INNER, LEFT, RIGHT, FULL JOINS and Cartesian Product

A Cartesian product is a join that has no join criteria, as in SELECT \* FROM A,B or SELECT \* FROM A INNER JOIN B. The result of this is a set that has *cardinality(A) \* cardinality(B)* rows.

Example:

SELECT \* FROM sailor, reverse;

Cardinality of sailor = 10

Cardinality of reverse = 10

Sailor table:

|  |  |  |  |
| --- | --- | --- | --- |
| Sid | sname | Rating | Age |
| 22 | Dustin | 7 | 45 |
| 29 | Brutus | 1 | 33 |
| 31 | Lubber | 8 | 55 |
| 32 | Andy | 8 | 25 |
| 58 | Rusty | 10 | 35 |
| 64 | Horatio | 7 | 35 |
| 71 | Zorba | 10 | 16 |
| 74 | Horatio | 9 | 35 |
| 85 | Art | 3 | 25 |
| 95 | Bob | 3 | 63 |

Reserve table:

|  |  |  |
| --- | --- | --- |
| Sid | Bid | Day |
| 22 | 101 | 1998-10-10 |
| 22 | 102 | 1998-10-10 |
| 22 | 103 | 1998-10-08 |
| 22 | 104 | 1998-10-07 |
| 31 | 102 | 1998-10-11 |
| 31 | 103 | 1998-11-06 |
| 31 | 104 | 1998-11-12 |
| 64 | 101 | 1998-09-05 |
| 64 | 102 | 1998-09-08 |
| 74 | 103 | 1998-09-08 |

After taking the Cartesian Product = 10 x 10 = 100

Cartesian Product:

SELECT \* FROM sailor, reverse;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sid | Sname | Rating | Age | sid | Bid | Day |
| 22 | Dustin | 7 | 45 | 22 | 101 | 1998-10-10 |
| 22 | Dustin | 7 | 45 | 22 | 102 | 1998-10-10 |
| 22 | Dustin | 7 | 45 | 22 | 103 | 1998-10-08 |
| 22 | Dustin | 7 | 45 | 22 | 104 | 1998-10-07 |
| 22 | Dustin | 7 | 45 | 31 | 102 | 1998-10-11 |
| 22 | Dustin | 7 | 45 | 31 | 103 | 1998-11-06 |
| 22 | Dustin | 7 | 45 | 31 | 104 | 1998-11-12 |
| 22 | Dustin | 7 | 45 | 64 | 101 | 1998-09-05 |
| 22 | Dustin | 7 | 45 | 64 | 102 | 1998-09-08 |
| 22 | Dustin | 7 | 45 | 74 | 103 | 1998-09-08 |

To get those sailors who have reserved the boats.

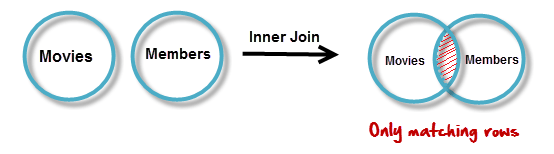
SELECT \* FROM sailor S, reverse R WHERE S.sid = R.sid;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sid | sname | Rating | Age | sid | Bid | Day |
| 22 | Dustin | 7 | 45 | 22 | 101 | 1998-10-10 |
| 22 | Dustin | 7 | 45 | 22 | 102 | 1998-10-10 |
| 22 | Dustin | 7 | 45 | 22 | 103 | 1998-10-08 |
| 22 | Dustin | 7 | 45 | 22 | 104 | 1998-10-07 |
| 31 | Lubber | 8 | 55 | 31 | 102 | 1998-10-11 |
| 31 | Lubber | 8 | 55 | 31 | 103 | 1998-11-06 |
| 31 | Lubber | 8 | 55 | 31 | 104 | 1998-11-12 |
| 64 | Horatio | 7 | 35 | 64 | 101 | 1998-09-05 |
| 64 | Horatio | 7 | 35 | 64 | 102 | 1998-09-08 |
| 95 | Bob | 3 | 63 | 74 | 103 | 1998-09-08 |

### INNER JOIN

The inner JOIN is used to return rows from both tables that satisfy the given condition.

Suppose , you want to get list of members who have rented movies together with titles of movies rented by them. You can simply use an INNER JOIN for that, which returns rows from both tables that satisfy with given conditions.

SELECT list[](https://www.guru99.com/images/InnerJoin.png)

SELECT Attribute\_list FROM table1

INNER JOIN table2 USING(common attribute)

WHERE condition

GROUP BY

SELECT Attribute\_list FROM table1

INNER JOIN table2 ON Table1.attribute = Table2.attribute

WHERE condition

GROUP BY

We can use INNER JOIN statement to get the same result. Before using JOIN statement there should be a common attribute which will join the tables. It means we must have the keys which will make the relations with the tables.

In sailor and reserve tables, (sid) is a common variable which will be use JOIN them.

SELECT \* FROM sailor INNER JOIN reverse; will return same result as in the above statement.

To get those sailors who have reserved the boats.

JOIN with USING statement.

SELECT \* FROM sailor INNER JOIN reverse USING (sid);

JOIN with ON statement.

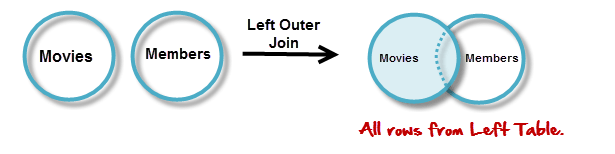
SELECT \* FROM sailor S INNER JOIN reverse R ON S.sid = R.sid;

Both above statements will return the following result which contains (sid) column one time.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| sid | sname | Rating | Age | Bid | Day |
| 22 | Dustin | 7 | 45 | 101 | 1998-10-10 |
| 22 | Dustin | 7 | 45 | 102 | 1998-10-10 |
| 22 | Dustin | 7 | 45 | 103 | 1998-10-08 |
| 22 | Dustin | 7 | 45 | 104 | 1998-10-07 |
| 31 | Lubber | 8 | 55 | 102 | 1998-10-11 |
| 31 | Lubber | 8 | 55 | 103 | 1998-11-06 |
| 31 | Lubber | 8 | 55 | 104 | 1998-11-12 |
| 64 | Horatio | 7 | 35 | 101 | 1998-09-05 |
| 64 | Horatio | 7 | 35 | 102 | 1998-09-08 |
| 95 | Bob | 3 | 63 | 103 | 1998-09-08 |

**LEFT JOIN**

Assume now you want to get titles of all movies together with names of members who have rented them. It is clear that some movies have not being rented by any one. We can simply use **LEFT JOIN**for the purpose.

[](https://www.guru99.com/images/LeftOuterJoin.png)

The LEFT JOIN returns all the rows from the table on the left even if no matching rows have been found in the table on the right. **Where no matches have been found in the table on the right, NULL is returned.**

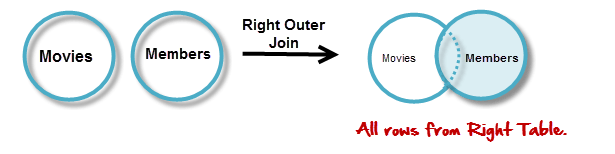
SELECT \* FROM sailor LEFT JOIN reverse USING (sid);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| sid | sname | Rating | Age | Bid | Day |
| 22 | Dustin | 7 | 45 | 101 | 1998-10-10 |
| 22 | Dustin | 7 | 45 | 102 | 1998-10-10 |
| 22 | Dustin | 7 | 45 | 103 | 1998-10-08 |
| 22 | Dustin | 7 | 45 | 104 | 1998-10-07 |
| 29 | Brutus | 1 | 33 | Null | Null |
| 31 | Lubber | 8 | 55 | 102 | 1998-10-11 |
| 31 | Lubber | 8 | 55 | 103 | 1998-11-06 |
| 31 | Lubber | 8 | 55 | 104 | 1998-11-12 |
| 32 | Andy | 8 | 25 | Null | Null |
| 58 | Rusty | 10 | 35 | Null | Null |
| 64 | Horatio | 7 | 35 | 101 | 1998-09-05 |
| 64 | Horatio | 7 | 35 | 102 | 1998-09-08 |
| 71 | Zorba | 10 | 16 | Null | Null |
| 74 | Horatio | 9 | 35 | Null | Null |
| 85 | Art | 3 | 25 | Null | Null |
| 95 | Bob | 3 | 63 | 103 | 1998-09-08 |

### RIGHT JOIN

RIGHT JOIN is obviously the opposite of LEFT JOIN. The RIGHT JOIN returns all the columns from the table on the right even if no matching rows have been found in the table on the left. Where no matches have been found in the table on the left, NULL is returned.

In our example,  let's assume that you need to get names of members and movies rented by them. Now we have a new member who has not rented any movie yet

[](https://www.guru99.com/images/RightOuterJoin.png)

SELECT \* FROM sailor RIGHT JOIN reverse USING (sid);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| sid | Bid | Day | sname | Rating | Age |
| 22 | 101 | 1998-10-10 | Dustin | 7 | 45 |
| 22 | 102 | 1998-10-10 | Dustin | 7 | 45 |
| 22 | 103 | 1998-10-08 | Dustin | 7 | 45 |
| 22 | 104 | 1998-10-07 | Dustin | 7 | 45 |
| 31 | 102 | 1998-10-11 | Lubber | 8 | 55 |
| 31 | 103 | 1998-11-06 | Lubber | 8 | 55 |
| 31 | 104 | 1998-11-12 | Lubber | 8 | 55 |
| 64 | 101 | 1998-09-05 | Horatio | 7 | 35 |
| 64 | 102 | 1998-09-08 | Horatio | 7 | 35 |
| 74 | 103 | 1998-09-08 | Horatio | 9 | 35 |

Query:

Display the names of employees total children

SELECT fname, count(essn) FROM employee INNER JOIN dependent ON essn = ssn GROUP BY (essn);

Query: Display the name of employees working on total number of projects and total hours worked.

SELECT fname, count(pno) AS Projects, sum(hours) AS Hours Worked FROM employee INNER JOIN works\_on ON essn = ssn GROUP BY (essn);

[Home](https://www.mysqltutorial.org/) / [Basic MySQL Tutorial](https://www.mysqltutorial.org/basic-mysql-tutorial.aspx) / MySQL INNER JOIN

# **MySQL INNER JOIN**

**Summary**: in this tutorial, you will learn how to use the MySQL INNER JOIN clause to select data from multiple tables based on join conditions.

## **Introduction to MySQL INNER JOIN clause**

The INNER JOIN matches each row in one table with every row in other tables and allows you to query rows that contain columns from both tables.

The INNER JOIN is an optional clause of the [SELECT](https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx) statement. It appears immediately after the FROM clause. Here is the syntax of the INNER JOIN clause:

**SELECT**

select\_list

**FROM** t1

**INNER** **JOIN** t2 **ON** join\_condition1

**INNER** **JOIN** t3 **ON** join\_condition2

...;

In this syntax:

* First, specify the main table that appears in the FROM clause (t1).
* Second, specify the table that will be joined with the main table, which appears in the INNER JOIN clause (t2, t3,…).
* Third, specify a join condition after the ON keyword of the INNER JOIN clause. The join condition specifies the rule for matching rows between the main table and the table appeared in the INNER JOIN clause.

Assuming that you want to join two tables t1 and t2.

The following statement illustrates how to join two tables t1 and t2 using the INNER JOIN clause:

**SELECT**

select\_list

**FROM**

t1

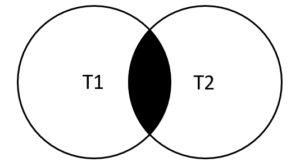
**INNER** **JOIN** t2 **ON** join\_condition;

The INNER JOIN clause compares each row in the t1 table with every row in the t2 table based on the join condition.

If rows from both tables cause the join condition to evaluate to TRUE, the INNER JOIN creates a new row whose columns contain all columns of rows from the tables and includes this new row in the result set. Otherwise, the INNER JOIN just ignores the rows.

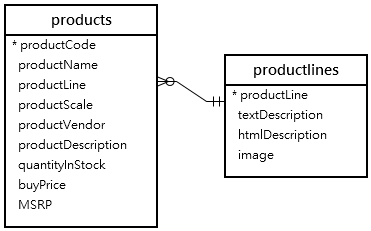
In case no row between tables causes the join condition to evaluate to TRUE, the INNER JOIN returns an empty result set. This logic is also applied when you join more than 2 tables.

The following Venn diagram illustrates how the INNER JOIN clause works:



## **MySQL INNER JOIN examples**

Let’s look at the products and productlines tables in the [sample database](https://www.mysqltutorial.org/mysql-sample-database.aspx).



In this diagram, the table products has the column productLine that references the column  productline of the table productlines . The column productLine in the table products is called the [foreign key](https://www.mysqltutorial.org/mysql-foreign-key/) column.

Typically, you join tables that have foreign key relationships like the  productlines and products tables.

Suppose you want to get:

* The productCode and productName from the products table.
* The textDescription of product lines from the productlines table.

To do this, you need to select data from both tables by matching rows based on values in the productline column using the INNER JOIN clause as follows:

**SELECT**

productCode,

productName,

textDescription

**FROM**

products t1

**INNER** **JOIN** productlines t2

**ON** t1.productline = t2.productline;

[**Try It Out**](https://www.mysqltutorial.org/tryit/query/mysql-inner-join/#1)

### MySQL INNER JOIN - Products Data Example

Because the joined columns of both tables have the same name  productline, you can use the USING syntax:

**SELECT**

productCode,

productName,

textDescription

**FROM**

products

**INNER** **JOIN** productlines **USING** (productline);

**SELECT**

productCode,

productName

**FROM**

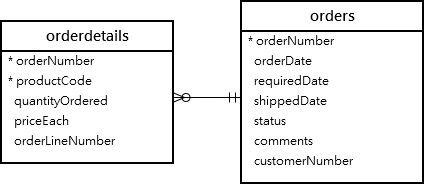
products

LEFT OUTER **JOIN** productlines **USING** (productline) limit 10;

The query returns the same result set. However, the USING syntax is much shorter and cleaner.

### MySQL INNER JOIN with GROUP BY clause example

See the following orders and orderdetails tables:



This query returns order number, order status and total sales from the orders and orderdetails tables using the INNER JOIN clause with the [GROUP BY](https://www.mysqltutorial.org/mysql-group-by.aspx)clause:

**SELECT**

t1.orderNumber,

count(t1.orderNumber),

**SUM**(quantityOrdered \* priceEach) total

**FROM**

orders t1

**INNER** **JOIN** orderdetails t2

**ON** t1.orderNumber = t2.orderNumber

**GROUP** **BY** orderNumber limit 10;

### I would like to know the customers names also be required. So we need to join third table customer.

**SELECT**

t3.customername,

t1.orderNumber,

count(t1.orderNumber),

**SUM**(quantityOrdered \* priceEach) total

**FROM**

orders t1

**INNER** **JOIN** orderdetails t2

**ON** t1.orderNumber = t2.orderNumber

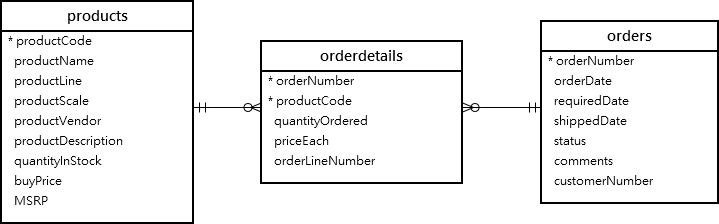
**INNER** **JOIN** customers t3

**ON** t3.customernumber = t1.customernumber

**GROUP** **BY** orderNumber limit 10;

### MySQL INNER JOIN – join three tables example

See the following products, orders and orderdetails tables:



This query uses two INNER JOIN clauses to join three tables: orders, orderdetails, and products:

**SELECT**

orderNumber,

orderDate,

orderLineNumber,

productName,

quantityOrdered,

priceEach

**FROM**

orders

**INNER** **JOIN**

orderdetails **USING** (orderNumber)

**INNER** **JOIN**

products **USING** (productCode)

**ORDER** **BY**

orderNumber,

orderLineNumber limit 10;

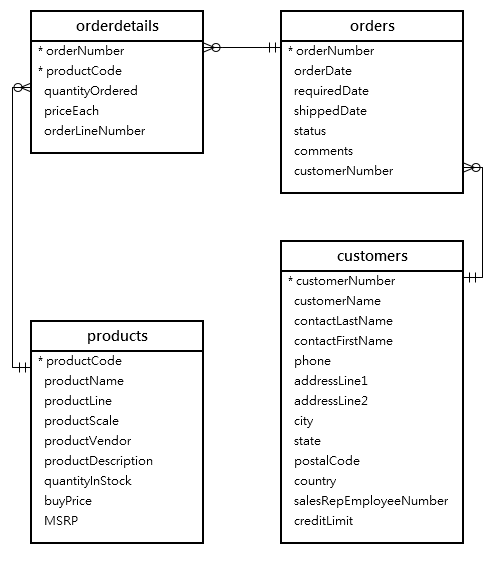
This picture shows the partial output:



### MySQL INNER JOIN – join four tables example

See the following orders, orderdetails, customers and products tables:

**Give the information about each order when it was placed, what item was ordered, how much quantity and price of each item.**



This example uses three INNER JOIN clauses to query data from the four tables above:

**Give the information about each order when it was placed, what item was ordered, how much quantity and price of each item.**

**SELECT**

orderNumber,

orderDate,

customerName,

orderLineNumber,

productName,

quantityOrdered,

priceEach

**FROM**

orders

**INNER** **JOIN** orderdetails

**USING** (orderNumber)

**INNER** **JOIN** products

**USING** (productCode)

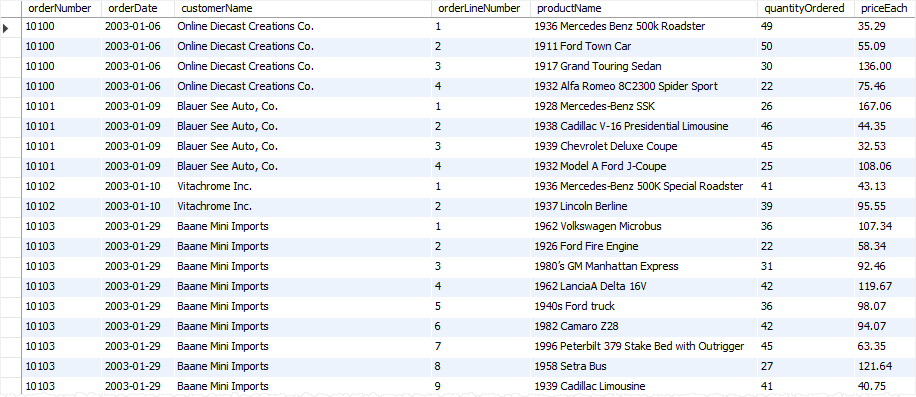
**INNER** **JOIN** customers

**USING** (customerNumber)

**ORDER** **BY**

orderNumber,

orderLineNumber;



## **MySQL INNER JOIN using other operators**

So far, you have seen that the join condition used the equal operator (=) for matching rows.

In addition to the equal operator (=), you can use other operators such as greater than ( >), less than ( <), and not-equal ( <>) operator to form the join condition.

The following query uses a less-than ( <) join to find sales price of the product whose code is S10\_1678 that is less than the manufacturer’s suggested retail price (MSRP) for that product.

**SELECT**

orderNumber,

productName,

msrp,

priceEach

**FROM**

products p

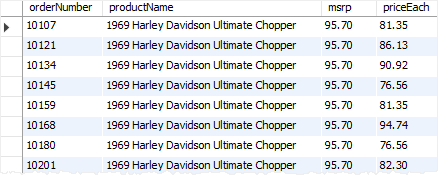
**INNER** **JOIN** orderdetails o

**ON** p.productcode = o.productcode

**AND** p.msrp > o.priceEach

**WHERE**

p.productcode = 'S10\_1678';



### Why should we use joins?

Now you may think, why we use JOINs when we can do the same task running queries. Especially if you have some experience in database programming you know we can run queries one by one, use output of each in successive queries. Of course, that is possible. But using JOINs, you can get the work done by using only a one query with any search parameters. On the other hand **MySQL can achieve better performance** with JOINs as it can use Indexing. Simply use of single JOIN query instead running multiple queries do reduce server overhead. Using multiple queries instead that leads more data transfers between MySQL and applications (software). Further it requires more data manipulations in application end also.

**It is clear that we can achieve better MySQL and application performances by use of JOINs.**