**SQL: JOINS**

This SQL tutorial explains how to use SQL **JOINS** with syntax, visual illustrations, and examples.

**Description**

SQL JOINS are used to retrieve data from multiple tables. A SQL JOIN is performed whenever two or more tables are listed in a SQL statement.

There are 4 different types of SQL joins:

* SQL INNER JOIN (sometimes called simple join)
* SQL LEFT OUTER JOIN (sometimes called LEFT JOIN)
* SQL RIGHT OUTER JOIN (sometimes called RIGHT JOIN)
* SQL FULL OUTER JOIN (sometimes called FULL JOIN)

So let's discuss SQL JOIN syntax, look at visual illustrations of SQL JOINS and explore some examples.

**DDL/DML for Examples**

If you want to follow along with this tutorial, get the DDL to create the tables and the DML to populate the data. Then try the examples in your own database!

**SQL INNER JOIN (simple join)**

Chances are, you've already written a SQL statement that uses an SQL INNER JOIN. It is the most common type of SQL join. SQL INNER JOINS return all rows from multiple tables where the join condition is met.

Syntax

The syntax for the INNER JOIN in SQL is:

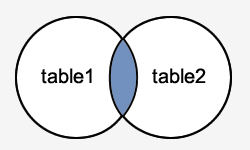
SELECT columns

FROM table1

INNER JOIN table2 ON table1.column = table2.column;

Visual Illustration

In this visual diagram, the SQL INNER JOIN returns the shaded area:



The SQL INNER JOIN would return the records where *table1* and *table2* intersect.

Example

Let's look at an example of how to use the INNER JOIN in a query.

In this example, we have a table called *customers* with the following data:

| customer\_id | last\_name | first\_name |  | favorite\_website |
| --- | --- | --- | --- | --- |
| 4000 | Jackson | Joe |  | techonthenet.com |
| 5000 | Smith | Jane |  | digminecraft.com |
| 6000 | Ferguson | Samantha |  | bigactivities.com |
| 7000 | Reynolds | Allen |  | checkyourmath.com |
| 8000 | Anderson | Paige |  | NULL |
| 9000 | Johnson | Derek |  | techonthenet.com |

And a table called *orders* with the following data:

| order\_id | customer\_id | order\_date |
| --- | --- | --- |
| 1 | 7000 | 2016/04/18 |
| 2 | 5000 | 2016/04/18 |
| 3 | 8000 | 2016/04/19 |
| 4 | 4000 | 2016/04/20 |
| 5 | NULL | 2016/05/01 |

Enter the following SQL statement:

SELECT customers.customer\_id, orders.order\_id, orders.order\_date

FROM customers

INNER JOIN orders

ON customers.customer\_id = orders.customer\_id

ORDER BY customers.customer\_id;

There will be 4 records selected. These are the results that you should see:

| customer\_id | order\_id | order\_date |
| --- | --- | --- |
| 4000 | 4 | 2016/04/20 |
| 5000 | 2 | 2016/04/18 |
| 7000 | 1 | 2016/04/18 |
| 8000 | 3 | 2016/04/19 |

This example would return all rows from the *customers* and *orders* tables where there is a matching *customer\_id* value in both the *customers* and *orders* tables.

The rows where *customer\_id* is equal to 6000 and 9000 in the *customers* table would be omitted, since they do not exist in both tables. The row where the *order\_id* is 5 from the *orders* table would be omitted, since the *customer\_id* of NULL does not exist in the *customers* table.

Old Syntax

As a final note, it is worth mentioning that the INNER JOIN example above could be rewritten using the older implicit syntax as follows (but we still recommend using the INNER JOIN keyword syntax):

SELECT customers.customer\_id, orders.order\_id, orders.order\_date

FROM customers, orders

WHERE customers.customer\_id = orders.customer\_id

ORDER BY customers.customer\_id;

**SQL LEFT OUTER JOIN**

Another type of join is called a LEFT OUTER JOIN. This type of join returns all rows from the LEFT-hand table specified in the ON condition and **only** those rows from the other table where the joined fields are equal (join condition is met).

Syntax

The syntax for the LEFT OUTER JOIN in SQL is:

SELECT columns

FROM table1

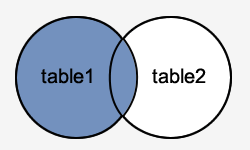
LEFT [OUTER] JOIN table2

ON table1.column = table2.column;

In some databases, the OUTER keyword is omitted and written simply as LEFT JOIN.

Visual Illustration

In this visual diagram, the SQL LEFT OUTER JOIN returns the shaded area:



The SQL LEFT OUTER JOIN would return the all records from *table1* and only those records from *table2* that intersect with *table1*.

Example

Now let's look at an example that shows how to use the LEFT OUTER JOIN in a SELECT statement.

Using the same *customers* table as the previous example:

| customer\_id | last\_name | first\_name | favorite\_website |
| --- | --- | --- | --- |
| 4000 | Jackson | Joe | techonthenet.com |
| 5000 | Smith | Jane | digminecraft.com |
| 6000 | Ferguson | Samantha | bigactivities.com |
| 7000 | Reynolds | Allen | checkyourmath.com |
| 8000 | Anderson | Paige | NULL |
| 9000 | Johnson | Derek | techonthenet.com |

And the *orders* table with the following data:

| order\_id | customer\_id | order\_date |
| --- | --- | --- |
| 1 | 7000 | 2016/04/18 |
| 2 | 5000 | 2016/04/18 |
| 3 | 8000 | 2016/04/19 |
| 4 | 4000 | 2016/04/20 |
| 5 | NULL | 2016/05/01 |

Enter the following SQL statement:

SELECT customers.customer\_id, orders.order\_id, orders.order\_date

FROM customers

LEFT OUTER JOIN orders

ON customers.customer\_id = orders.customer\_id

ORDER BY customers.customer\_id;

There will be 6 records selected. These are the results that you should see:

| customer\_id | order\_id | order\_date |
| --- | --- | --- |
| 4000 | 4 | 2016/04/20 |
| 5000 | 2 | 2016/04/18 |
| 6000 | NULL | NULL |
| 7000 | 1 | 2016/04/18 |
| 8000 | 3 | 2016/04/19 |
| 9000 | NULL | NULL |

This LEFT OUTER JOIN example would return all rows from the *customers* table and only those rows from the *orders* table where the joined fields are equal.

If a *customer\_id* value in the *customers* table does not exist in the *orders* table, all fields in the *orders* table will display as NULL in the result set. As you can see, the rows where *customer\_id* is 6000 and 9000 would be included with a LEFT OUTER JOIN but the *order\_id* and *order\_date* fields display NULL.

**SQL RIGHT OUTER JOIN**

Another type of join is called a SQL RIGHT OUTER JOIN. This type of join returns all rows from the RIGHT-hand table specified in the ON condition and **only** those rows from the other table where the joined fields are equal (join condition is met).

Syntax

The syntax for the RIGHT OUTER JOIN in SQL is:

SELECT columns

FROM table1

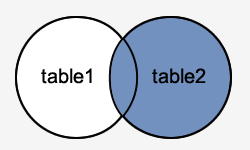
RIGHT [OUTER] JOIN table2

ON table1.column = table2.column;

In some databases, the OUTER keyword is omitted and written simply as RIGHT JOIN.

Visual Illustration

In this visual diagram, the SQL RIGHT OUTER JOIN returns the shaded area:



The SQL RIGHT OUTER JOIN would return the all records from *table2* and only those records from *table1* that intersect with *table2*.

Example

Now let's look at an example that shows how to use the RIGHT OUTER JOIN in a SELECT statement.

Using the same *customers* table as the previous example:

| customer\_id | last\_name | first\_name | favorite\_website |
| --- | --- | --- | --- |
| 4000 | Jackson | Joe | techonthenet.com |
| 5000 | Smith | Jane | digminecraft.com |
| 6000 | Ferguson | Samantha | bigactivities.com |
| 7000 | Reynolds | Allen | checkyourmath.com |
| 8000 | Anderson | Paige | NULL |
| 9000 | Johnson | Derek | techonthenet.com |

And the *orders* table with the following data:

| order\_id | customer\_id | order\_date |
| --- | --- | --- |
| 1 | 7000 | 2016/04/18 |
| 2 | 5000 | 2016/04/18 |
| 3 | 8000 | 2016/04/19 |
| 4 | 4000 | 2016/04/20 |
| 5 | NULL | 2016/05/01 |

Enter the following SQL statement:

SELECT customers.customer\_id, orders.order\_id, orders.order\_date

FROM customers

RIGHT OUTER JOIN orders

ON customers.customer\_id = orders.customer\_id

ORDER BY customers.customer\_id;

There will be 5 records selected. These are the results that you should see:

| customer\_id | order\_id | order\_date |
| --- | --- | --- |
| NULL | 5 | 2016/05/01 |
| 4000 | 4 | 2016/04/20 |
| 5000 | 2 | 2016/04/18 |
| 7000 | 1 | 2016/04/18 |
| 8000 | 3 | 2016/04/19 |

This RIGHT OUTER JOIN example would return all rows from the *orders* table and only those rows from the *customers* table where the joined fields are equal.

If a *customer\_id* value in the *orders* table does not exist in the *customers* table, all fields in the *customers* table will display as NULL in the result set. As you can see, the row where *order\_id* is 5 would be included with a RIGHT OUTER JOIN but the *customer\_id* field displays NULL.

**SQL FULL OUTER JOIN**

Another type of join is called a SQL FULL OUTER JOIN. This type of join returns all rows from the LEFT-hand table and RIGHT-hand table with NULL values in place where the join condition is not met.

Syntax

The syntax for the SQL **FULL OUTER JOIN** is:

SELECT columns

FROM table1

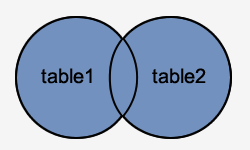
FULL [OUTER] JOIN table2

ON table1.column = table2.column;

In some databases, the OUTER keyword is omitted and written simply as FULL JOIN.

Visual Illustration

In this visual diagram, the SQL FULL OUTER JOIN returns the shaded area:



The SQL FULL OUTER JOIN would return the all records from both *table1* and *table2*.

Example

Let's look at an example that shows how to use the FULL OUTER JOIN in a SELECT statement.

Using the same *customers* table as the previous example:

| customer\_id | last\_name | first\_name | favorite\_website |
| --- | --- | --- | --- |
| 4000 | Jackson | Joe | techonthenet.com |
| 5000 | Smith | Jane | digminecraft.com |
| 6000 | Ferguson | Samantha | bigactivities.com |
| 7000 | Reynolds | Allen | checkyourmath.com |
| 8000 | Anderson | Paige | NULL |
| 9000 | Johnson | Derek | techonthenet.com |

And the *orders* table with the following data:

| order\_id | customer\_id | order\_date |
| --- | --- | --- |
| 1 | 7000 | 2016/04/18 |
| 2 | 5000 | 2016/04/18 |
| 3 | 8000 | 2016/04/19 |
| 4 | 4000 | 2016/04/20 |
| 5 | NULL | 2016/05/01 |

Enter the following SQL statement:

SELECT customers.customer\_id, orders.order\_id, orders.order\_date

FROM customers

FULL OUTER JOIN orders

ON customers.customer\_id = orders.customer\_id

ORDER BY customers.customer\_id;

There will be 7 records selected. These are the results that you should see:

| customer\_id | order\_id | order\_date |
| --- | --- | --- |
| NULL | 5 | 2016/05/01 |
| 4000 | 4 | 2016/04/20 |
| 5000 | 2 | 2016/04/18 |
| 6000 | NULL | NULL |
| 7000 | 1 | 2016/04/18 |
| 8000 | 3 | 2016/04/19 |
| 9000 | NULL | NULL |

This FULL OUTER JOIN example would return all rows from the *orders* table and all rows from the *customers* table. Whenever the joined condition is not met, a NULL value would be extended to those fields in the result set. This means that if a *customer\_id* value in the *customers* table does not exist in the *orders* table, all fields in the *orders* table will display as NULL in the result set. Also, if a *customer\_id* value in the *orders* table does not exist in the *customers* table, all fields in the *customers* table will display as NULL in the result set.

As you can see, the rows where the *customer\_id* is 6000 and 9000 would be included but the *order\_id* and *order\_date* fields for those records contains a NULL value. The row where the *order\_id* is 5 would be also included but the *customer\_id* field for that record has a NULL value.

**Cross Join - Animated**

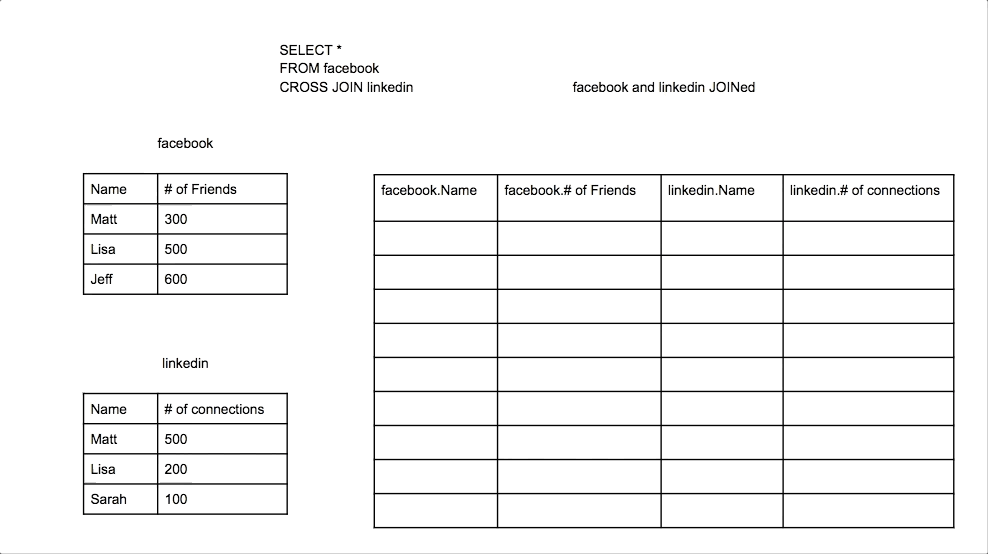
*Last modified: June 22, 2021*

This is the fifth most common type of JOIN in SQL. Cross join does not look for matches between any values in the two data sets. Instead for each row in first table every row of second table will be attached to it and added to the final table one by one.

**SELECT** **\***

**FROM** facebook

**CROSS** **JOIN** linkedin



Why use a CROSS JOIN vs a UNION, LEFT JOIN, RIGHT JOIN, INNER JOIN, FULL OUTER JOIN? To help understand, Let’s think about the different questions they are asking.

* **CROSS JOIN:** How many combinations of friends and connections do I have?
* **UNION:** How many friends do my Facebook friends have and how many connections do my LinkedIn connections have?
* **LEFT join:** How many friends and connections do my Facebook friends have? (Regardless of if they are on LinkedIn)
* **RIGHT join:** How many friends and connections do my LinkedIn connections have? (Regardless of if they are on facebook)
* **INNER join:** How many friends and connections do my friends who are on both on Facebook and LinkedIn have?
* **FULL OUTER join:** How many friends and connections do my Facebook friends or LinkedIn connections have?

An [equi join](https://www.scaler.com/topics/equi-join-in-sql/" \o "" \t "_blank) is any [JOIN operation](https://www.scaler.com/topics/joins-in-sql/) that uses only and only the equals sign. If there is a query with more than one join condition, out of which one condition has an equals sign, and the other doesn't, then this query would be considered a non-equi join in SQL. Thus Equi Joins in SQL joins multiple tables on the basis of an equality condition. In contrast, the Non-Equi Join joins the table on the basis of conditions other than the equality conditions, such as !=, >, <, etc.

**Introduction**

Whenever we have to retrieve data from a table, we use selection, projection, and join methods. Selection is a method in which we have to retrieve data using a condition such as a WHERE condition from a single table. Projection is a method in which we have to retrieve data from a single table without using a condition. We use joins if we want to select or retrieve data from multiple tables. It retrieves data that includes related rows/data from multiple tables.

Equi and Non-Equi Join in SQL are types of joins that retrieve data from multiple tables based on some condition. Equi Join in SQL will retrieve only the data that is equal in both the tables or only retrieves the matching column values; therefore, it uses an equality operator to [join multiple tables](https://www.scaler.com/topics/join-multiple-tables-in-sql/). On the other hand, Non-Equi join is used to retrieve data without using the equality operator. However, we can use other operators except the equality one in the non-equi join to join multiple tables.

**EQUI JOIN in SQL**

Equi Join in SQL is a type of INNER Join that displays the output by performing a join operation between two or more tables based on the common column between them. It uses the equality ( = ) symbol to compare the data between two columns; if the data matches, it retrieves it. Equi Join compares each column value of the source table with each value in the corresponding target table, and if both the values are equal, it retrieves it.

The Equi Join in SQL returns only the data in all the tables we are comparing based on the common column field. It does not display null or unmatchable data. The equality operator in the Equi Join operation is used to refer to the equality in the WHERE clause. However, it returns the same result when we use the JOIN keyword with the ON clause along with column names and their respective tables.

**Syntax:**

> SELECT \* FROM TableName1, TableName2

WHERE TableName1.ColumnName = TableName2.ColumnName;

**OR**

> SELECT \* FROM TableName1 JOIN TableName2

ON TableName1.ColumnName = TableName2.ColumnName;

**Example:**

Suppose we have two tables, namely **state** and **city**, which contain the name of the states and the name of the cities, respectively. In this example, we will map the cities with the states in which they are present.

The table *state* is shown below:

| **State\_ID** | **State\_Name** |
| --- | --- |
| 1 | Uttar Pradesh |
| 2 | Uttarakhand |
| 3 | Madhya Pradesh |

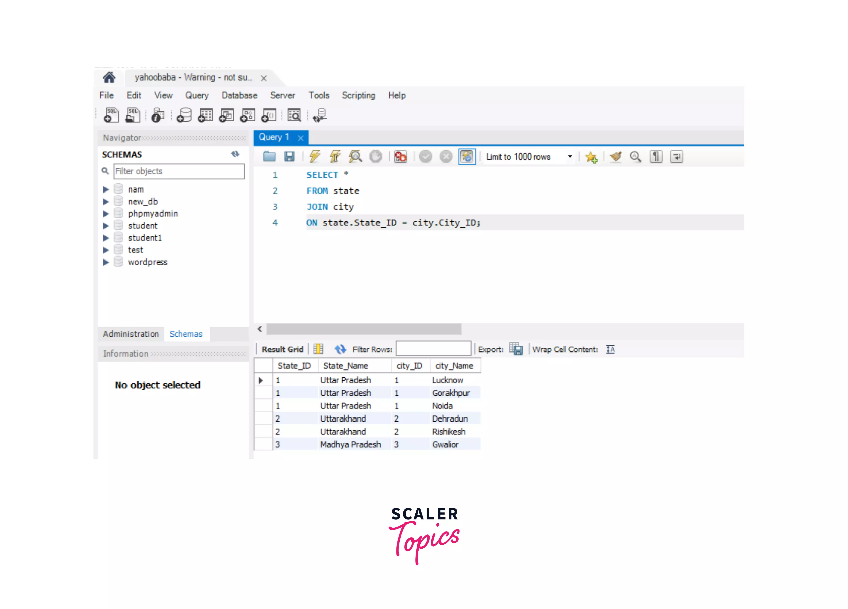
The table *city* is shown below:

| **City\_ID** | **City\_Name** |
| --- | --- |
| 1 | Lucknow |
| 1 | Gorakhpur |
| 1 | Noida |
| 2 | Dehradun |
| 2 | Rishikesh |
| 3 | Gwalior |

Now, if we execute a query of Equi-join using the equality operation and the WHERE clause, then-

SELECT \* FROM state, city WHERE state.State\_Id = city.City\_Id;

**Output:**



We can try the above example with the second syntax-

SELECT \* FROM state

JOIN city

ON state.State\_ID = city.City\_ID;

Therefore, we can retrieve the data from two tables using these two methods.