**SQL: Joins**

Learn 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 joined in a SQL statement.

There are 4 different types of SQL joins:

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

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

**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 SQL **INNER JOIN** is:

SELECT columns

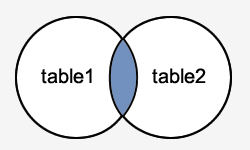
FROM table1

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**

Here is an example of a SQL INNER JOIN:

SELECT s.supplier\_id, s.supplier\_name, od.order\_date

FROM suppliers AS s

INNER JOIN order\_details AS od

ON s.supplier\_id = od.supplier\_id;

This SQL INNER JOIN example would return all rows from the suppliers and orders tables where there is a matching supplier\_id value in both the suppliers and orders tables.

Let's look at some data to explain how the INNER JOINS work:

We have a table called *suppliers* with two fields (supplier\_id and supplier\_ name). It contains the following data:

|  |  |
| --- | --- |
| **supplier\_id** | **supplier\_name** |
| 10000 | IBM |
| 10001 | Hewlett Packard |
| 10002 | Microsoft |
| 10003 | NVIDIA |

We have another table called *orders* with three fields (order\_id, supplier\_id, and order\_date). It contains the following data:

|  |  |  |
| --- | --- | --- |
| **order\_id** | **supplier\_id** | **order\_date** |
| 500125 | 10000 | 2003/05/12 |
| 500126 | 10001 | 2003/05/13 |
| 500127 | 10004 | 2003/05/14 |

If we run the SQL statement (that contains an INNER JOIN) below:

SELECT suppliers.supplier\_id, suppliers.supplier\_name, orders.order\_date

FROM suppliers

INNER JOIN orders

ON suppliers.supplier\_id = orders.supplier\_id;

Our result set would look like this:

|  |  |  |
| --- | --- | --- |
| **supplier\_id** | **name** | **order\_date** |
| 10000 | IBM | 2003/05/12 |
| 10001 | Hewlett Packard | 2003/05/13 |

The rows for *Microsoft* and *NVIDIA* from the supplier table would be omitted, since the supplier\_id's 10002 and 10003 do not exist in both tables. The row for 500127 (order\_id) from the orders table would be omitted, since the supplier\_id 10004 does not exist in the suppliers table.

**Old Syntax**

As a final note, it is worth mentioning that the SQL 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 suppliers.supplier\_id, suppliers.supplier\_name, orders.order\_date

FROM suppliers, orders

WHERE suppliers.supplier\_id = orders.supplier\_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 SQL **LEFT OUTER JOIN** is:

SELECT columns

FROM table1

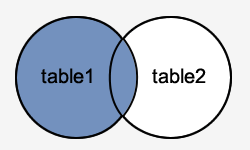
LEFT [OUTER] JOIN table2

ON table1.column = table2.column;

In some databases, the LEFT OUTER JOIN keywords are replaced with 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**

Here is an example of a SQL LEFT OUTER JOIN:

SELECT suppliers.supplier\_id, suppliers.supplier\_name, orders.order\_date

FROM suppliers

LEFT OUTER JOIN orders

ON suppliers.supplier\_id = orders.supplier\_id;

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

If a supplier\_id value in the suppliers table does not exist in the orders table, all fields in the orders table will display as <null*>* in the result set.

Let's look at some data to explain how LEFT OUTER JOINS work:

We have a table called **suppliers** with two fields (supplier\_id and name). It contains the following data:

|  |  |
| --- | --- |
| **supplier\_id** | **supplier\_name** |
| 10000 | IBM |
| 10001 | Hewlett Packard |
| 10002 | Microsoft |
| 10003 | NVIDIA |

We have a second table called **orders** with three fields (order\_id, supplier\_id, and order\_date). It contains the following data:

|  |  |  |
| --- | --- | --- |
| **order\_id** | **supplier\_id** | **order\_date** |
| 500125 | 10000 | 2003/05/12 |
| 500126 | 10001 | 2003/05/13 |

If we run the SQL statement (that contains a LEFT OUTER JOIN) below:

SELECT suppliers.supplier\_id, suppliers.supplier\_name, orders.order\_date

FROM suppliers

LEFT OUTER JOIN orders

ON suppliers.supplier\_id = orders.supplier\_id;

Our result set would look like this:

|  |  |  |
| --- | --- | --- |
| **supplier\_id** | **supplier\_name** | **order\_date** |
| 10000 | IBM | 2003/05/12 |
| 10001 | Hewlett Packard | 2003/05/13 |
| 10002 | Microsoft | <null> |
| 10003 | NVIDIA | <null> |

The rows for *Microsoft* and *NVIDIA* would be included because a LEFT OUTER JOIN was used. However, you will notice that the order\_date field for those records contains a <null> value.

**Old Syntax**

As a final note, it is worth mentioning that the LEFT OUTER JOIN example above could be rewritten using the older implicit syntax that utilizes the outer join operator (+) as follows (but we still recommend using the LEFT OUTER JOIN keyword syntax):

SELECT suppliers.supplier\_id, suppliers.supplier\_name, orders.order\_date

FROM suppliers, orders

WHERE suppliers.supplier\_id = orders.supplier\_id(+);

**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 SQL **RIGHT OUTER JOIN** is:

SELECT columns

FROM table1

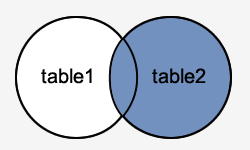
RIGHT [OUTER] JOIN table2

ON table1.column = table2.column;

In some databases, the RIGHT OUTER JOIN keywords are replaced with 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**

Here is an example of a SQL RIGHT OUTER JOIN:

SELECT orders.order\_id, orders.order\_date, suppliers.supplier\_name

FROM suppliers

RIGHT OUTER JOIN orders

ON suppliers.supplier\_id = orders.supplier\_id;

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

If a supplier\_id value in the orders table does not exist in the suppliers table, all fields in the suppliers table will display as <null*>* in the result set.

Let's look at some data to explain how RIGHT OUTER JOINS work:

We have a table called **suppliers** with two fields (supplier\_id and name). It contains the following data:

|  |  |
| --- | --- |
| **supplier\_id** | **supplier\_name** |
| 10000 | Apple |
| 10001 | Google |

We have a second table called **orders** with three fields (order\_id, supplier\_id, and order\_date). It contains the following data:

|  |  |  |
| --- | --- | --- |
| **order\_id** | **supplier\_id** | **order\_date** |
| 500125 | 10000 | 2013/08/12 |
| 500126 | 10001 | 2013/08/13 |
| 500127 | 10002 | 2013/08/14 |

If we run the SQL statement (that contains a RIGHT OUTER JOIN) below:

SELECT orders.order\_id, orders.order\_date, suppliers.supplier\_name

FROM suppliers

RIGHT OUTER JOIN orders

ON suppliers.supplier\_id = orders.supplier\_id;

Our result set would look like this:

|  |  |  |
| --- | --- | --- |
| **order\_id** | **order\_date** | **supplier\_name** |
| 500125 | 2013/08/12 | Apple |
| 500126 | 2013/08/13 | Google |
| 500127 | 2013/08/14 | <null> |

The row for *500127* (order\_id) would be included because a RIGHT OUTER JOIN was used. However, you will notice that the supplier\_name field for that record contains a <null> value.

**Old Syntax**

As a final note, it is worth mentioning that the RIGHT OUTER JOIN example above could be rewritten using the older implicit syntax that utilizes the outer join operator (+) as follows (but we still recommend using the RIGHT OUTER JOIN keyword syntax):

SELECT orders.order\_id, orders.order\_date, suppliers.supplier\_name

FROM suppliers, orders

WHERE suppliers.supplier\_id(+) = orders.supplier\_id;

**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 nulls in place where the join condition is not met.

**Syntax**

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

SELECT columns

FROM table1

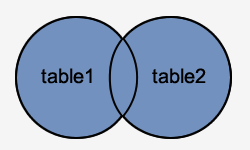
FULL [OUTER] JOIN table2

ON table1.column = table2.column;

In some databases, the FULL OUTER JOIN keywords are replaced with 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**

Here is an example of a SQL FULL OUTER JOIN:

SELECT suppliers.supplier\_id, suppliers.supplier\_name, orders.order\_date

FROM suppliers

FULL OUTER JOIN orders

ON suppliers.supplier\_id = orders.supplier\_id;

This FULL OUTER JOIN example would return all rows from the suppliers table and all rows from the orders table and whenever the join condition is not met, <nulls> would be extended to those fields in the result set.

If a supplier\_id value in the suppliers table does not exist in the orders table, all fields in the orders table will display as <null*>* in the result set. If a supplier\_id value in the orders table does not exist in the suppliers table, all fields in the suppliers table will display as <null> in the result set.

Let's look at some data to explain how FULL OUTER JOINS work:

We have a table called **suppliers** with two fields (supplier\_id and name). It contains the following data:

|  |  |
| --- | --- |
| **supplier\_id** | **supplier\_name** |
| 10000 | IBM |
| 10001 | Hewlett Packard |
| 10002 | Microsoft |
| 10003 | NVIDIA |

We have a second table called **orders** with three fields (order\_id, supplier\_id, and order\_date). It contains the following data:

|  |  |  |
| --- | --- | --- |
| **order\_id** | **supplier\_id** | **order\_date** |
| 500125 | 10000 | 2013/08/12 |
| 500126 | 10001 | 2013/08/13 |
| 500127 | 10004 | 2013/08/14 |

If we run the SQL statement (that contains a FULL OUTER JOIN) below:

SELECT suppliers.supplier\_id, suppliers.supplier\_name, orders.order\_date

FROM suppliers

FULL OUTER JOIN orders

ON suppliers.supplier\_id = orders.supplier\_id;

Our result set would look like this:

|  |  |  |
| --- | --- | --- |
| **supplier\_id** | **supplier\_name** | **order\_date** |
| 10000 | IBM | 2013/08/12 |
| 10001 | Hewlett Packard | 2013/08/13 |
| 10002 | Microsoft | <null> |
| 10003 | NVIDIA | <null> |
| <null> | <null> | 2013/08/14 |

The rows for *Microsoft* and *NVIDIA* would be included because a FULL OUTER JOIN was used. However, you will notice that the order\_date field for those records contains a <null> value.

The row for supplier\_id 10004 would be also included because a FULL OUTER JOIN was used. However, you will notice that the supplier\_id and supplier\_name field for those records contain a <null> value.

**Old Syntax**

As a final note, it is worth mentioning that the FULL OUTER JOIN example above could not have been written in the old syntax without using a [**UNION query**](http://www.techonthenet.com/sql/union.php).