totn Oracle / PLSQL

# Oracle / PLSQL: Joins

This Oracle tutorial explains how to use **JOINS** (inner and outer) in Oracle with syntax, visual illustrations, and examples.

## Description

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

There are 4 different types of Oracle joins:

- Oracle INNER JOIN (or sometimes called simple join)
- Oracle LEFT OUTER JOIN (or sometimes called LEFT JOIN)
- Oracle RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)
- Oracle FULL OUTER JOIN (or sometimes called FULL JOIN)

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

## INNER JOIN (simple join)

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

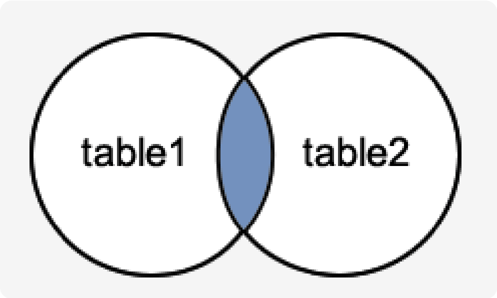
### Syntax

The syntax for the INNER JOIN in Oracle/PLSQL is:

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

### Visual Illustration

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



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

## Example

Here is an example of an Oracle INNER JOIN:

```
SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers
INNER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
```

This Oracle 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 Oracle SELECT 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 Oracle 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;
```

## LEFT OUTER JOIN

Another type of join is called an Oracle 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

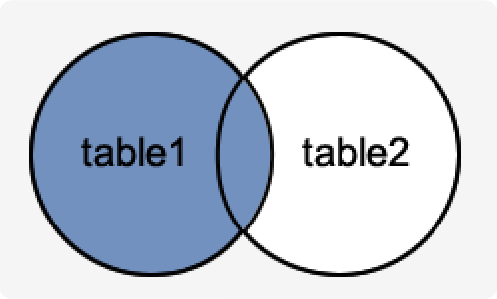
The syntax for the Oracle **LEFT OUTER JOIN** is:

```
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 Oracle LEFT OUTER JOIN returns the shaded area:



The Oracle 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 an Oracle 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 supplier_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 |
|---|---|---|

| order_id | supplier_id | order_date |
|---|---|---|
| 500125 | 10000 | 2003/05/12 |
| 500126 | 10001 | 2003/05/13 |

If we run the SELECT 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(+);
```

## RIGHT OUTER JOIN

Another type of join is called an Oracle 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

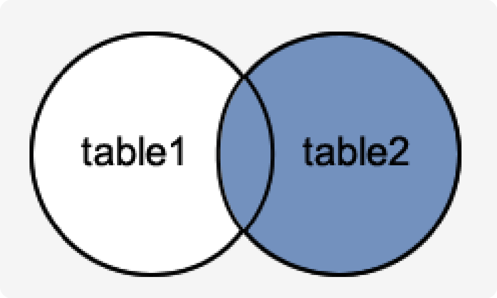
The syntax for the Oracle **RIGHT OUTER JOIN** is:

```
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 Oracle RIGHT OUTER JOIN returns the shaded area:



The Oracle 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 an Oracle 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 supplier_name). It contains the following data:

| supplier_id | supplier_name |
|---|---|
| 10000 | Apple |

| supplier_id | supplier_name |
|---|---|
| 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 SELECT 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;
```

## FULL OUTER JOIN

Another type of join is called an Oracle 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

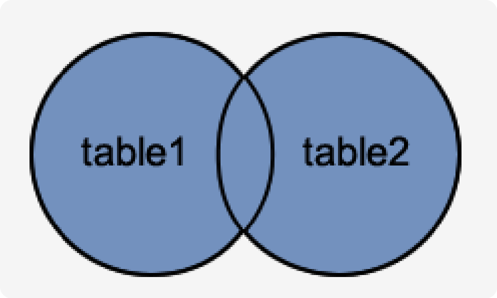
The syntax for the Oracle **FULL OUTER JOIN** is:

```
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 Oracle FULL OUTER JOIN returns the shaded area:



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

## Example

Here is an example of an Oracle 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 supplier_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 SELECT 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.