**Join:**

A [SQL](https://en.wikipedia.org/wiki/SQL) join clause combines [records](https://en.wikipedia.org/wiki/Row_(database)) from two or more [tables](https://en.wikipedia.org/wiki/Table_(database)) in a relational [database](https://en.wikipedia.org/wiki/Database). It creates a set that can be saved as a table or used as it is. A join is a means for combining [fields](https://en.wikipedia.org/wiki/Field_(computer_science)) from two tables (or more) 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](https://en.wikipedia.org/wiki/View_(database)), or joined table) can join to itself in a self-join [1].

A programmer writes a join statement to identify the records for joining. If the evaluated predicate is true, the combined record is then produced in the expected format, a record set or a temporary table [1].

**Natural join:**

Natural join (\bowtie) is a [binary operator](https://en.wikipedia.org/wiki/Binary_relation) that is written as (R \bowtie S) where R and S are [relations](https://en.wikipedia.org/wiki/Relation_(database)).[[2]](https://en.wikipedia.org/wiki/Join_(SQL)#cite_note-2) The result of the natural join is the set of all combinations of [tuples](https://en.wikipedia.org/wiki/Tuples) in R and S that are equal on their common attribute names [2].

The natural join is arguably one of the most important operators since it is the relational counterpart of logical AND. Note carefully that if the same variable appears in each of two predicates that are connected by AND, then that variable stands for the same thing and both appearances must always be substituted by the same value. In particular, natural join allows the combination of relations that are associated by a [foreign key](https://en.wikipedia.org/wiki/Foreign_key) [2].

More formally the semantics of the natural join are defined as follows:

R \bowtie S = \left\{ t \cup s \ \vert \ t \in R \ \land \ s \in S \ \land \ \mathit{Fun}(t \cup s) \right\}

* **Inner join:**

An inner join requires each record in the two joined tables to have matching records, and is a commonly used join operation in [applications](https://en.wikipedia.org/wiki/Application_software) 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.

The result of the join can be defined as the outcome of first taking the [Cartesian product](https://en.wikipedia.org/wiki/Cartesian_product) (or [Cross join](https://en.wikipedia.org/wiki/Join_(SQL)#Cross_join)) of all records in the tables (combining every record in table A with every record in table B) and then returning all records which satisfy the join predicate. Actual SQL implementations normally use other approaches, such as [hash joins](https://en.wikipedia.org/wiki/Hash_join) or [sort-merge joins](https://en.wikipedia.org/wiki/Sort-merge_join), since computing the Cartesian product is slower and would often require a prohibitively large memory space to store.

SQL specifies two different syntactical ways to express joins: "explicit join notation" and "implicit join notation". Although "implicit join notation" was deprecated in 1992, and its use is not considered a best practice, database systems still support it.

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** \*

**FROM** employee

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

The "implicit join notation" simply lists the tables for joining, in the FROM clause of the SELECT statement, using commas to separate them. Thus it specifies a [cross join](https://en.wikipedia.org/wiki/Join_(SQL)#Cross_join), and the WHERE clause may apply additional filter-predicates (which function comparably to the join-predicates in the explicit notation).

The following example is equivalent to the previous one, but this time using implicit join notation:

**SELECT** \*

**FROM** employee, department

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

* **Left Join:**

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

For example, this allows us to find an employee's department, but still shows the employee(s) even when they have not been assigned to a department (contrary to the inner-join example above, where unassigned employees were excluded from the result) [3].

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;

* **Right 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 records that have no match in 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 [4].

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;

**References:**

1. <https://en.wikipedia.org/wiki/Join_(SQL)>
2. <https://en.wikipedia.org/wiki/Join_(SQL)#Natural_join_.28.E2.8B.88.29>
3. https://en.wikipedia.org/wiki/Join\_(SQL)#Natural\_join\_.28.E2.8B.88.29
4. <https://en.wikipedia.org/wiki/Join_(SQL)#Right_outer_join>
5. <http://www.w3schools.com/sql/sql_join_left.asp>