**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

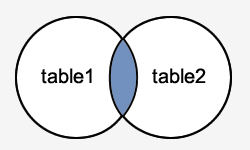
FROM table1

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

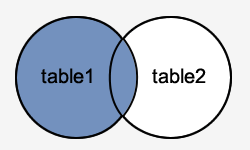
LEFT [OUTER] JOIN table2

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 |
| 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

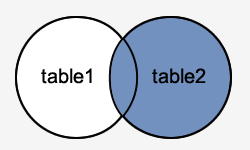
RIGHT [OUTER] JOIN table2

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 |
| 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

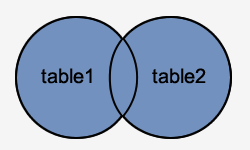
FULL [OUTER] JOIN table2

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.

**Joins**

**EQUI JOIN**

A join which contains an equal to ‘=’ operator in the joins condition.

**NON-EQUI JOIN**

 A join which contains an operator other than equal to ‘=’ in the joins condition.

**SELF JOIN**

Joining the table itself is called self join.

**NATURAL JOIN**

Natural join compares all the common columns

### JOINS IN ORACLE-different joins in oracle with example

1. The purpose of a join is to combine the data across tables.

2. A join is actually performed by the where clause which combines the specified rows of tables.

3. If a join involves in more than two tables then [Oracle](http://dwhlaureate.blogspot.in/2014/05/oracle-database-12c-new-feature-fetch.html) joins first two tables based on the joins condition and then compares the result with the next table and so on.

Assume that we have the following tables.

SQL> select \* from dept;

|  |  |  |
| --- | --- | --- |
| **DEPTNO** | **DNAME** | **LOC** |
| 10 | INVENTORY | HYBD |
| 20 | FINANCE | BGLR |
| 30 | HR | MUMBAI |

SQL> select \* from emp;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **MGR** | **DEPTNO** |
| 111 | saketh | analyst | 444 | 10 |
| 222 | sudha | clerk | 333 | 20 |
| 333 | jagan | manager | 111 | 10 |
| 444 | madhu | engineer | 222 | 40 |

**1.**      **EQUI JOIN**

A join which contains an equal to ‘=’ operator in the joins condition.

*Ex:*

*SQL>*select empno,ename,job,dname,loc from emp e,dept d where e.deptno=d.deptno;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 333 | jagan | manager | INVENTORY | HYBD |
| 222 | sudha | clerk | FINANCE | BGLR |

Using clause

SQL> select empno,ename,job ,dname,loc from emp e join dept d using(deptno);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 333 | jagan | manager | INVENTORY | HYBD |
| 222 | sudha | clerk | FINANCE | BGLR |

On clause

SQL> select empno,ename,job,dname,loc from emp e join dept d on(e.deptno=d.deptno);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 333 | jagan | manager | INVENTORY | HYBD |
| 222 | sudha | clerk | FINANCE | BGLR |

**2.**      **NON-EQUI JOIN**

 A join which contains an operator other than equal to ‘=’ in the joins condition.

 Ex:

SQL> select empno,ename,job,dname,loc from emp e,dept d where e.deptno > d.deptno;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 222 | sudha | clerk | INVENTORY | HYBD |
| 444 | madhu | engineer | INVENTORY | HYBD |
| 444 | madhu | engineer | FINANCE | BGLR |
| 444 | madhu | engineer | HR | MUMBAI |

**3.**      **SELF JOIN**

Joining the table itself is called self join.

Ex:

SQL> select e1.empno,e2.ename,e1.job,e2.deptno from emp e1,emp e2 where e1.empno=e2.mgr;

|  |  |  |  |
| --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DEPTNO** |
| 111 | jagan | analyst | 10 |
| 222 | madhu | clerk | 40 |
| 333 | sudha | manager | 20 |
| 444 | saketh | engineer | 10 |

**4.**      **NATURAL JOIN**

Natural join compares all the common columns.

Ex:

SQL> select empno,ename,job,dname,loc from emp natural join dept;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 333 | jagan | manager | INVENTORY | HYBD |
| 222 | sudha | clerk | FINANCE | BGLR |

**5.**      **CROSS JOIN**

This will gives the cross product.

Ex:

SQL> select empno,ename,job,dname,loc from emp cross join dept;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 222 | sudha | clerk | INVENTORY | HYBD |
| 333 | jagan | manager | INVENTORY | HYBD |
| 444 | madhu | engineer | INVENTORY | HYBD |
| 111 | saketh | analyst | FINANCE | BGLR |
| 222 | sudha | clerk | FINANCE | BGLR |
| 333 | jagan | manager | FINANCE | BGLR |
| 444 | madhu | engineer | FINANCE | BGLR |
| 111 | saketh | analyst | HR | MUMBAI |
| 222 | sudha | clerk | HR | MUMBAI |
| 333 | jagan | manager | HR | MUMBAI |
| 444 | madhu | engineer | HR | MUMBAI |

**6.**      **OUTER JOIN**

Outer join gives the non-matching records along with matching records.

**LEFT OUTER JOIN**

This will display the all matching records and the records which are in left hand side table those that are not in right hand side table.

Ex:

SQL> select empno,ename,job,dname,loc from emp e left outer join dept d

on(e.deptno=d.deptno);

Or

SQL> select empno,ename,job,dname,loc from emp e,dept d where

e.deptno=d.deptno(+);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 333 | jagan | manager | INVENTORY | HYBD |
| 222 | sudha | clerk | FINANCE | BGLR |
| 444 | madhu | engineer |  |  |

**RIGHT OUTER JOIN**

This will display the all matching records and the records which are in right hand side table those that are not in left hand side table.

Ex:

SQL> select empno,ename,job,dname,loc from emp e right outer join dept d

on(e.deptno=d.deptno);

Or

SQL> select empno,ename,job,dname,loc from emp e,dept d where e.deptno(+) =

d.deptno;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 333 | jagan | manager | INVENTORY | HYBD |
| 222 | sudha | clerk | FINANCE | BGLR |
|  |  |  | HR | MUMBAI |

**FULL OUTER JOIN**

This will display the all matching records and the non-matching records from both tables.

Ex:

SQL> select empno,ename,job,dname,loc from emp e full outer join dept d

on(e.deptno=d.deptno);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 333 | jagan | manager | INVENTORY | HYBD |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 222 | sudha | clerk | FINANCE | BGLR |
| 444 | madhu | engineer |  |  |
|  |  |  | HR | MUMBAI |

**7.**      **INNER JOIN**

This will display all the records that have matched.

Ex:

SQL> select empno,ename,job,dname,loc from emp inner join dept using(deptno);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPNO** | **ENAME** | **JOB** | **DNAME** | **LOC** |
| 111 | saketh | analyst | INVENTORY | HYBD |
| 333 | jagan | manager | INVENTORY | HYBD |
| 222 | sudha | clerkx` | FINANCE | BGLR |

<http://dwhlaureate.blogspot.in/2012/08/joins-in-oracle.html>

<http://www.techonthenet.com/oracle/joins.php>

**Subquery Vs Joins**

**What Makes A Subquery**

* A subquery is a "SELECT" statement within the "WHERE" or "HAVING" clause of an outer "INSERT," "UPDATE," "MERGE" or "DELETE" statement. The subquery also is called the "inner query" or "inner select." The subquery may have a "FROM" clause with one or more tables and may optionally have a "WHERE," "GROUP BY" or "HAVING" clause. It is always enclosed in parenthesis.

**Advantages and Disadvantages of Subqueries**

* Subqueries are advantageous because they structure the query to isolate each part of the statement, perform the same operation that would ordinarily require complex joins and unions and are easier to read. Subqueries even were the basis for the name "Structured Query Language" (SQL) because of their easily readable structure. A disadvantage is that you cannot modify a table and select from the same table within a subquery in the same SQL statement. Subqueries also can take longer to execute than a join because of how the database optimizer processes them.

**What Makes A Join**

* Most subqueries can be rewritten as joins, and most joins can be rewritten as subqueries. A join defines two or more tables by a related column. Tables usually are joined on primary and foreign keys. For example, an employee table might have a primary key of an employee id column, while a timesheet table also has an employee id column that is a foreign key to the employee table. The SQL join can be written as "WHERE employee.empid = timesheet.empid" or "FROM employee JOIN timesheet ON (employee.empid = timesheet.empid)."

**Advantages and Disadvantages of Joins**

* The main advantage of a join is that it executes faster. The performance increase might not be noticeable by the end user. However, because the columns are specifically named and indexed and optimized by the database engine, the retrieval time almost always will be faster than that of a subquery. There are also inner and outer joins, left and right joins, full joins and cross joins. A disadvantage of using joins is that they are not as easy to read as subqueries. Another disadvantage is that it can be confusing as to which join is the appropriate type of join to use to yield the correct desired result set.

<http://www.ehow.com/info_8700587_advantages-disadvantages-join-subquery.html>

*In JOINs RDBMS can create an execution plan that is better for your query and can predict what data should be loaded to be processed and save time, unlike the sub-query where it will run all the queries and load all their data to do the processing.*

*The good thing in sub-queries is that they are more readable than JOINs: that’s why most new SQL people prefer them; it is the easy way; but when it comes to performance, JOINS are better in most cases even though they are not hard to read too.*

*Sub-queries are the logically correct way to solve problems of the form, “Get facts from A, conditional on facts from B”. In such instances, it makes more logical sense to stick B in a sub-query than to do a join. It is also safer, in a practical sense, since you don’t have to be cautious about getting duplicated facts from A due to multiple matches against B.*

*Practically speaking, however, the answer usually comes down to performance. Some optimisers suck lemons when given a join vs a sub-query, and some suck lemons the other way, and this is optimiser-specific, DBMS-version-specific and query-specific.*

*Historically, explicit joins usually win, hence the established wisdom that joins are better, but optimisers are getting better all the time, and so I prefer to write queries first in a logically coherent way, and then restructure if performance constraints warrant this.*

<http://www.chrislondon.co/joins-vs-subqueries/>