جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قسم الامن السيبراني

Lecture: (7)

Subject: Database Systems
Level: Second
Lecturer: Asst. Lecturer Qusai AL-Durrah

Table Joining

In a reporting system often, you might require to combine data from two or more tables to get the required information for analysis and reporting. To retrieve data from two or more tables, you have to combine the tables through the operation known as "Joining of tables". Joining is a method of establishing a relationship between tables using a common column.

An SQL join clause combines columns from one or more tables in a relational database. It creates a set that can be saved as a table or used as it is. **A JOIN is a means for combining columns from one (self-join) or more tables by using values common to each**. ANSI-standard SQL specifies five types of JOIN: INNER, LEFT OUTER, RIGHT OUTER, FULL OUTER and CROSS. As a special case, a table (base table, view, or joined table) can JOIN to itself in a self-join.

Relational databases are usually normalized to **eliminate duplication** of information such as when entity types have one-to-many relationships. For example, a Department may be associated with a number of Employees. Joining separate tables for Department and Employee effectively creates another table which combines the information from both tables.

All subsequent explanations on join types in this article make use of the following two tables. The rows in these tables serve to illustrate the effect of different types of joins and join predicates. In the following tables the DepartmentID column of the Department table (which can be designated as Department.DepartmentID) is the primary key, while Employee.DepartmentID is a foreign key.

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Joining Types

1. Cross join

CROSS JOIN returns the Cartesian product of rows from tables in the join. In other words, it will produce rows which combine each row from the first table with each row from the second table.

Example of an implicit cross join:

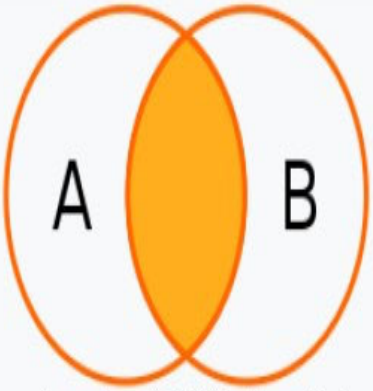
SELECT * FROM employee, department;

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Rafferty	31	Sales	31
Jones	33	Sales	31
Heisenberg	33	Sales	31
Smith	34	Sales	31
Robinson	34	Sales	31
Williams	NULL	Sales	31
Rafferty	31	Engineering	33
Jones	33	Engineering	33
Heisenberg	33	Engineering	33
Smith	34	Engineering	33
Robinson	34	Engineering	33
Williams	NULL	Engineering	33
Rafferty	31	Clerical	34
Jones	33	Clerical	34
Heisenberg	33	Clerical	34
Smith	34	Clerical	34
Robinson	34	Clerical	34
Williams	NULL	Clerical	34
Rafferty	31	Marketing	35
Jones	33	Marketing	35
Heisenberg	33	Marketing	35
Smith	34	Marketing	35
Robinson	34	Marketing	35
Williams	NULL	Marketing	35

The cross join does not itself apply any predicate to filter rows from the joined table. The results of a cross join can be filtered by using a WHERE clause which may then produce the equivalent of an inner join.

2. Inner join

An inner join requires each row in the two joined tables to have matching column values, and is a commonly used join operation in applications but should not be assumed to be the best choice in all situations. Inner join creates a new result table by combining column values of two tables (A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied by matching non-NULL values, column values for each matched pair of rows of A and B are combined into a result row.



A Venn Diagram representing an Inner Join SQL statement between the tables A and B.

The "explicit join notation" uses the JOIN keyword, optionally preceded by the INNER keyword, to specify the table to join, and the ON keyword to specify the predicates for the join, as in the following example:

SELECT <fields> FROM <Table 1> INNER JON <Table 2> ON <Condition>

Example 1 “explicit join notation”:

SELECT employee.LastName, employee.DepartmentID, department.DepartmentName
FROM employee
INNER JOIN department **ON**
employee.DepartmentID = department.DepartmentID

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Robinson	34	Clerical	34
Jones	33	Engineering	33
Smith	34	Clerical	34
Heisenberg	33	Engineering	33
Rafferty	31	Sales	31

Example 2 “implicit join notation”:

SELECT * FROM employee, department

WHERE employee.DepartmentID = department.DepartmentID;

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

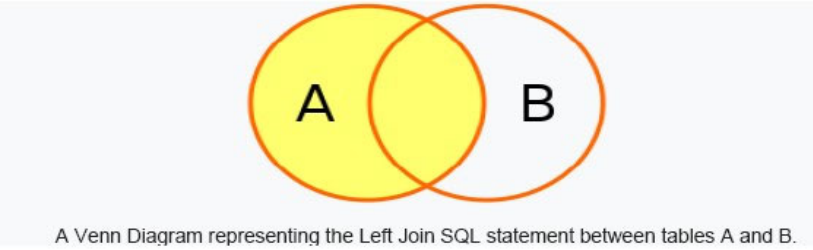
Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Robinson	34	Clerical	34
Jones	33	Engineering	33
Smith	34	Clerical	34
Heisenberg	33	Engineering	33
Rafferty	31	Sales	31

3. Left outer join

The result of a left outer join (or simply **left join**) for tables A and B always contains all rows of the "left" table (A), even if the join-condition does not find any matching row in the "right" table (B). This means that if the ON clause matches 0 (zero) rows in B (for a given row in A), the join will still return a row in the result (for that row), but with NULL in each column from B.



A Venn Diagram representing the Left Join SQL statement between tables A and B.

A **left outer join** returns all the values from an inner join plus all values in the left table that do not match to the right table, including rows with NULL (empty) values in the link column.

For example, this allows us to find an employee's department, but still shows employees that have not been assigned to a department (contrary to the inner-join example above, where unassigned employees were excluded from the result).

Example of a left outer join (the OUTER keyword is optional), with the additional result row (compared with the inner join) italicized:

SELECT * FROM employee **LEFT OUTER JOIN**

department **ON** employee.DepartmentID = department.DepartmentID;

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

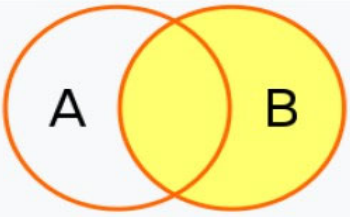
Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Jones	33	Engineering	33
Rafferty	31	Sales	31
Robinson	34	Clerical	34
Smith	34	Clerical	34
Williams	NULL	NULL	NULL
Heisenberg	33	Engineering	33

4. Right outer join

A right outer join (or right join) closely resembles a left outer join, except with the treatment of the tables reversed. Every row from the "right" table (B) will appear in the joined table at least once. If no matching row from the "left" table (A) exists, NULL will appear in columns from A for those rows that have no match in B.



A Venn Diagram representing the Right Join SQL statement between tables A and B.

A right outer join returns all the values from the right table and matched values from the left table (NULL in the case of no matching join predicate). For example, this allows us to find each employee and his or her department, but still show departments that have no employees.

Below is an example of a right outer join (the OUTER keyword is optional), with the additional result row italicized:

SELECT * FROM employee **RIGHT OUTER JOIN** department **ON** employee.DepartmentID = department.DepartmentID;

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Heisenberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	*Marketing*	*35*

5. Full outer join

a full outer join combines the effect of applying both left and right outer joins. Where rows in the FULL OUTER JOINed tables do not match, the result set will have NULL values for every column of the table that lacks a matching row. For those rows that do match, a single row will be produced in the result set (containing columns populated from both tables).

For example, this allows us to see each employee who is in a department and each department that has an employee, but also see each employee who is not part of a department and each department which doesn't have an employee.

Example of a full outer join (the OUTER keyword is optional):

SELECT * FROM employee **FULL OUTER JOIN** department **ON** employee.DepartmentID = department.DepartmentID;

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Williams	NULL	NULL	NULL
Heisenberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	*Marketing*	*35*

Example: Consider the following three tables :

teachers table

```
 id |  name
----+--------
  1 | Volker
  2 | Elke
(2 rows)
```

Projects table

```
 id |    name    | duration | teacher
----+------------+----------+---------
  1 | compiler   |      180 |       1
  2 | xpaint     |      120 |       1
  3 | game       |      250 |       2
  4 | Perl       |       80 |       4
(4 rows)
```

Assign table

```
 project | stud | percentage
---------+------+------------
       1 |    2 |         10
       1 |    4 |         60
       1 |    1 |         30
       2 |    1 |         50
       2 |    4 |         50
       3 |    2 |         70
       3 |    4 |         30
(7 rows)
```

. SELECT * FROM teachers, projects *where* teachers.id = projects.id;

```
 id |  name  | id |   name   | duration | teacher
----+--------+----+----------+----------+---------
  1 | Volker |  1 | compiler |      180 |       1
  2 | Elke   |  2 | xpaint   |      120 |       1
```

SELECT * FROM teachers, projects where teachers.id != projects.id;

```
 id |  name  | id |   name   | duration | teacher
----+--------+----+----------+----------+---------
  1 | Volker |  2 | xpaint   |      180 |       1
  1 | Volker |  3 | game     |      180 |       1
  1 | Volker |  4 | Perl     |      180 |       1
  2 | Elke   |  1 | compiler |      120 |       1
  2 | Elke   |  3 | game     |      120 |       1
  2 | Elke   |  4 | Perl     |      120 |       1
```