

How To Use MySQL JOINS {With Examples}

April 20, 2021

MYSQL

[Home](#) » [Databases](#) » How To Use MySQL JOINS {With Examples}

Introduction

JOINS in MySQL are used to **combine information** located in multiple tables and retrieve that information in a single result.

Relational databases contain several logically related tables linked together, and each table contains unique data or common data. JOINS help retrieve data from tables based on a common field between them.

In this tutorial, you will learn what MySQL JOINS are and how to use them.



Prerequisites

- MySQL Server and MySQL shell installed
- A [MySQL user](#) account with root privileges



What Are JOINS in MySQL?

The **JOIN** statement in MySQL is a method of linking data between several tables in a database based on common column's values in those tables.

Common values are usually the same column name and data type present in the tables being joined. Those common columns are called the *join key* or *common key*.

JOINS can be used in the **SELECT**, **UPDATE**, and **DELETE** statements.

MySQL JOIN Example

In this tutorial, we will use an example database with two tables: *customer_list* and *payments*. The goal is to make the concept of joins clear by showing the results of each join type in an example.

- The *customer_list* table is a list of people we are going to call customers. It contains information about each customer as well as their account balance. Each customer has a **unique ID**.

```
MySQL localhost:3306 ssl customer_list SQL > SELECT * FROM customer_list;
```

CustomerID	CustomerName	Address	PhoneNumber	City	PostalCode	Country	AccountBalance
1	Matthew Shakman	Example Street 1	458796	Berlin	42215	Germany	554
2	Ana Bonnet	Example Street 2	458797	Mexico	62216	Mexico	21
3	Antonio Moreira	Example Street 3	458798	London	82217	UK	336
4	Harry Vintage	Example Street 4	458799	Vienna	92218	Austria	3699
5	Miles Moore	Example Street 5	458800	Rome	62219	Italy	784
6	Jon Favreau	Example Street 6	458801	Algiers	52220	Algeria	514
7	Louis Leterrier	Example Street 7	458802	Copenhagen	2221	Denmark	201
8	Jon Watts	Example Street 8	458803	Djibouti	256222	Djibouti	4
9	Kenneth Branagh	Example Street 9	458804	Roseau	532223	Dominica	1
10	Joe Johnston	Example Street 10	458805	Santo Domingo	22294	Dominican Republic	222
11	Joss Whedon	Example Street 11	458806	Dili	2225	East Timor	963
12	Shane Black	Example Street 12	458807	Quito	332226	Ecuador	789
13	Alan Taylor	Example Street 13	458808	Cairo	29227	Egypt	8900
14	Anthony Russo	Example Street 14	458809	San Salvador	52228	El Salvador	551
15	James Gunn	Example Street 15	458810	Tehran	92229	Iran	0
16	Taika Waititi	Example Street 16	458811	Malabo	2230	Equatorial Guinea	366
17	Peyton Reed	Example Street 17	458812	Asmara	447231	Eritrea	666
18	Joe Russo	Example Street 18	458813	Tallinn	442232	Estonia	9877
19	Scott Derrickson	Example Street 19	458814	Mbabana	2233	Eswatini	456
20	Ryan Coogler	Example Street 20	458815	Addis Ababa	25234	Ethiopia	631

20 rows in set (0.0005 sec)

- The *payments* table contains information about each customer's deposits and withdrawals connected to **customer IDs**.

```
MySQL localhost:3306 ssl customer_list SQL > SELECT * FROM payments;
```

CustomerID	Deposit	Withdrawal
1	700	146
2	30	
4	4000	301
5	800	16
6	600	86
9	50	49
10	300	78
12	800	11
13	9000	
14	600	49
17	670	4
18	10000	123
19	460	4

13 rows in set (0.0004 sec)

The **common column** for these two tables is **CustomerID**, and we are going to use it as a condition in JOINS.



Note: Learn [how to create a table in MySQL](#).

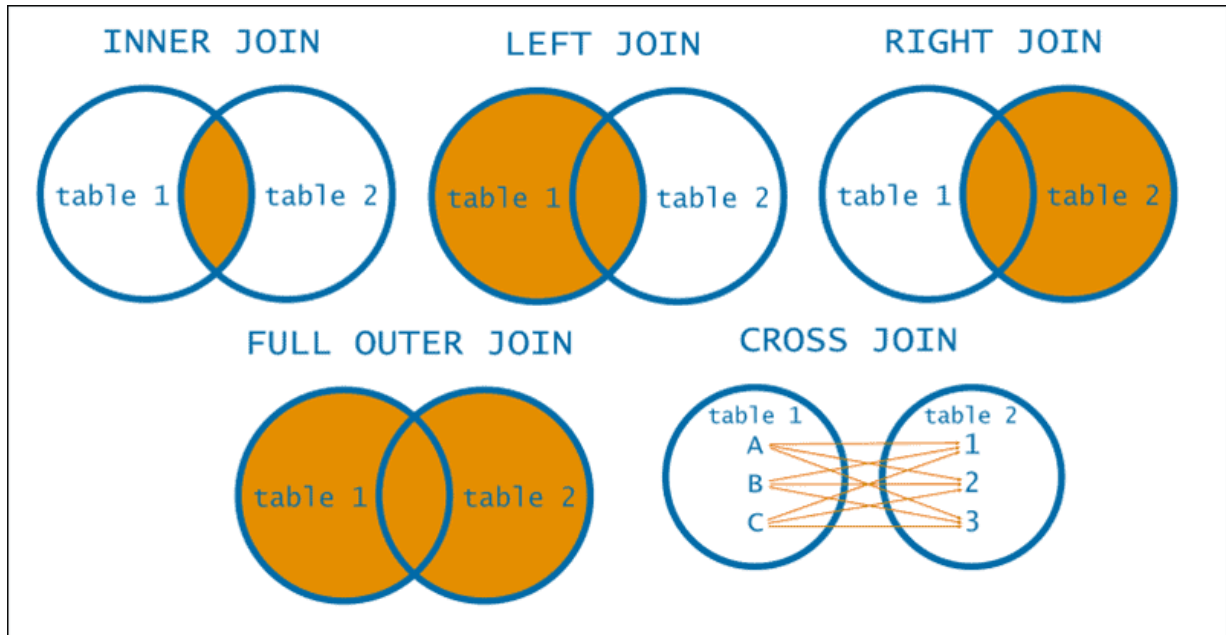
MySQL JOIN Types

There are several MySQL JOIN types, and each type helps get different results when joining tables:

- INNER JOIN** – Results return matching data from both tables.

2. **LEFT OUTER JOIN** – Results are from the left table and matching data from the right table.
3. **RIGHT OUTER JOIN** – Results are from the right table and matching data from the left table.
4. **FULL OUTER JOIN** – Results are from both tables when there is matching data.
5. **CROSS JOIN** – Results are a combination of every row from the joined tables.

The following Venn diagrams represent each join type graphically:



Different join types allow users to get results when information is present in only one of the joined tables.

INNER JOIN

The simplest join type is **INNER JOIN**. The **INNER JOIN** results with a set of records that satisfy the given condition in joined tables. It matches each row in one table with every row in other tables and allows users to query rows containing columns from both tables.

The syntax for an **INNER JOIN** is:

```
SELECT table1.column1, table1.column2, table2.column1, ...  
FROM table1  
INNER JOIN table2  
ON table1.matching_column = table2.matching_column;
```



The **matching_column** syntax represents the column common to both tables.

After the **SELECT** statement, if a column is unique to a table, there is no need to specify the table name.

Since **INNER JOIN** is considered the default join type, using only the **JOIN** statement is accepted.

For example:

```
MySQL localhost:3306 ssl customer_list SQL > SELECT customer_list.CustomerName, payments.Deposit
-> FROM customer_list
-> INNER JOIN payments
-> ON customer_list.CustomerID = payments.CustomerID;
```

CustomerName	Deposit
Matthew Shakman	700
Ana Bonnet	30
Harry Vintage	4000
Miles Moore	800
Jon Favreau	600
Kenneth Branagh	50
Joe Johnston	300
Shane Black	800
Alan Taylor	9000
Anthony Russo	600
Peyton Reed	670
Joe Russo	10000
Scott Derrickson	460

13 rows in set (0.0005 sec)

In this example, **table1.column1** is **customer_list.CustomerName**, while **table2.column1** is **payments.Deposit**. The common column for both tables is *CustomerID*.

The result-set returned shows the list of customers from the *customer_list* table and the deposits made by those customers, located in the *payments* table.



Note: See [how to find duplicate values in a table](#) in different ways, including **INNER JOIN**.

LEFT OUTER JOIN

The **LEFT OUTER JOIN** (or **LEFT JOIN**) returns all records from the table on the left side of the join and matching records from the table on the right side of the join. If there are rows for which there are no matching rows on the right-side table, the result value displayed is *NULL*.

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

```
SELECT table1.column1, table1.column2, table2.column1, ...
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```



For example:

In this example, the left table is *customer_list*, while the right table is *payments*. The result set returns **all** the customers from the *customer_list* table and the **matching results** from the *payments* table.

In places where a customer made no deposit, the returned value displayed is *NULL*.

RIGHT OUTER JOIN

The **RIGHT OUTER JOIN** (**RIGHT JOIN**) is essentially the reverse of **LEFT OUTER JOIN**.

The **RIGHT JOIN** returns all records from the table on the right side of the join and matching records from the table on the left side of the join. If there are rows for which there are no matching rows on the left-side table, the result value displayed is *NULL*.

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

```
SELECT table1.column1,table1.column2,table2.column1,....  
FROM table1  
RIGHT JOIN table2  
ON table1.matching_column = table2.matching_column;
```



For example:

In this example, the results returned show any customers that have made a deposit. The customers who did not make a deposit are **not shown** in the result.

FULL OUTER JOIN

MySQL does not support **FULL JOIN**. To get an equivalent result, use a combination of **LEFT JOIN**, **UNION ALL**, and **RIGHT JOIN**, which outputs a union of table 1 and table 2, returning all records from both tables. The columns existing only in one table will be displayed as *NULL* in the opposite table.

The syntax is:

```
SELECT * FROM table1
LEFT JOIN table2 ON table1.matching_column = table2.matching_column
UNION ALL
SELECT * FROM table1
RIGHT JOIN table2 ON table1.matching_column = table2.matching_column
```



For example:

FULL OUTER JOIN does not exclude duplicates, which is why we use **UNION ALL** to display a union of the two tables, including the duplicates. *NULL* values are displayed where there is no matching data, i.e., where customers did not make a deposit or withdrawal, or no customer ID is present.

If you want to exclude duplicates from the result set, use the **UNION** statement instead of **UNION ALL** to remove duplicate rows:

The result-set includes all matching results, **excluding duplicate rows**.

CROSS JOIN

The **CROSS JOIN** (also called CARTESIAN JOIN) joins each row of one table to every row of another table. The **CROSS JOIN** happens when the matching column or the **WHERE** condition are not specified. The result-set of a **CROSS** join is the product of the number of rows of the joined tables.

Use **CROSS JOIN** when you want a combination of every row from two tables. **CROSS JOIN** is useful when you want to make a **combination** of items, for example, colors or sizes.

If the **WHERE** condition is specified, the **CROSS JOIN** functions like an **INNER JOIN**.

The syntax for **CROSS JOIN** is:

```
SELECT table1.column1, table1.column2, table2.column1, ...  
FROM table1  
CROSS JOIN table2;
```



For example:

In this example, each record from the *CustomerName* column is joined to each row from the *Withdrawal* column. It does not make sense to use **CROSS JOIN** in a database like this, but the example illustrates the result.

Why Are JOINS Useful?

- **It's faster.** JOINS help retrieve data from two or more related database tables in a single query. JOINS are particularly useful because it is much faster than having to run queries one by one to get the same results.
- **MySQL performs better.** Another benefit of using JOINS is that MySQL performs better because it uses indexing when performing joins.
- **Using JOINS reduces server overhead.** You run only a single query, resulting in better and faster performance.



Note: Read our article to learn [how to create an index in MySQL](#).

Conclusion

You now know all the different types of MySQL JOINS and how to use them. Feel free to test out different types of joins, as it will be much clearer when you put them to use and see the results for your database example.

Was this article helpful?

Yes

No

Bosko Marijan

Having worked as an educator and content writer, combined with his lifelong passion for all things high-tech, Bosko strives to simplify intricate concepts and make them user-friendly. That has led him to technical writing at PhoenixNAP, where he continues his mission of spreading knowledge.

Next you should read

[Databases, SysAdmin,
Web Servers](#)

**MySQL Stored
Procedures
{Create, List, Alter,
& Drop}**

April 13, 2021

MySQL stored
procedures group
multiple tasks into one
and save the task on
the server for future
use. Stored...

[READ MORE](#)

[Backup and Recovery,](#)
[MySQL](#)

[How To Use MySQL Triggers](#)

April 12, 2021

MySQL triggers provide control over data validation when inserting, updating or deleting data from a...

[READ MORE](#)

[MySQL, SysAdmin](#)

[MySQL Commands Cheat Sheet](#)

January 20, 2021

Need a reference sheet for all the important MySQL commands? Check out this MySQL Commands article which...

[READ MORE](#)

[MySQL](#)

[How To Create A Table In MySQL](#)

November 3, 2020


MySQL is a well-known, free and open-source database application. One of the most crucial processes in MySQL...

[READ MORE](#)

 Live Chat

 Get a Quote

 Support | 1-855-330-1509

 Sales | 1-877-588-5918

[Privacy Center](#) [Do not sell or share my personal information](#)

[Contact Us](#)
[Legal](#)
[Privacy Policy](#)
[Terms of Use](#)
[DMCA](#)
[GDPR](#)
[Sitemap](#)

© 2022 Copyright phoenixNAP | Global IT Services. All Rights Reserved.