
7499	ALLEN	30	30	CHICAGO

14 rows selected.

Retrieving Records with Equijoins

in the slide examaple.

- The SELECT clause specifies the column names to retrieve:
  - employee name, employee number, and department number, which are columns in the emp table
  - department number, department name, and location, which are columns in the DEPT table.

The FROM clause specifies the two tables that the database must access:

EMP table

DEPT table

The WHERE clause specifies how the tables are to be joined:

EMP.DEPTNO=DEPT.DEPTNO

Oualifying Ambiguous Column Names

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

Improve performance by using table prefixes.

Distinguish columns that have identical names but reside in different tables by using column aliases.

Qualifying Ambiguous Column Names

You need to gualify the names of the columns in the WHERE clause ""itli the table names to avoid ambiguity without the table prefixes. the DEPTNO column could be from either the DEPT table or the EMP 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. Howevwr, you will gain improved performance by using the table prefix because you tell the Oracle Server exactly where to find the columns.

EQUIJOIN

```
SELECT emp.empno, emp.ename, emp.deptno,
dept.deptno, dept.loc
FROM emp, dept
WHERE  emp.deptno = Dept.deptno;
```

EMPNO	ENAME	DEPTNO	DEPTNO	LOC
7698	BLAKE	30	30	CHICAGO
7369	SMITH	20	20	DALLAS

14 rows selected.

Using Table Aliases

The following two scripts are equivalent. In the second one table aliases are used.

```
SELECT e.ename, e.deptno, d.dname

FROM emp e , dept d

WHERE e.deptno = d.deptno ;
```

ENAME	DEPTNO	DNAME
BLAKE	30	SALES
SMITH	20	RESEARCH

14 rows selected.

Additional Search Conditions
Using the AND Operator

Additional Search Conditions

In addition to the join, you may have criteria for your WHERE clause. For example, to display King's employee number, name, department number, and departments localion, you need an additional condition in the WHERE clause.

```
SELECT EMP.EMPNO, EMP.ENAME, EMP.DEPTNO,
DEPT.DEPTNO, DEPT.LOC
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
AND   INITCAP(ename) = 'King' ;
```

EMPNO	ENAME	DEPTNO	DEPTNO	LOC
7839	KING	10	10	NEW YORK

ANSI Ssyntax

ORACLE'da Inner Join 'i yapan

FROM emp e, dept d WHERE e.deptno = d.deptno

yerine

FROM emp e INNER JOIN dept d

inner Join koşulu yazıldıktan sonra iki tablo arasında bağlantıyı kuran

ON e.deptno = d.deptno

koşulu yazılır.

Eğer, ayrıca satırlardan yeni süzme yapılacaksa WHERE ile istenen koşul konulabilir:

WHERE INITCAP(ename) = 'King' ;

Biçimindeki koşul yazılabilir.

Örnek

```
SELECT e.empno, e.ename, e.deptno, d.deptno, d.loc
FROM emp e INNER JOIN dept d
ON e.deptno = d.deptno
WHERE  INITCAP(ename) = 'King' ;
```

EMPNO	ENAME	DEPTNO	DEPTNO	LOC
7839	KING	10	10	NEW YORK

Non-Equijoins

The relationship between the EMP table and the SALGRADE table is a non-equijoin, meaning that no column in the EMP table corresponds directly to a column in the SALGRADE table.

The relationship between the two tables is that the SAL column in the EMP table is between the LOSAL and HISAL column of the SALGRADE table.

The relationship is obtained using an operator other than equal (=).

```
SELECT      *
FROM emp ;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7698	BLAKE	MANAGER	7839	01/05/1981	2850		30

14 rows selected.

```
SELECT      *
FROM salgrade ;
```

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

5 rows selected.

Retreive records where Salary in the EMP table is between low salary and high salary in the SALGRADE table.

```
SELECT e.ename, e.sal, s.grade

FROM EMP e, SALGRADE s

WHERE e.sal

BETWEEN s.losal AND s.hisal ;
```

ENAME	SAL	GRADE
SMITH	800	1
JAMES	950	1

14 rows selected.

ANSI Syntax

```
SELECT e.ename, e.sal, s.grade
FROM EMP e INNER JOIN SALGRADE s
ON  e.sal
BETWEEN s.losal AND s.hisal ;
```

ENAME	SAL	GRADE
SMITH	800	1
JAMES	950	1

14 rows selected.

Joining More Than Two Tables

```
SELECT e.ename, e.deptno, d.dname, s.grade
FROM emp e, dept d, salgrade s
WHERE e.deptno = d.deptno AND
e.sal BETWEEN  s.losal AND hisal;
```

ENAME	DEPTNO	DNAME	GRADE
KING	10	ACCOUNTING	5
CLARK	10	ACCOUNTING	4
MILLER	10	ACCOUNTING	2
FORD	20	RESEARCH	4
SCOTT	20	RESEARCH	4
JONES	20	RESEARCH	4
ADAMS	20	RESEARCH	1
SMITH	20	RESEARCH	1
BLAKE	30	SALES	4
ALLEN	30	SALES	3
TURNER	30	SALES	3
MARTIN	30	SALES	2
WARD	30	SALES	2
JAMES	30	SALES	1

14 rows selected.

ANSI Syntax

```
SELECT e.ename, e.deptno, d.dname, s.grade
FROM salgrade s, emp e INNER JOIN dept d
ON e.deptno = d.deptno
WHERE
e.sal BETWEEN  s.losal AND s.hisal;
```

ENAME	DEPTNO	DNAME	GRADE
KING	10	ACCOUNTING	5
CLARK	10	ACCOUNTING	4
MILLER	10	ACCOUNTING	2

14 rows selected.

Retrieving Records with Non-Equijoins

```
SELECT e.ename, e.sal, s.grade
FROM EMP e, SALGRADE s
WHERE e.sal +e.comm > s.hisal
```

ENAME	SAL	GRADE
TURNER	1500	1
WARD	1250	1
ALLEN	1600	1
MARTIN	1250	1
TURNER	1500	2
WARD	1250	2
ALLEN	1600	2
MARTIN	1250	2
MARTIN	1250	3

9 rows selected.

Non-Equijoins (continued)

The slide example creates a non-equijoin 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 salary grade table contain grades that overlap. That is, the salary value for an employee can only lie 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 salary grade table. That is, no employee earns less than the lowest value contained in the LOSAL column or more than the highest value contained in the HISAL column.

Note: Other operators such as <= and >= could 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 for performance reasons, not because of possible ambiguity.

ANSI Syntax (Non Equijoin)

```
SELECT e.ename, e.sal, s.grade
FROM EMP e INNER JOIN SALGRADE s
ON e.sal +e.comm > s.hisal ;
```

ENAME	SAL	GRADE
ALLEN	1600	1
WARD	1250	1
MARTIN	1250	1
TURNER	1500	1
ALLEN	1600	2
WARD	1250	2
MARTIN	1250	2
TURNER	1500	2
MARTIN	1250	3

9 rows selected.

Outer joins

When two tables are joined with an inner join, data will only be returned if matching data exists in both tables. An outer join is like saying "and also include the rows from one table if there are no matching rows in the other one."

With an outer join the columns from the table where data is "missing" are returned as NULL values.

Outer joins come in two basic flavours, called Left and Right. Left outer joins mean that the data must be contained in the table defined to the left side of the equivalence, but not necessarily the right hand side. Right outer joins, of course, work the other way around.

To illustrate this, cut and paste the code below into a Query Analyser window and try running it. I have used the newer ANSI syntax here, and the older equivalents are included but commented out using the "--" comment notation. Comment them back in if you want to try them.

Outer Joins

Returning Records with No Direct Match with Outer Joins

If a row does not satisfy a join condition, the row will not appear in the query result. For example, in the equijoin condition of EMP and DEPT tables, department OPERATIONS does not appear because no one works in that department.

```
SELECT e.ename , e.deptno, d.dname
FROM  emp e, dept d
WHERE      e.deptno   =  d.deptno;
```

ENAME	DEPTNO	DNAME
BLAKE	30	SALES
SMITH	20	RESEARCH
ALLEN	30	SALES
WARD	30	SALES
JONES	20	RESEARCH
MARTIN	30	SALES
CLARK	10	ACCOUNTING
SCOTT	20	RESEARCH
KING	10	ACCOUNTING
TURNER	30	SALES
ADAMS	20	RESEARCH
JAMES	30	SALES
FORD	20	RESEARCH
MILLER	10	ACCOUNTING

14 rows selected.

No employee in the OPERATIONS department

Outer Joins

Returning Records with No Direct Match with Outer Joins

If a row does not satisfy a join condition, the row will not appear in the query result. For example, in the equijoin condition of EMP and DEPT tables, department OPERATIONS does not appear because no one works in that department.

Outer Joins (Old usage)

You use an outer join to also see rows that do not usually meet the join condition.
Outer join operator is the plus sign (+).

```
SELECT   tablel.column,   table2. column
FROM     tablel,   table2
WHERE    tablel.column(+)  =  table2.column;
```

```
SELECT   tablel.column,   table2.column
FROM     tablel,   table2
WHERE    tablel.column. =   table2. column {+) ;
```

Returning Records with No Direct Match with Outer Joins
The missing row(s) can be returned if an outerjoin operator is used in the join condition. The operator is a plus sign enclosed in parenthesis (+), and it is placed on the "side" of the equality that the join rhctt a deficient in mfürmcınun. This operator has the effect of creating one or more mil l rows, to which one or more rows from the nondeficient table can be joined in the syntax.

In the condition that joins (or relates) the lables together, 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.)

Old Usage

```
SELECT e.ename, e.deptno, d.dname
FROM   emp e, dept d
WHERE     e.deptno(+)  =  d.deptno;
```

ENAME	DEPTNO	DNAME
SMITH	20	RESEARCH
ALLEN	30	SALES
WARD	30	SALES
JONES	20	RESEARCH
MARTIN	30	SALES
BLAKE	30	SALES
CLARK	10	ACCOUNTING
SCOTT	20	RESEARCH
KING	10	ACCOUNTING
TURNER	30	SALES
ADAMS	20	RESEARCH
JAMES	30	SALES
FORD	20	RESEARCH
MILLER	10	ACCOUNTING
		OPERATIONS

15 rows selected.

ANSI Right Outer Join Syntax

```
SELECT e.ename, e.deptno, d.dname
FROM   emp e RIGHT OUTER JOIN dept d
ON     e.deptno  =  d.deptno;
```

ENAME	DEPTNO	DNAME
SMITH	20	RESEARCH
ALLEN	30	SALES
WARD	30	SALES
JONES	20	RESEARCH
MARTIN	30	SALES
BLAKE	30	SALES
CLARK	10	ACCOUNTING
SCOTT	20	RESEARCH
KING	10	ACCOUNTING
TURNER	30	SALES
ADAMS	20	RESEARCH
JAMES	30	SALES
FORD	20	RESEARCH
MILLER	10	ACCOUNTING
		OPERATIONS

15 rows selected.

ANSI Left Outer Join Syntax

```
SELECT e.ename, e.deptno, d.dname
FROM   emp e RIGHT OUTER JOIN dept d
ON     e.deptno  =  d.deptno;
```

ENAME	DEPTNO	DNAME
SMITH	20	RESEARCH
ALLEN	30	SALES
WARD	30	SALES
JONES	20	RESEARCH
MARTIN	30	SALES
BLAKE	30	SALES
CLARK	10	ACCOUNTING
SCOTT	20	RESEARCH
KING	10	ACCOUNTING
TURNER	30	SALES
ADAMS	20	RESEARCH
JAMES	30	SALES
FORD	20	RESEARCH
MILLER	10	ACCOUNTING

14 rows selected.

Old usage

```
SELECT e.ename, e.deptno, d.dname
FROM   emp e, dept d
WHERE     e.deptno   =  d.deptno(+);
```

ENAME	DEPTNO	DNAME
SMITH	20	RESEARCH
ALLEN	30	SALES
WARD	30	SALES
JONES	20	RESEARCH
MARTIN	30	SALES
BLAKE	30	SALES
CLARK	10	ACCOUNTING
SCOTT	20	RESEARCH
KING	10	ACCOUNTING
TURNER	30	SALES
ADAMS	20	RESEARCH
JAMES	30	SALES
FORD	20	RESEARCH
MILLER	10	ACCOUNTING

14 rows selected.

OUTER JOIN

Previously, we had looked at left join, or inner join, where we select rows common to the participating tables to a join. What about the cases where we are interested in selecting elements in a table regardless of whether they are present in the second table? We will now need to use the **SQL OUTER JOIN** command.

The syntax for performing an outer join in SQL is database-dependent. For example, in Oracle, we will place an "(+)" in the **WHERE** clause on the other side of the table for which we want to include all the rows.

Let's assume that we have the following two tables,

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

Table ***Geography***

region_name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

and we want to find out the sales amount for all of the stores. If we do a regular join, we will not be able to get what we want because we will have missed "New York," since it does not appear in the ***Store_Information*** table. Therefore, we need to perform an outer join on the two tables above:

OUTER JOIN

```
SELECT A1.store_name, SUM(A2.Sales) SALES
FROM Geography A1, Store_Information A2
WHERE A1.store_name = A2.store_name (+)
GROUP BY A1.store_name
```

Note that in this case, we are using the Oracle syntax for outer join.

Result:

store_name	SALES
Boston	$700
New York	
Los Angeles	$1800
San Diego	$250

Note: NULL is returned when there is no match on the second table. In this case, "New York" does not appear in the table ***Store_Information***, thus its corresponding "SALES" column is NULL.

ORDER BY usage in OUTER JOIN

```
SELECT e.ename, d.DEPTNO, d.dname
FROM emp e, dept d
WHERE e.deptno(+) = d.deptno
ORDER BY e.deptno;
```

ENAME	DEPTNO	DNAME
ALLEN	30	SALES
WARD	30	SALES
	40	OPERATIONS

15 rows selected.

ANSI Syntax for OUTER JOIN

```
SELECT e.ename, d.DEPTNO, d.dname
FROM emp e RIGHT OUTER JOIN dept d
ON e.deptno = d.deptno
ORDER BY e.deptno;
```

ENAME	DEPTNO	DNAME
CLARK	10	ACCOUNTING
MILLER	10	ACCOUNTING
KING	10	ACCOUNTING
JONES	20	RESEARCH
SMITH	20	RESEARCH
SCOTT	20	RESEARCH
FORD	20	RESEARCH
ADAMS	20	RESEARCH
WARD	30	SALES
TURNER	30	SALES
ALLEN	30	SALES
JAMES	30	SALES
MARTIN	30	SALES
BLAKE	30	SALES
	40	OPERATIONS

15 rows selected.

Using Outer Joins

```
SELECT e.ename, d.DEPTNO, d.dname
FROM emp e, dept d
WHERE e.deptno(+) = d.deptno
ORDER BY e.deptno;
```

ENAME	DEPTNO	DNAME
MILLER	10	ACCOUNTING
KING	10	ACCOUNTING
CLARK	10	ACCOUNTING
SMITH	20	RESEARCH
FORD	20	RESEARCH
ADAMS	20	RESEARCH
SCOTT	20	RESEARCH
JONES	20	RESEARCH
TURNER	30	SALES
JAMES	30	SALES
ALLEN	30	SALES
MARTIN	30	SALES
BLAKE	30	SALES
WARD	30	SALES
	40	OPERATIONS

15 rows selected.

Joining a Table to Itself

SQL JOIN

Now we want to look at joins. To do joins correctly in SQL requires many of the elements we have introduced so far. Let's assume that we have the following two tables,

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

Table ***Geography***

region_name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

and we want to find out sales by region. We see that table ***Geography*** includes information on regions and stores, and table ***Store_Information*** contains sales information for each store. To get the sales information by region, we have to combine the information from the two tables. Examining the two tables, we find that they are linked via the common field, "store_name". We will first present the SQL statement and explain the use of each segment later:

Joining a Table to Itself

```
SELECT A1.region_name REGION, SUM(A2.Sales) SALES
FROM Geography A1, Store_Information A2
WHERE A1.store_name = A2.store_name
GROUP BY A1.region_name
```

Result:

REGION	SALES
East	$700
West	$2050

The first two lines tell SQL to select two fields, the first one is the field "region_name" from table ***Geography*** (aliased as REGION), and the second one is the sum of the field "Sales" from table ***Store_Information*** (aliased as SALES). Notice how the table aliases are used here: Geography is aliased as A1, and Store_Information is aliased as A2. Without the aliasing, the first line would become

```
SELECT Geography.region_name REGION,

SUM(Store_Information.Sales) SALES
```

which is much more cumbersome. In essence, table aliases make the entire SQL statement easier to understand, especially when multiple tables are included.

Next, we turn our attention to line 3, the **WHERE** statement. This is where the condition of the join is specified. In this case, we want to make sure that the content in "store_name" in table Geography matches that in table ***Store_Information***, and the way to do it is to set them equal. This **WHERE** statement is essential in making sure you get the correct output. Without the correct **WHERE** statement, a Cartesian Join will result. Cartesian joins will result in the query returning every possible combination of the two (or whatever the number of tables in the **FROM** statement) tables. In this case, a Cartesian join would result in a total of 4 x 4 = 16 rows being returned.

Self Joins

MGR in the WORKER table is equal to EMPNO in the MANAGER table.

```
SELECT WORKER.ename, WORKER.empno ,
MANAGER.ename, MANAGER.empno
FROM emp WORKER, emp MANAGER
WHERE WORKER.mgr = MANAGER.empno ;
```

ENAME	EMPNO	ENAME	EMPNO
JAMES	7900	BLAKE	7698
TURNER	7844	BLAKE	7698
MARTIN	7654	BLAKE	7698
WARD	7521	BLAKE	7698
ALLEN	7499	BLAKE	7698
FORD	7902	JONES	7566
SCOTT	7788	JONES	7566
MILLER	7934	CLARK	7782
ADAMS	7876	SCOTT	7788
CLARK	7782	KING	7839
JONES	7566	KING	7839
BLAKE	7698	KING	7839
SMITH	7369	FORD	7902

13 rows selected.

Self Joins

Use two alias for emp:

```
SELECT e.ename, e.empno , m.ename, m.empno

FROM emp e, emp m

WHERE e.mgr = m.empno ;
```

ENAME	EMPNO	ENAME	EMPNO
JAMES	7900	BLAKE	7698
TURNER	7844	BLAKE	7698
MARTIN	7654	BLAKE	7698
WARD	7521	BLAKE	7698
ALLEN	7499	BLAKE	7698
FORD	7902	JONES	7566
SCOTT	7788	JONES	7566
MILLER	7934	CLARK	7782
ADAMS	7876	SCOTT	7788
CLARK	7782	KING	7839
JONES	7566	KING	7839
BLAKE	7698	KING	7839
SMITH	7369	FORD	7902

13 rows selected.

ANSI Syntax for SELF JOIN

```
SELECT e.ename, e.empno , m.ename, m.empno
FROM emp e INNER JOIN emp m
ON e.mgr = m.empno ;
```

ENAME	EMPNO	ENAME	EMPNO
SMITH	7369	FORD	7902
ALLEN	7499	BLAKE	7698
WARD	7521	BLAKE	7698
JONES	7566	KING	7839
MARTIN	7654	BLAKE	7698
BLAKE	7698	KING	7839
CLARK	7782	KING	7839
SCOTT	7788	JONES	7566
TURNER	7844	BLAKE	7698
ADAMS	7876	SCOTT	7788
JAMES	7900	BLAKE	7698
FORD	7902	JONES	7566
MILLER	7934	CLARK	7782

13 rows selected.

SELF JOINS

```
SELECT worker.ename || 'works for ' || manager.ename
FROM emp worker, emp manager
WHERE worker.mgr = manager.empno;
```

WORKER.ENAME||'WORKSFOR'||MANAGER.ENAME
SMITHworks for FORD
ALLENworks for BLAKE
WARDworks for BLAKE
JONESworks for KING
MARTINworks for BLAKE
BLAKEworks for KING
CLARKworks for KING
SCOTTworks for JONES
TURNERworks for BLAKE
ADAMSworks for SCOTT
JAMESworks for BLAKE
FORDworks for JONES
MILLERworks for CLARK

13 rows selected.

Joining a Table to Itself (continued)

The slide example joins the EMP table to itself. To simulate two tables in the FROM clause, there are two aliases, namely WORKER and MANAGER, for the same table EMP.

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

ANSI syntax for SELF JOINS

```
SELECT worker.ename || 'works for ' || manager.ename
FROM emp worker INNER JOIN emp manager
ON worker.mgr = manager.empno;
```

WORKER.ENAME||'WORKSFOR'||MANAGER.ENAME
SMITHworks for FORD
ALLENworks for BLAKE
WARDworks for BLAKE
JONESworks for KING
MARTINworks for BLAKE
BLAKEworks for KING
CLARKworks for KING
SCOTTworks for JONES
TURNERworks for BLAKE
ADAMSworks for SCOTT
JAMESworks for BLAKE
FORDworks for JONES
MILLERworks for CLARK

13 rows selected.

Self Joins

```
SELECT e.ename, e.empno , m.ename, m.empno
FROM emp e, emp m
WHERE e.mgr = m.empno;
```

ENAME	EMPNO	ENAME	EMPNO
JAMES	7900	BLAKE	7698
TURNER	7844	BLAKE	7698

13 rows selected.

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

- Find Blake in the EMP table by looking at the ENAME column.
- Find the manager number for Blake by looking at the MGR column. Blake's manager number is 7839.
- Find the name of the manager with EMPNO 7839 by looking at the ENAME column. King's employee number is 7839, so King is Blake's manager.

In this process, you look in the table twice. The first time you look in the table to find Blake in the ENAME column and MGR value of 7839. The second time you look in the EMPNO column to find 7839 and the ENAME column to find King

ANSI Syntax for Self Join

```
SELECT e.ename, e.empno , m.ename, m.empno
FROM emp e INNER JOIN emp m
ON e.mgr = m.empno;
```

Summary

SELECT *tablel. Column , table2. column*
FROM *tablel , table2*
WHERE *tablel. columnl* = **table2.** *column2;*

- **Equijoin**
- **Non-equijoin**
- **Outer join**
- **Self join**

Summary

There are multiple ways to join tables. The common thread, though, is that you want to link them through a condition in the WHERE clause. The method you choose will be based on the required result and the data structures that you are using.

Exercices

Solution 1
```
SELECT e.ename, e.deptno, d.dname
FROM emp e , dept d
WHERE e.deptno = d.deptno ;
```

Solution 2
```
SELECT e.job, d.loc
FROM emp e , dept d
WHERE e.deptno = d.deptno
AND e.deptno = 30;
```

Solution 3
```
SELECT e.ename, d.dname, d.loc
FROM emp e , dept d
WHERE comm IS NOT NULL
AND e.deptno = d.deptno ;
```

Solution 4
```
SELECT e.ename, d.dname, d.loc
FROM emp e , dept d
WHERE comm IS NOT NULL
AND e.deptno = d.deptno ;
```

Solution 5
```
SELECT e.ename, e.job, e.deptno, d.dname
FROM emp e, dept d
WHERE e.deptno = d.deptno
AND
d.loc = 'DALLAS' ;
```

Solution 6
```
SELECT  e.ename "İşçi"  , e.empno "İşçi No" ,
          m.ename  "Manager"  , m.empno  "Mgr No"
FROM emp e,  emp m
WHERE  e.mgr = m.empno ;
```

Solution 7
```
SELECT  e.ename "İşçi"  , e.empno "İşçi No" ,
          m.ename  "Manager"  , m.empno  "Mgr No"
FROM emp e,  emp m
WHERE  e.mgr = m.empno(+) ;
```

Solution 8a

```
SELECT  e.deptno "Bölüm" ,  e.ename "İşçi" ,
          b.ename
FROM emp  e ,  emp  b
WHERE  e.deptno = b.deptno
ORDER BY e.empno ;
```

Solution 8b

```
SELECT  e.deptno "Bölüm" ,  e.ename "İşçi" ,
          b.ename
FROM emp  e ,  emp  b
WHERE  e.deptno = b.deptno
        AND  e.ename != b.ename
ORDER BY e.empno ;
```

Solution 8c

```
SELECT  e.deptno "Bölüm" ,  e.ename "İşçi" , b.ename
FROM emp  e ,  emp  b
WHERE  e.deptno = b.deptno
AND e.ename <> b.ename
ORDER BY e.deptno ;
```

Solution 9a

```
DESC salgrade;
```

Solution 9b

```
SELECT  e.ename "İşçi" , e.job "İşi" , d.dname "Bölümü" ,
          e.sal "Maaş" , s.grade  "Barem"
FROM  emp  e,  dept  d ,  salgrade  s
WHERE
             e.deptno = d.deptno
      AND    e.sal  BETWEEN  s.losal  AND  s.hisal ;
```

Solution 10

```
SELECT  e.ename "İşçi" , e.hiredate "İşe Giriş Tarihi" , b.hiredate "Blake"
FROM  emp  e,  emp  b
WHERE
      e.hiredate > b.hiredate
      AND    b.ename = 'BLAKE' ;
```

Solution 11

```
SELECT  e.ename "İşçi" , e.hiredate "İşe Giriş Tarihi" , m.ename "Manageri" ,
m.hiredate "Managerin Giriş Tar"
FROM  emp  e,  emp  m
WHERE
      e.hiredate < m.hiredate
      AND    e.mgr = m.empno ;
```