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)

**INNER JOIN (simple join)**

It is the most common type of join. Oracle INNER JOINS return all rows from multiple tables where the join condition is met.

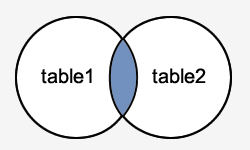
SELECT columns

FROM table1

INNER JOIN table2

ON table1.column = table2.column;

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.

SELECT S.SUPPLIER\_ID, S.SUPPLIER\_NAME, O.ORDER\_DATE

FROM PI\_SUPPLIERS S

INNER JOIN PI\_ORDERS O

ON S.SUPPLIER\_ID = O.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).

**PI\_SUPPLIER Table:**

|  |  |
| --- | --- |
| **SUPPLIER\_ID** | **SUPPLIER\_NAME** |
| 1000 | IBM |
| 1001 | Hewlett Packard |
| 1002 | Microsoft |
| 1003 | NVIDIA |

We have another table called *orders* with three fields (order\_id, supplier\_id, and order\_date).

**PI\_ORDERS Table:**

|  |  |  |
| --- | --- | --- |
| **ORDER\_ID** | **SUPPLIER\_ID** | **ORDER\_DATE** |
| 5011 | 1000 | 1-Jan-21 |
| 5012 | 1001 | 2-Jan-21 |
| 5013 | 1004 | 3-Jan-21 |

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

SELECT S.SUPPLIER\_ID, S.SUPPLIER\_NAME, O.ORDER\_DATE

FROM PI\_SUPPLIERS S

INNER JOIN PI\_ORDERS O

ON S.SUPPLIER\_ID = O.SUPPLIER\_ID;

Our result set would look like this:

|  |  |  |
| --- | --- | --- |
| **SUPPLIER\_ID** | **SUPPLIER\_NAME** | **ORDER\_DATE** |
| 1000 | IBM | 1/JAN/2021 |
| 1001 | Hewlett Packard | 2/JAN/2021 |

The rows for *Microsoft* and *NVIDIA* from the supplier table would be omitted, since the supplier\_id's 1002 and 1003 do not exist in both tables. The row for 5013 (order\_id) from the orders table would be omitted, since the supplier\_id 1004 does not exist in the suppliers table.

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 S.SUPPLIER\_ID, S.SUPPLIER\_NAME, O.ORDER\_DATE

FROM PI\_SUPPLIERS S,PI\_ORDERS O

WHERE S.SUPPLIER\_ID = O.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).

SELECT columns

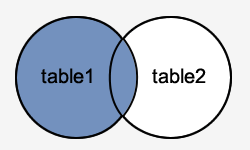
FROM table1

LEFT [OUTER] JOIN table2

ON table1.column = table2.column;

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

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*.

Here is an example of an Oracle LEFT OUTER JOIN:

SELECT S.SUPPLIER\_ID, S.SUPPLIER\_NAME, O.ORDER\_DATE

FROM PI\_SUPPLIERS S

LEFT OUTER JOIN PI\_ORDERS O

ON S.SUPPLIER\_ID = O.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.

**PI\_SUPPLIER Table:**

|  |  |
| --- | --- |
| **SUPPLIER\_ID** | **SUPPLIER\_NAME** |
| 1000 | IBM |
| 1001 | Hewlett Packard |
| 1002 | Microsoft |
| 1003 | NVIDIA |

We have a second table called *orders* with three fields (order\_id, supplier\_id, and order\_date).

**PI\_ORDERS Table:**

|  |  |  |
| --- | --- | --- |
| **ORDER\_ID** | **SUPPLIER\_ID** | **ORDER\_DATE** |
| 5011 | 1000 | 1-Jan-21 |
| 5012 | 1001 | 2-Jan-21 |
| 5013 | 1004 | 3-Jan-21 |

SELECT S.SUPPLIER\_ID, S.SUPPLIER\_NAME, O.ORDER\_DATE

FROM PI\_SUPPLIERS S

LEFT OUTER JOIN PI\_ORDERS O

ON S.SUPPLIER\_ID = O.SUPPLIER\_ID;

|  |  |  |
| --- | --- | --- |
| **SUPPLIER\_ID** | **SUPPLIER\_NAME** | **ORDER\_DATE** |
| 1000 | IBM | 1/JAN/2021 |
| 1001 | Hewlett Packard | 2/JAN/2021 |
| 1002 | Microsoft | NULL |
| 1003 | 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.

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).

SELECT columns

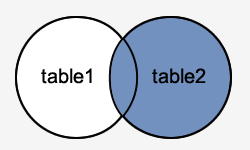
FROM table1

RIGHT [OUTER] JOIN table2

ON table1.column = table2.column;

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

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*.

Here is an example of an Oracle RIGHT OUTER JOIN:

SELECT O.ORDER\_ID, O.ORDER\_DATE, S.SUPPLIER\_NAME

FROM PI\_SUPPLIERS S

RIGHT OUTER JOIN PI\_ORDERS O

ON S.SUPPLIER\_ID = O.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.

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

|  |  |
| --- | --- |
| **SUPPLIER\_ID** | **SUPPLIER\_NAME** |
| 1000 | IBM |
| 1001 | Hewlett Packard |
| 1002 | Microsoft |
| 1003 | 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** |
| 5011 | 1000 | 1-Jan-21 |
| 5012 | 1001 | 2-Jan-21 |
| 5013 | 1004 | 3-Jan-21 |

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

SELECT O.ORDER\_ID, O.ORDER\_DATE, S.SUPPLIER\_NAME

FROM PI\_SUPPLIERS S

RIGHT OUTER JOIN PI\_ORDERS O

ON S.SUPPLIER\_ID = O.SUPPLIER\_ID;

Our result set would look like this:

|  |  |  |  |
| --- | --- | --- | --- |
| **ORDER\_ID** | **ORDER\_DATE** | **SUPPLIER\_NAME** |  |
| 5011 | 1-Jan-21 | IBM |  |
| 5012 | 2-Jan-21 | Hewlett Packard |  |
| 5013 | 3-Jan-21 | <NULL> |  |

The row for *5013* (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.

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.

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

SELECT columns

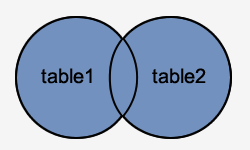
FROM table1

FULL [OUTER] JOIN table2

ON table1.column = table2.column;

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

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*.

Here is an example of an Oracle FULL OUTER JOIN:

SELECT S.SUPPLIER\_ID, S.SUPPLIER\_NAME, O.ORDER\_DATE

FROM PI\_SUPPLIERS S

FULL OUTER JOIN PI\_ORDERS O

ON S.SUPPLIER\_ID = O.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** |
| 1000 | IBM |
| 1001 | Hewlett Packard |
| 1002 | Microsoft |
| 1003 | 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** |
| 5011 | 1000 | 1-Jan-21 |
| 5012 | 1001 | 2-Jan-21 |
| 5013 | 1004 | 3-Jan-21 |

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

SELECT S.SUPPLIER\_ID, S.SUPPLIER\_NAME, O.ORDER\_DATE

FROM PI\_SUPPLIERS S

FULL OUTER JOIN PI\_ORDERS O

ON S.SUPPLIER\_ID = O.SUPPLIER\_ID;

Our result set would look like this:

|  |  |  |
| --- | --- | --- |
| **SUPPLIER\_ID** | **SUPPLIER\_NAME** | **ORDER\_DATE** |
| 1000 | IBM | 1-Jan-21 |
| 1001 | Hewlett Packard | 2-Jan-21 |
| 1002 | Microsoft | <NULL> |
| 1003 | NVIDIA | <NULL> |
| <NULL> | <NULL> | 3-Jan-21 |

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.

**SELF JOIN**

Joining a table to itself is called Self Join

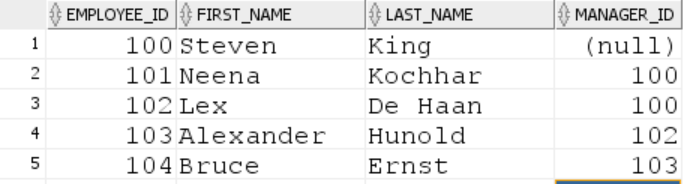
Why ? 🡪 used when a record in a table is related to another record in the same table.

Ex: Employee and Manger

* Employees are in an Employee table and have Managers.
* Managers are also Employees.
* Managers also have Managers.

How ? 🡪 Employee table has a column called Manager\_id…which represents the Employee\_id of their Manager.

*select employee\_id, first\_name, last\_name, manager\_id from employees;*



The **FROM** clause looks like this,

FROM emp A, emp B

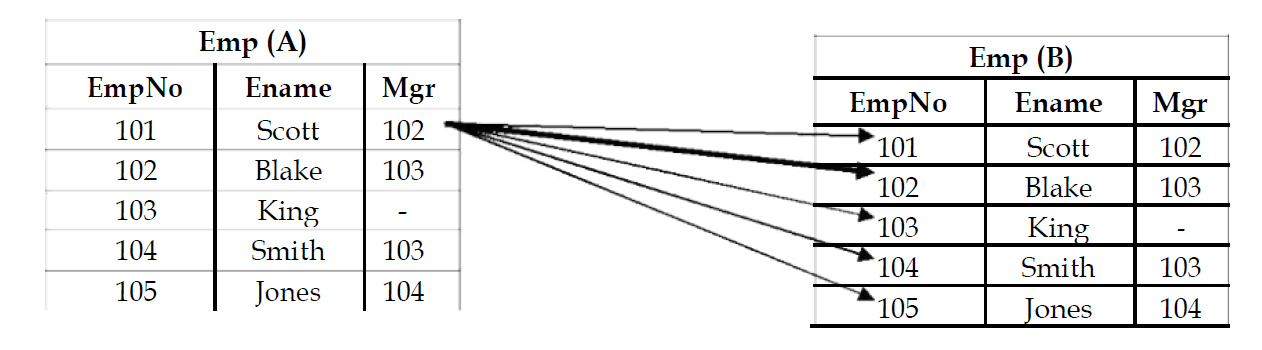
Or

FROM emp A join emp B - *ANSI style*

For ex: **Display employee name along with their manager name**

|  |
| --- |
| **SELECT (E1.FIRST\_NAME||' '||E1.LAST\_NAME) AS EMPLOYEE\_NAME,**  **(E2.FIRST\_NAME||' '||E2.LAST\_NAME) AS MANAGER\_NAME**  **FROM EMPLOYEES E1 INNER JOIN EMPLOYEES E2**  **ON E1.MANAGER\_ID = E2.EMPLOYEE\_ID ;** |

**Now, let us see how this i.e the logic (the above query) works,**



Now, when we RUN the above query – in Oracle :

it starts matching the “**Mgr**” column of **Emp A** with the “**EmpNo**” of E**mp B** –

we get two tables because, in **self join** – a duplicate of the table required is created.

Now let us consider the **first employee Scott** – it starts the **mgrid** of **Scott** with the **EmpNo** of all the records in **Emp B** – when two **ids** match, then the E**mpNo** in **Emp B** becomes the **Mgr** of the **EmpNo** in **Emp A**. Thus, we can see that – **Mgr id:**102 of Scott is matching with **EmpNo:**102 of **Blake** in **Emp B**. Therefore, Blake is the Manager of Scott.

Similarly we do the same for all the other records of **Emp A** and thus find the employees and their respective managers.

**Display the employees who are getting the same salary**

|  |
| --- |
| **SELECT (E1.FIRST\_NAME||' '||E1.LAST\_NAME) AS EMPLOYEE\_NAME,**  **E1.SALARY**  **FROM EMPLOYEES E1 INNER JOIN EMPLOYEES E2**  **ON E1.SALARY = E2.SALARY**  **AND E1.EMPLOYEE\_ID <> E2.EMPLOYEE\_ID;** |
|  |