

# SQL - CARTESIAN or CROSS JOINS

The CARTESIAN JOIN or CROSS JOIN returns the Cartesian product of the sets of records from two or more joined tables. Thus, it equates to an inner join where the join-condition always evaluates to either True or where the join-condition is absent from the statement.

## Syntax

The basic syntax of the **CARTESIAN JOIN** or the **CROSS JOIN** is as follows –

```
SELECT table1.column1, table2.column2...  
FROM table1, table2 [, table3 ]
```

## Example

Consider the following two tables.

**Table 1** – CUSTOMERS table is as follows.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

7	Muffy	24	Indore	10000.00
---	-------	----	--------	----------

Table 2: ORDERS Table is as follows –

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Now, let us join these two tables using CARTESIAN JOIN as follows –

```
SQL> SELECT ID, NAME, AMOUNT, DATE
      FROM CUSTOMERS, ORDERS;
```

This would produce the following result –

ID	NAME	AMOUNT	DATE
1	Ramesh	3000	2009-10-08 00:00:00
1	Ramesh	1500	2009-10-08 00:00:00
1	Ramesh	1560	2009-11-20 00:00:00
1	Ramesh	2060	2008-05-20 00:00:00
2	Khilan	3000	2009-10-08 00:00:00
2	Khilan	1500	2009-10-08 00:00:00

	2		Khilan		1560		2009-11-20 00:00:00	
	2		Khilan		2060		2008-05-20 00:00:00	
	3		kaushik		3000		2009-10-08 00:00:00	
	3		kaushik		1500		2009-10-08 00:00:00	
	3		kaushik		1560		2009-11-20 00:00:00	
	3		kaushik		2060		2008-05-20 00:00:00	
	4		Chaitali		3000		2009-10-08 00:00:00	
	4		Chaitali		1500		2009-10-08 00:00:00	
	4		Chaitali		1560		2009-11-20 00:00:00	
	4		Chaitali		2060		2008-05-20 00:00:00	
	5		Hardik		3000		2009-10-08 00:00:00	
	5		Hardik		1500		2009-10-08 00:00:00	
	5		Hardik		1560		2009-11-20 00:00:00	
	5		Hardik		2060		2008-05-20 00:00:00	
	6		Komal		3000		2009-10-08 00:00:00	
	6		Komal		1500		2009-10-08 00:00:00	
	6		Komal		1560		2009-11-20 00:00:00	
	6		Komal		2060		2008-05-20 00:00:00	
	7		Muffy		3000		2009-10-08 00:00:00	
	7		Muffy		1500		2009-10-08 00:00:00	
	7		Muffy		1560		2009-11-20 00:00:00	
	7		Muffy		2060		2008-05-20 00:00:00	
+-----+-----+-----+-----+-----+-----+-----+-----+-----+								

# SQL - Using Joins

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Consider the following two tables –

**Table 1 – CUSTOMERS Table**

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

**Table 2 – ORDERS Table**

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500

101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060
+-----+-----+-----+-----+			

Now, let us join these two tables in our SELECT statement as shown below.

```
SQL> SELECT ID, NAME, AGE, AMOUNT
      FROM CUSTOMERS, ORDERS
      WHERE CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result.

+-----+-----+-----+-----+				
ID	NAME	AGE	AMOUNT	
+-----+-----+-----+-----+				
3	kaushik	23	3000	
3	kaushik	23	1500	
2	Khilan	25	1560	
4	Chaitali	25	2060	
+-----+-----+-----+-----+				

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal to symbol.

There are different types of joins available in SQL –

- **INNER JOIN** – returns rows when there is a match in both tables.
- **LEFT JOIN** – returns all rows from the left table, even if there are no matches in the right table.

- **RIGHT JOIN** – returns all rows from the right table, even if there are no matches in the left table.
- **FULL JOIN** – returns rows when there is a match in one of the tables.
- **SELF JOIN** – is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- **CARTESIAN JOIN** – returns the Cartesian product of the sets of records from the two or more joined tables.

Let us now discuss each of these joins in detail.

---

 **Print Page**

---

# SQL - UNIONS CLAUSE

The SQL UNION clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.

To use this UNION clause, each SELECT statement must have

- The same number of columns selected
- The same number of column expressions
- The same data type and
- Have them in the same order

But they need not have to be in the same length.

## Syntax

The basic syntax of a **UNION** clause is as follows –

```
SELECT column1 [, column2 ]  
FROM table1 [, table2 ]  
[WHERE condition]
```

UNION

```
SELECT column1 [, column2 ]  
FROM table1 [, table2 ]  
[WHERE condition]
```

Here, the given condition could be any given expression based on your requirement.

## Example

Consider the following two tables.

**Table 1** – CUSTOMERS Table is as follows.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

**Table 2** – ORDERS Table is as follows.

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Now, let us join these two tables in our SELECT statement as follows –



```
SQL> SELECT ID, NAME, AMOUNT, DATE
      FROM CUSTOMERS
      LEFT JOIN ORDERS
      ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
UNION
      SELECT ID, NAME, AMOUNT, DATE
      FROM CUSTOMERS
      RIGHT JOIN ORDERS
      ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result –

ID	NAME	AMOUNT	DATE
1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL

## The UNION ALL Clause

The UNION ALL operator is used to combine the results of two SELECT statements including duplicate rows.

The same rules that apply to the UNION clause will apply to the UNION ALL operator.

## Syntax

The basic syntax of the **UNION ALL** is as follows.

```
SELECT column1 [, column2 ]  
FROM table1 [, table2 ]  
[WHERE condition]
```

UNION ALL

```
SELECT column1 [, column2 ]  
FROM table1 [, table2 ]  
[WHERE condition]
```

Here, the given condition could be any given expression based on your requirement.

## Example

Consider the following two tables,

**Table 1** – CUSTOMERS Table is as follows.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

7	Muffy	24	Indore	10000.00
---	-------	----	--------	----------

**Table 2** – ORDERS table is as follows.

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Now, let us join these two tables in our SELECT statement as follows –

```
SQL> SELECT ID, NAME, AMOUNT, DATE
      FROM CUSTOMERS
      LEFT JOIN ORDERS
      ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
UNION ALL
      SELECT ID, NAME, AMOUNT, DATE
      FROM CUSTOMERS
      RIGHT JOIN ORDERS
      ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result –

ID	NAME	AMOUNT	DATE
----	------	--------	------

1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00

There are two other clauses (i.e., operators), which are like the UNION clause.

- **SQL INTERSECT Clause** – This is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement.
- **SQL EXCEPT Clause** – This combines two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement.

# SQL - Group By

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

## Syntax

The basic syntax of a GROUP BY clause is shown in the following code block. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

## Example

Consider the CUSTOMERS table is having the following records –

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00

4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

If you want to know the total amount of the salary on each customer, then the GROUP BY query would be as follows.

```
SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS
      GROUP BY NAME;
```

This would produce the following result –

NAME	SUM(SALARY)
Chaitali	6500.00
Hardik	8500.00
kaushik	2000.00
Khilan	1500.00
Komal	4500.00
Muffy	10000.00
Ramesh	2000.00

Now, let us look at a table where the CUSTOMERS table has the following records with duplicate names –

ID	NAME	AGE	ADDRESS	SALARY
----	------	-----	---------	--------

1	Ramesh	32	Ahmedabad	2000.00
2	Ramesh	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	kaushik	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Now again, if you want to know the total amount of salary on each customer, then the GROUP BY query would be as follows –

```
SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS
      GROUP BY NAME;
```

This would produce the following result –

NAME	SUM(SALARY)
Hardik	8500.00
kaushik	8500.00
Komal	4500.00
Muffy	10000.00
Ramesh	3500.00

# SQL - Having Clause

The **HAVING Clause** enables you to specify conditions that filter which group results appear in the results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

## Syntax

The following code block shows the position of the HAVING Clause in a query.

```
SELECT  
FROM  
WHERE  
GROUP BY  
HAVING  
ORDER BY
```

The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used. The following code block has the syntax of the SELECT statement including the HAVING clause –

```
SELECT column1, column2  
FROM table1, table2  
WHERE [ conditions ]  
GROUP BY column1, column2  
HAVING [ conditions ]
```



ORDER BY column1, column2

## Example

Consider the CUSTOMERS table having the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an example, which would display a record for a similar age count that would be more than or equal to 2.

```
SQL > SELECT ID, NAME, AGE, ADDRESS, SALARY
FROM CUSTOMERS
GROUP BY age
HAVING COUNT(age) >= 2;
```

This would produce the following result –

ID	NAME	AGE	ADDRESS	SALARY
----	------	-----	---------	--------

+	-----	+	-----	+	-----	+	-----	+		
	2		Khilan		25		Delhi		1500.00	
+	-----	+	-----	+	-----	+	-----	+		

---

 **Print Page**

---