A SQL join is where you run a query that joins multiple tables.

This SQL joins tutorial presents basic examples of SQL joins, as well as an introduction to the various join types.

SQL Join Types

| **Join Type** | **Description** |
| --- | --- |
| Venn diagram of an inner join INNER JOIN | Returns rows when there is at least one row in both tables that match the join condition. |
| Venn diagram of left join LEFT OUTER JOIN or LEFT JOIN | Returns rows that have data in the left table (left of the JOIN keyword), even if there’s no matching rows in the right table. |
| Venn diagram of right join RIGHT OUTER JOIN or RIGHT JOIN | Returns rows that have data in the right table (right of the JOIN keyword), even if there’s no matching rows in the left table. |
| Venn diagram of full join FULL OUTER JOIN or FULL JOIN | Returns all rows, as long as there’s matching data in one of the tables. |
| CROSS JOIN | Returns rows which combine each row from the first table with each row from the second table. |

The ANSI SQL standard specifies five types of joins, as listed in the following table.

There are also other terms for various join operations, such as the following:

| **Join** | **Description** |
| --- | --- |
| Self-join | When a table joins to itself. |
| Natural join | An implicit join based on the common columns in the two tables being joined. |
| Equi-join | A join containing only equality comparisons in the join predicate. |

SQL Join Syntax

Inner joins can be specified in either the FROM or WHERE clauses. Outer joins and cross joins can be specified in the FROM clause only.

To create a SQL join in the FROM clause, do something like this:

SELECT \*

FROM Table1 < JoinType > Table2 [ ON ( JoinCondition ) ]

Where JoinType specifies what kind of join is performed, and JoinCondition defines the predicate to be evaluated for each pair of joined rows.

To specify a join in the WHERE clause, do something like this:

SELECT \*

FROM Table1, Table2 [ WHERE ( JoinCondition ) ]

Again, JoinCondition defines the predicate to be evaluated for each pair of joined rows.

Also, everything enclosed in square brackets ([]) is optional.

Sample Tables for the Examples in this Tutorial

Most of the examples in this tutorial perform joins against the following two tables.

The PetTypes table:

+-------------+-----------+

| PetTypeId   | PetType   |

|-------------+-----------|

| 1           | Bird      |

| 2           | Cat       |

| 3           | Dog       |

| 4           | Rabbit    |

+-------------+-----------+

(4 rows affected)

The Pets table:

+---------+-------------+-----------+-----------+------------+

| PetId   | PetTypeId   | OwnerId   | PetName   | DOB        |

|---------+-------------+-----------+-----------+------------|

| 1       | 2           | 3         | Fluffy    | 2020-11-20 |

| 2       | 3           | 3         | Fetch     | 2019-08-16 |

| 3       | 2           | 2         | Scratch   | 2018-10-01 |

| 4       | 3           | 3         | Wag       | 2020-03-15 |

| 5       | 1           | 1         | Tweet     | 2020-11-28 |

| 6       | 3           | 4         | Fluffy    | 2020-09-17 |

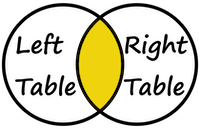
| 7       | 3           | 2         | Bark      | NULL       |

| 8       | 2           | 4         | Meow      | NULL       |

+---------+-------------+-----------+-----------+------------+

(8 rows affected)

The Inner Join



The SQL INNER JOIN returns rows when there is at least one row in both tables that match the join condition.

SELECT

Pets.PetName,

PetTypes.PetType

FROM Pets

INNER JOIN PetTypes

ON Pets.PetTypeId = PetTypes.PetTypeId;

Result:

-----------+-----------+

| PetName   | PetType   |

|-----------+-----------|

| Fluffy    | Cat       |

| Fetch     | Dog       |

| Scratch   | Cat       |

| Wag       | Dog       |

| Tweet     | Bird      |

| Fluffy    | Dog       |

| Bark      | Dog       |

| Meow      | Cat       |

+-----------+-----------+

(8 rows affected)

To specify an inner join in the FROM clause, we use INNER JOIN. We also use the ON keyword to define the predicate to be evaluated for each pair of joined rows.

Regardless of the join type, we qualify our column names with the table names. The reason we do this is to avoid any ambiguity regarding the column names between the tables. Both tables could have columns of the same name (like in our example), and in such cases, the DBMS won’t know which column you’re referring to. Prefixing the column names with their table names ensures that you’re referencing the right column, and prevents any errors that could result from any ambiguity about which column you’re referring to.

In this example, both tables have a PetTypeId column. The Pets.PetTypeId column is a [foreign key](https://database.guide/what-is-a-foreign-key/) to the PetTypes.PetTypeId column, which is the [primary key](https://database.guide/what-is-a-primary-key/) for that table.

In this example, we can see that all pets are returned, but not all of the pet types are returned. There are no rabbits in the Pets table, and so the Rabbits pet type is not returned.

The reason the Rabbits type is not returned is because the INNER JOIN only returns rows when there is at least one row in both tables that match the join condition. In this case, Rabbits is only in one table (the PetTypes table).

Note that the join type is optional. Therefore, most (if not all) DBMSs allow you to omit the INNER keyword. When you omit this (i.e. only specify JOIN), it is assumed to be an inner join.

Therefore, we could rewrite the above example to this:

SELECT

Pets.PetName,

PetTypes.PetType

FROM Pets

JOIN PetTypes

ON Pets.PetTypeId = PetTypes.PetTypeId;

Also, as with any SQL statement, the FROM clause can be on one whole line if you prefer:

SELECT

Pets.PetName,

PetTypes.PetType

FROM Pets JOIN PetTypes ON Pets.PetTypeId = PetTypes.PetTypeId;

Aliases

It’s common practice to use table [aliases](https://database.guide/sql-alias-explained/) when performing SQL joins. Aliases help to make the code more concise, and easier to read.

Therefore, we could change the previous example to this:

SELECT

p.PetName,

pt.PetType

FROM Pets p INNER JOIN PetTypes pt

ON p.PetTypeId = pt.PetTypeId;

Result:

-----------+-----------+

| PetName   | PetType   |

|-----------+-----------|

| Fluffy    | Cat       |

| Fetch     | Dog       |

| Scratch   | Cat       |

| Wag       | Dog       |

| Tweet     | Bird      |

| Fluffy    | Dog       |

| Bark      | Dog       |

| Meow      | Cat       |

+-----------+-----------+

(8 rows affected)

The Equi-Join

The above join can also be referred to as an *equi-join*. An equi-join is a join containing only equality comparisons in the join predicate.

Another way of writing the above join is like this:

SELECT

p.PetName,

pt.PetType

FROM

Pets p,

PetTypes pt

WHERE p.PetTypeId = pt.PetTypeId;

Result:

+-----------+-----------+

| PetName   | PetType   |

|-----------+-----------|

| Fluffy    | Cat       |

| Fetch     | Dog       |

| Scratch   | Cat       |

| Wag       | Dog       |

| Tweet     | Bird      |

| Fluffy    | Dog       |

| Bark      | Dog       |

| Meow      | Cat       |

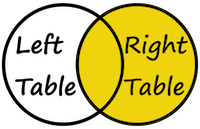
+-----------+-----------+

This is an example of specifying an inner join in the WHERE clause. We simply provided a comma separated list of the tables, and then a WHERE condition. If we’d omitted the WHERE condition, we would’ve ended up with a CROSS JOIN.

Many beginners find the above syntax much easier to understand than the INNER JOIN syntax. Feel free to use this syntax if you prefer, however, be aware that most SQL professionals prefer to use the INNER JOIN syntax from the previous example.

See [SQL Inner Join](https://database.guide/sql-inner-join/) for more examples, including an inner join that joins 3 tables.

The Right Join



Also known as the RIGHT OUTER JOIN, the RIGHT JOIN returns rows that have data in the right table (right of the JOIN keyword), even if there’s no matching rows in the left table.

SELECT

p.PetName,

pt.PetType

FROM Pets p

RIGHT JOIN PetTypes pt

ON p.PetTypeId = pt.PetTypeId;

Result:

+-----------+-----------+

| PetName   | PetType   |

|-----------+-----------|

| Tweet     | Bird      |

| Fluffy    | Cat       |

| Scratch   | Cat       |

| Meow      | Cat       |

| Fetch     | Dog       |

| Wag       | Dog       |

| Fluffy    | Dog       |

| Bark      | Dog       |

| NULL      | Rabbit    |

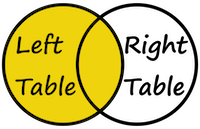
+-----------+-----------+

(9 rows affected)

In this case, we got an extra PetType value – Rabbit – even though there’s no pet in the Pets table of that type. This results in a NULL value in the PetName column against Rabbit.

See [SQL Right Join](https://database.guide/sql-right-join/) for more examples, including a right join that joins 3 tables.

The Left Join



Also known as the LEFT OUTER JOIN, the SQL LEFT JOIN returns rows that have data in the left table (left of the JOIN keyword), even if there’s no matching rows in the right table.

This is the opposite of the RIGHT JOIN.

If we change the previous example to use a left join, we get the following result.

SELECT

p.PetName,

pt.PetType

FROM Pets p

LEFT JOIN PetTypes pt

ON p.PetTypeId = pt.PetTypeId;

Result:

+-----------+-----------+

| PetName   | PetType   |

|-----------+-----------|

| Fluffy    | Cat       |

| Fetch     | Dog       |

| Scratch   | Cat       |

| Wag       | Dog       |

| Tweet     | Bird      |

| Fluffy    | Dog       |

| Bark      | Dog       |

| Meow      | Cat       |

+-----------+-----------+

(8 rows affected)

In this particular case, our results are the same as with the inner join.

However, if we swap the table order around in our FROM clause, we’ll get a similar result to the right join in the previous example.

SELECT

p.PetName,

pt.PetType

FROM PetTypes pt

LEFT JOIN Pets p

ON p.PetTypeId = pt.PetTypeId;

Result:

+-----------+-----------+

| PetName   | PetType   |

|-----------+-----------|

| Tweet     | Bird      |

| Fluffy    | Cat       |

| Scratch   | Cat       |

| Meow      | Cat       |

| Fetch     | Dog       |

| Wag       | Dog       |

| Fluffy    | Dog       |

| Bark      | Dog       |

| NULL      | Rabbit    |

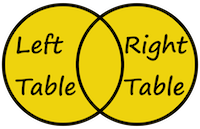
+-----------+-----------+

(9 rows affected)

So you can see that any resulting difference between the left and right joins depends solely on how you order the columns in the FROM clause.

See [SQL Left Join](https://database.guide/sql-left-join/) for more examples, including a left join that joins 3 tables.

The Full Join



The SQL FULL JOIN (or FULL OUTER JOIN) returns all rows, as long as there’s matching data in one of the tables.

In other words, it’s like having both a left and right join in one join.

Here’s an example of a full join.

SELECT

p.PetName,

pt.PetType

FROM Pets p

FULL JOIN PetTypes pt

ON p.PetTypeId = pt.PetTypeId;

Result:

+-----------+-----------+

| PetName   | PetType   |

|-----------+-----------|

| Fluffy    | Cat       |

| Fetch     | Dog       |

| Scratch   | Cat       |

| Wag       | Dog       |

| Tweet     | Bird      |

| Fluffy    | Dog       |

| Bark      | Dog       |

| Meow      | Cat       |

| NULL      | Rabbit    |

+-----------+-----------+

(9 rows affected)

This returns the same result that we got with the right join, but it would’ve returned a different result if there had been a row in the left table that didn’t have a corresponding value in the right table.

Let’s swap the table names around and run it again.

SELECT

p.PetName,

pt.PetType

FROM PetTypes pt

FULL JOIN Pets p

ON p.PetTypeId = pt.PetTypeId;

Result:

+-----------+-----------+

| PetName   | PetType   |

|-----------+-----------|

| Tweet     | Bird      |

| Fluffy    | Cat       |

| Scratch   | Cat       |

| Meow      | Cat       |

| Fetch     | Dog       |

| Wag       | Dog       |

| Fluffy    | Dog       |

| Bark      | Dog       |

| NULL      | Rabbit    |

+-----------+-----------+

(9 rows affected)

Same result.

See [SQL Full Join](https://database.guide/sql-full-join/) for more examples, including a full join that joins 3 tables.

The Cross Join

The SQL CROSS JOIN returns rows which combine each row from the first table with each row from the second table.

In other words, it returns the Cartesian product of rows from tables in the join.

SELECT

p.PetName,

pt.PetType

FROM Pets p

CROSS JOIN PetTypes pt;

Result:

+-----------+-----------+

| PetName   | PetType   |

|-----------+-----------|

| Fluffy    | Bird      |

| Fetch     | Bird      |

| Scratch   | Bird      |

| Wag       | Bird      |

| Tweet     | Bird      |

| Fluffy    | Bird      |

| Bark      | Bird      |

| Meow      | Bird      |

| Fluffy    | Cat       |

| Fetch     | Cat       |

| Scratch   | Cat       |

| Wag       | Cat       |

| Tweet     | Cat       |

| Fluffy    | Cat       |

| Bark      | Cat       |

| Meow      | Cat       |

| Fluffy    | Dog       |

| Fetch     | Dog       |

| Scratch   | Dog       |

| Wag       | Dog       |

| Tweet     | Dog       |

| Fluffy    | Dog       |

| Bark      | Dog       |

| Meow      | Dog       |

| Fluffy    | Rabbit    |

| Fetch     | Rabbit    |

| Scratch   | Rabbit    |

| Wag       | Rabbit    |

| Tweet     | Rabbit    |

| Fluffy    | Rabbit    |

| Bark      | Rabbit    |

| Meow      | Rabbit    |

+-----------+-----------+

(32 rows affected)

As you can probably imagine, this could be very dangerous if you run it against the wrong tables.

It is the same as doing this:

SELECT

p.PetName,

pt.PetType

FROM Pets p, PetTypes pt;

You can add a WHERE clause to a cross join, which will turn it into an inner join.

Like this:

SELECT

p.PetName,

pt.PetType

FROM Pets p

CROSS JOIN PetTypes pt

WHERE p.PetTypeId = pt.PetTypeId;

Result:

+-----------+-----------+

| PetName   | PetType   |

|-----------+-----------|

| Fluffy    | Cat       |

| Fetch     | Dog       |

| Scratch   | Cat       |

| Wag       | Dog       |

| Tweet     | Bird      |

| Fluffy    | Dog       |

| Bark      | Dog       |

| Meow      | Cat       |

+-----------+-----------+

(8 rows affected)

See [SQL Cross Join](https://database.guide/sql-cross-join/) for more examples.

The Natural Join

The SQL NATURAL JOIN is a type of equi-join where the join predicate arises implicitly by comparing all columns in both tables that have the same column names in the joined tables.

The result set contains only one column for each pair of equally named columns. If no columns with the same names are found, the result will be a cross join.

SELECT

Pets.PetName,

PetTypes.PetType

FROM Pets NATURAL JOIN PetTypes;

Result:

petname | pettype

---------+---------

 Fluffy  | Cat

 Fetch   | Dog

 Scratch | Cat

 Wag     | Dog

 Tweet   | Bird

 Fluffy  | Dog

 Bark    | Dog

 Meow    | Cat

(8 rows)

Actually, the natural join isn’t actually a join type, as considered by the ANSI standard. It’s a keyword that you can optionally insert in order to make the join a natural join.

Therefore, we could change the above example to NATURAL INNER JOIN if we wanted to:

SELECT

Pets.PetName,

PetTypes.PetType

FROM Pets NATURAL INNER JOIN PetTypes;

As mentioned earlier, inner joins are the default join type, so if you omit the join type (eg, INNER, LEFT, RIGHT, etc), then it is treated as an inner join.

If the formatting of these results looks different than the previous results, it’s because I had to jump over to [PostgreSQL](https://database.guide/what-is-postgresql/) to run this query. I ran the previous examples in SQL Server, but [SQL Server](https://database.guide/what-is-sql-server/) doesn’t support the natural join.

See [SQL Natural Join](https://database.guide/sql-natural-join/) for more examples, including a natural join that joins 3 tables.

The Self Join

The SQL SELF JOIN is joining a table to itself.

A classic example of a self join is in an Employees table. In such a table, one employee might report to another employee. Therefore, you could use a self join to join the table on its employee ID column and manager ID column.

Suppose we have the following table:

+--------------+-------------+------------+-------------+

| EmployeeId   | FirstName   | LastName   | ReportsTo   |

|--------------+-------------+------------+-------------|

| 1            | Homer       | Connery    | NULL        |

| 2            | Bart        | Pitt       | 1           |

| 3            | Maggie      | Griffin    | 1           |

| 4            | Peter       | Farnsworth | 2           |

| 5            | Marge       | Morrison   | NULL        |

| 6            | Lisa        | Batch      | 5           |

| 7            | Dave        | Zuckerberg | 6           |

| 8            | Vlad        | Cook       | 7           |

+--------------+-------------+------------+-------------+

We can do a self join on this table to return all employees and their managers.

SELECT

CONCAT(e1.FirstName, ' ', e1.LastName) AS Employee,

CONCAT(e2.FirstName, ' ', e2.LastName) AS Manager

FROM Employees e1

LEFT JOIN Employees e2

ON e1.ReportsTo = e2.EmployeeId;

Result:

+------------------+-----------------+

| Employee         | Manager         |

|------------------+-----------------|

| Homer Connery    |                 |

| Bart Pitt        | Homer Connery   |

| Maggie Griffin   | Homer Connery   |

| Peter Farnsworth | Bart Pitt       |

| Marge Morrison   |                 |

| Lisa Batch       | Marge Morrison  |

| Dave Zuckerberg  | Lisa Batch      |

| Vlad Cook        | Dave Zuckerberg |

+------------------+-----------------+