

Lab Manual
Managing Data in RDBMS using SQL
(Advanced Query Language)
Week-4

Joining on Database Tables

This topic is about displaying/ accessing data from multiple tables by “Joining” them. Also, the topic covers how to obtain data from more than one table, using different available joining methods.

Several times the requirement may be as such, that more than one table needs to be accessed to get the desired report displayed. For example, in the database “EmployeeInfo”, “Employee” table has all the information about employee like: employee’s number, name, salary, hiredate, his job and department number in which employee is working in) and in the “Department” table, department’s number, name and it’s location information is available.

We can see the attribute “DEPTNO” exists in both the table.

However, if we want to display the employee’s number, name and the location of the department they are working in, then both the tables need to get accessed in the same query.

Empno	Name	Deptno
7839	King	10
7698	Blake	30
.....
7934	Miller	10

Deptno	Name	Location
10	Accounting	New York
20	Research	Dallas
30	Sales	Chicago
40	Operations	Boston

The expected outcome from the above two tables must be displayed like:

Empno	Name	Location
7839	King	New York
7698	Blake	Chicago
.....
7934	Miller	New York

***** To produce the report, we need to link “Employee” and “Department” tables and access data from both of them.**

Joining:

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.

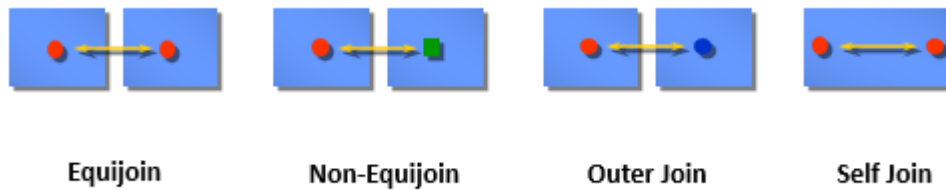
To display data from two or more related tables, simple join condition in the WHERE clause in the syntax will look like:

```

SELECT          table1.column, table2.column
FROM            table1, table2
WHERE           table1.column1 = table2.column2;
```

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

Types of Joins



Equijoin

To determine the name of an employee's department, we compare the value in the DEPTNO column in the EMPLOYEE table with the DEPTNO values in the DEPARTMENT table. The relationship between the EMPLOYEE and DEPARTMENT tables 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.

Equijoins are also called simple joins or inner joins.

EMPLOYEE			DEPARTMENT		
EMPNO	ENAME	DEPTNO	DEPTNO	DNAME	LOC
7839	KING	10	10	ACCOUNTING	NEW YORK
7698	BLAKE	30	30	SALES	CHICAGO
7782	CLARK	10	10	ACCOUNTING	NEW YORK
7566	JONES	20	20	RESEARCH	DALLAS
7654	MARTIN	30	30	SALES	CHICAGO
7499	ALLEN	30	30	SALES	CHICAGO
7844	TURNER	30	30	SALES	CHICAGO
7900	JAMES	30	30	SALES	CHICAGO
7521	WARD	30	30	SALES	CHICAGO
7902	FORD	20	20	RESEARCH	DALLAS
7369	SMITH	20	20	RESEARCH	DALLAS
...			...		
14 rows selected.			14 rows selected.		

Foreign key
Primary key

Now, let's see the syntax of retrieving/accessing data with Equijoin:

```
SQL> SELECT employee.empno, employee.ename,
        employee.deptno, department.deptno, department.loc
```

```
FROM employee, department
WHERE employee.deptno = department.deptno;
```

Here, the SELECT clause specifies the column names to access:

- 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

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

EMPNO	ENAME	DEPTNO	DEPTNO	LOC
7839	KING	10	10	NEW YORK
7698	BLAKE	30	30	CHICAGO
7782	CLARK	10	10	NEW YORK
7566	JONES	20	20	DALLAS
...				
14 rows selected.				

*** More than two tables can also be joined using the same "Equijoin" method.

Non-Equijoin

EMPLOYEE			SALGRADE		
EMPNO	ENAME	SAL	GRADE	LOSAL	HISAL
7839	KING	5000	1	700	1200
7698	BLAKE	2850	2	1201	1400
7782	CLARK	2450	3	1401	2000
7566	JONES	2975	4	2001	3000
7654	MARTIN	1250	5	3001	9999
7499	ALLEN	1600			
7844	TURNER	1500			
7900	JAMES	950			
...					
14 rows selected.					

"salary in the EMP table is between low salary and high salary in the SALGRADE table"

The relationship between the EMPLOYEE table and the SALARYGRADE table is a non-equijoin, means that no column in the EMPLOYEE table corresponds directly to a column in the SALARYGRADE table. The relationship between the two tables is that the SAL column in the EMPLOYEE table is between the LOSAL and HISAL column of the SALARYGRADE table. The relationship is obtained using an operator other than equal (=).


```
SQL> SELECT e.ename, e.sal, s.grade
      FROM employee e, salarygrade s
      WHERE e.sal BETWEEN s.losal AND s.hisal;
```

ENAME	SAL	GRADE
-----	-----	-----
JAMES	950	1
SMITH	800	1
ADAMS	1100	1
...		
14 rows selected.		

Outer Join

If a row does not satisfy a join condition, more elaborately, any of the row will not appear in the query result after performing any equijoin, then we can approach to use Outer Join. For example, in the equijoin condition of EMPLOYEE and DEPARTMENT tables, department "OPERATIONS" does not appear because no employee data is recorded as such who works in that department.

EMPLOYEE		DEPARTMENT	
ENAME	DEPTNO	DEPTNO	DNAME
-----	-----	-----	-----
KING	10	10	ACCOUNTING
BLAKE	30	30	SALES
CLARK	10	10	ACCOUNTING
JONES	20	20	RESEARCH
...		...	
		40	OPERATIONS


No employee in the OPERATIONS department

The conditions that we need to keep on our mind while doing Outer Joins are:

- Within the “Outer Join” operations the rows which do not usually meet the join conditions, can be seen in the output of the report
- Outer Join needs to be specified as, “LEFT” or “RIGHT” join while querying

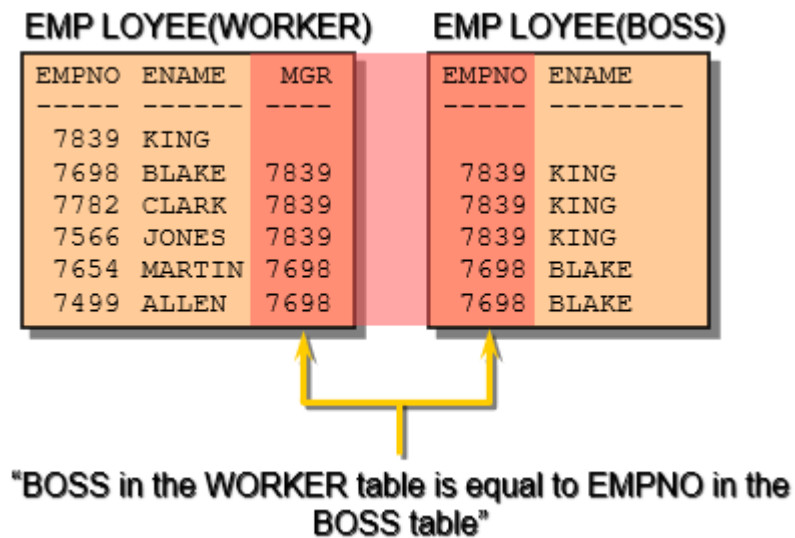
```
SQL> SELECT employee.name, department.deptno, department.name  
FROM employee LEFT JOIN department ON  
employee.deptno=department.deptno
```

Self-Join

In many situations, we need to join a table to itself. For example, in order to find the name of each employee’s manager(where the manager himself is also an employee in the table), you need to join the EMPLOYEE table to itself, and this type of joining performed considering this situation are called “Self Joins”

For instance, to find the name of Blake’s boss, we need to:

- Find Blake in the EMPLOYEE table by looking at the NAME column
- Find the number of the boss for Blake by looking at the BOSS column. Blake’s boss is 7839
- Find the name of the boss with EMPNO 7839 by looking at the NAME column. King’s employee number is 7839, so King is Blake’s boss.



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

```

SQL>SELECT worker.name, boss.name
      FROM employee worker, employee boss
      WHERE worker.boss = boss.empno
  
```

```

WORKER.NAME  BOSS.NAME
-----
BLAKE        KING
CLARK        KING
JONES        KING
MARTIN       BLAKE
...
13 rows selected.
  
```

Subquery on Database Table/Tables

This topic describes the problems that different types of subqueries can solve through single and multi-row outputs while accessing database.

Subquery is a advanced features used within the “Select” block in any SQL statement.

For example, if we are asked to find the salaries greater than Jone’s salary from the employee table then to solve this problem, we need two queries:

What is the salary of Jone?
Who earns more than Jone from the employee table?

However, to find both the answers combinedly through a single query we can use “subquery”. While performing subquery, there will be one inner “Select” block which will find a value (Jone’s salary). This value will than be compared with the other values (other employee’s salaries) sequentially to identify the “greater salaries” in the outer “Select” block/ main block of the query.

```
SQL> SELECT sal, ename  
FROM employee  
WHERE SALARY> (SELECT sal  
                FROM EMPLOYEE  
                WHERE ename= 'Jones');
```

While performing Subqueries always to remember:

- The subquery (inner query) executes once before the main query
- The result of the subquery is used by the main query (outer query)

Note: A subquery is a SELECT statement that is embedded in a clause of another SELECT statement. Powerful statements can be built out of simple ones by using subqueries. These approaches can be very useful when we need to select rows from a table with a condition that depends on the data in the table itself. The subquery can be placed in a number of SQL

clauses:

- WHERE clause
- HAVING clause
- FROM clause

Comparison operators fall into two classes: single-row operators (>, =, >=, <, <>, <=) and multiple-row operators (IN, ANY, ALL).

The subquery is often referred to as a nested SELECT, sub-SELECT, or inner SELECT statement.

Single-Row Subqueries

Single-Row Subqueries always:

- Return only one row
- Use single-row comparison operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

An example of Single-Row Subquery is given below:

```
SQL>SELECT ename, job, sal
FROM emp
WHERE sal > (SELECT MIN(sal)
            FROM emp);
```

The output that generated from this single-row subquery is:

ENAME	JOB	SAL
SMITH	CLERK	800

Multiple-Row Subqueries

Multiple-Row Subqueries always:

- Return MORE THAN ONE row
- Use multiple-row comparison operators

Operator	Meaning
IN	Equal to any member in the list
ANY	Compare value to each value returned by the subquery
ALL	Compare value to every value returned by the subquery

An example of Multiple-Row Subquery is given below:

```
SQL>SELECT empno, ename, job
FROM emp
WHERE sal < ANY (SELECT sal
FROM employee WHERE job = 'CLERK')
AND job <> 'CLERK'
```

EMPNO	ENAME	JOB
7654	MARTIN	SALESMAN
7521	WARD	SALESMAN

```
SQL>SELECT empno, ename, job
FROM emp
WHERE sal > ALL (SELECT avg(sal)
FROM employee
GROUP BY deptno) ;
```

EMPNO	ENAME	JOB
7839	KING	PRESIDENT
7566	JONES	MANAGER
7902	FORD	ANALYST
7788	SCOTT	ANALYST

Guidelines for using Subqueries:

- Enclose subqueries in parentheses
- Place subqueries on the right side of the comparison operator
- Use single-row operators with single-row subqueries
- Use multiple-row operators with multiple-row subqueries

However, Subqueries can be applied on multiple table(s) at a time. While performing subqueries on more than one table we need to ensure that the “Joining” among those tables are done properly.

An example is given below:

Let’s assume we want to query about the employee’s salaries who earns more than the average salary from the employee table, along with the department’s name they are working in.

However, the department’s name is available in a different table than employee table, as a result this is confirmed that we will be accessing both the tables, employee and department together by joining them.

```
SQL>select e.salary, d.name  
      from employee e, department d  
      where e.deptno=d.deptno  
      and e.salary> (select avg (e.salary)  
                    from employee e));
```