Contents

[JOIN Introduction 2](#_Toc523314034)

[Types of Join 2](#_Toc523314035)

[Join notation 2](#_Toc523314036)

[Implicit join notation 3](#_Toc523314037)

[Explicit join notation 3](#_Toc523314038)

[Example Data set up 3](#_Toc523314039)

[Tables with data 3](#_Toc523314040)

[SQL Queries: 4](#_Toc523314041)

[Cross join 4](#_Toc523314042)

[Inner join 6](#_Toc523314043)

[References: 7](#_Toc523314044)

# JOIN Introduction

An SQL **join** clause combines columns from one or more tables in a relational database.

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**.

# Types of Join

1. Cross join
2. Inner join
   1. Equi-join
      1. Natural join
3. Outer join
   1. Left outer join
   2. Right outer join
   3. Full outer join
4. Self-join

# Join notation

SQL specifies two different syntactical ways to express joins: the "**explicit join notation**" and the "**implicit join notation**".

## Implicit join notation

The "**implicit join notation**" is no longer considered a best practice, although database systems still support it.

## Explicit join notation

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

# Example Data set up

Relational databases are usually normalized to eliminate duplication of information such as when entity types have one-to-many relationships.

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.

## Tables with data

|  |  |  |  |
| --- | --- | --- | --- |
| 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 | |

Note: In the Employee table above, the employee "Williams" has not been assigned to any department yet. Also, note that no employees are assigned to the "Marketing" department.

## SQL Queries:

**CREATE** **TABLE** department

(

DepartmentID **INT** Primary **key**,

DepartmentName **VARCHAR**(20)

);

**CREATE** **TABLE** employee

(

LastName **VARCHAR**(20),

DepartmentID **INT** **references** department(DepartmentID)

);

**INSERT** **INTO** department **VALUES**(31, 'Sales');

**INSERT** **INTO** department **VALUES**(33, 'Engineering');

**INSERT** **INTO** department **VALUES**(34, 'Clerical');

**INSERT** **INTO** department **VALUES**(35, 'Marketing');

**INSERT** **INTO** employee **VALUES**('Rafferty', 31);

**INSERT** **INTO** employee **VALUES**('Jones', 33);

**INSERT** **INTO** employee **VALUES**('Heisenberg', 33);

**INSERT** **INTO** employee **VALUES**('Robinson', 34);

**INSERT** **INTO** employee **VALUES**('Smith', 34);

**INSERT** **INTO** employee **VALUES**('Williams', **NULL**);

# 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 explicit cross join:

**SELECT** \*

**FROM** employee **CROSS** **JOIN** department;

Example of an implicit cross join:

**SELECT** \*

**FROM** employee, department;

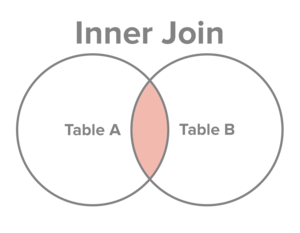
The cross join does not itself apply any predicate to filter rows from the joined table. The results of a cross join can be filtered by using a WHERE clause which may then produce the equivalent of an inner join.

Normal uses are for checking the server's performance.

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

# Inner join

Inner Join returns all records from joining tables, where the join condition is met.



The result of the join can be defined as the outcome of first taking the **Cartesian product (or Cross join)** of all rows in the tables (combining every row in table A with every row in table B) and then returning all rows which satisfy **the join predicate**. Actual SQL implementations normally use other approaches, such as **hash joins** or **sort-merge joins**, since computing the Cartesian product is slower and would often require a prohibitively large amount of memory to store.

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.

Example of **Inner Join**

**SELECT** employee.LastName, employee.DepartmentID, department.DepartmentName

**FROM** employee

**INNER** **JOIN** department **ON**

employee.DepartmentID = department.DepartmentID

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

Using implicit join notation:

**SELECT** \*

**FROM** employee, department

**WHERE** employee.DepartmentID = department.DepartmentID;

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

The employee "Williams" and the department "Marketing" do not appear in the query execution results. Neither of these has any matching rows in the other respective table: "Williams" has no associated department, and no employee has the department ID 35 ("Marketing").

When joining tables on columns that can contain NULL values, since NULL will never match any other value (not even NULL itself), unless the join condition explicitly uses a combination predicate that first checks that the joins columns are NOT NULL before applying the remaining predicate condition(s)

The Inner join can only be safely used in a database that enforces referential integrity or where the join columns are guaranteed not to be NULL.

# Equi-join

An equi-join is a specific type of comparator-based join, that uses only equality comparisons in the join-predicate. Using other comparison operators (such as <) disqualifies a join as an equi-join.

**SELECT** \*

**FROM** employee **JOIN** department

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

We can write equi-join as below,

**SELECT** \*

**FROM** employee, department

**WHERE** employee.DepartmentID = department.DepartmentID;

If columns in an equi-join have the same name, SQL-92 provides an optional shorthand notation for expressing equi-joins, by way of the **USING** construct

**SELECT** \*

**FROM** employee **INNER** **JOIN** department **USING** (DepartmentID);

The USING clause is not supported by MS SQL Server and Sybase.

# Natural join

The natural join is a special case of equi-join. The result of the natural join is the set of all combinations of tuples in R and S that are equal on their common attribute names. For an example consider the tables Employee and Dept and their natural join:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | *Employee* | | | | **Name** | **EmpId** | **DeptName** | | Harry | 3415 | Finance | | Sally | 2241 | Sales | | George | 3401 | Finance | | Harriet | 2202 | Sales | | | | | |  |  | | --- | --- | | *Dept* | | | **DeptName** | **Manager** | | Finance | George | | Sales | Harriet | | Production | Charles | | |
| *Employee*  ⋈ {\displaystyle \bowtie }  *Dept* | | | | |
| **Name** | **EmpId** | **DeptName** | **Manager** | |
| Harry | 3415 | Finance | George | |
| Sally | 2241 | Sales | Harriet | |
| George | 3401 | Finance | George | |
| Harriet | 2202 | Sales | Harriet | |

In particular, the natural join allows the combination of relations that are associated by a foreign key. For example, in the above example a foreign key probably holds from Employee.DeptName to Dept.DeptName and then the natural join of Employee and Dept combines all employees with their departments. This works because the foreign key holds between attributes with the same name. If this is not the case such as in the foreign key from Dept.manager to Employee.Name then these columns have to be renamed before the natural join is taken.

In the case that no columns with the same names are found, the result is a cross join.

**SELECT** \*

**FROM** employee **NATURAL** **JOIN** department;

PostgreSQL, MySQL and Oracle support natural joins; Microsoft T-SQL and IBM DB2 do not.

# References:

https://en.wikipedia.org/wiki/Join\_(SQL)

http://www.sql-join.com/sql-join-types/