**What are JOINS in MySQL?**

The Joins in MySQL are basically used to fetch the rows from two or more related tables. The tables involved in the join are basically related to each other using the primary key and foreign key relationship but the important point is that it is not mandatory. What it means, the tables involved in the MySQL Joins must have a common field. Common filed means both the column must be compatible in terms of data type and based on that common field the MySQL JOINS retrieves the data.

**Why should we use JOINS?**

With JOINS we can achieve better MySQL and application performances as it can use indexing. Instead of using multiple queries JOIN simply uses a single query with any search parameters which gives us better performance as compared to subqueries.

**Types of JOINS in MySQL**

There are mainly 3 types of joins in MySQL:

1. **INNER JOIN:** The Inner join returns only the matching records from both the tables involved in the Join. Non-matching records are eliminated.
2. **OUTER JOIN:** The Outer Join retrieves the matching records as well as non-matching records from both the tables involved in the join in MySQL.
3. **CROSS JOIN:** If two or more tables are combined with each other without any condition then we call it cross join in MySQL. In cross join, each record of a table is joins with each record of another table.

CREATE DATABASE Company;

USE Company;

CREATE TABLE Employee (

Id INT PRIMARY KEY,

Name VARCHAR(45) NOT NULL,

Department VARCHAR(45) NOT NULL,

Salary FLOAT NOT NULL,

Gender VARCHAR(45) NOT NULL,

Age INT NOT NULL,

City VARCHAR(45) NOT NULL

);

INSERT INTO Employee (Id, `Name`, Department, Salary, Gender, Age, City) VALUES (1001, 'John Doe', 'IT', 35000, 'Male', 25, 'London');

INSERT INTO Employee (Id, `Name`, Department, Salary, Gender, Age, City) VALUES (1002, 'Mary Smith', 'HR', 45000, 'Female', 27, 'London');

INSERT INTO Employee (Id, `Name`, Department, Salary, Gender, Age, City) VALUES (1003, 'James Brown', 'Finance', 50000, 'Male', 28, 'London');

INSERT INTO Employee (Id, `Name`, Department, Salary, Gender, Age, City) VALUES (1004, 'Mike Walker', 'Finance', 50000, 'Male', 28, 'London');

INSERT INTO Employee (Id, `Name`, Department, Salary, Gender, Age, City) VALUES (1005, 'Linda Jones', 'HR', 75000, 'Female', 26, 'London');

INSERT INTO Employee (Id, `Name`, Department, Salary, Gender, Age, City) VALUES (1006, 'Anurag Mohanty', 'IT', 35000, 'Male', 25, 'Mumbai');

INSERT INTO Employee (Id, `Name`, Department, Salary, Gender, Age, City) VALUES (1007, 'Priyanla Dewangan', 'HR', 45000, 'Female', 27, 'Mumbai');

INSERT INTO Employee (Id, `Name`, Department, Salary, Gender, Age, City) VALUES (1008, 'Sambit Mohanty', 'IT', 50000, 'Male', 28, 'Mumbai');

INSERT INTO Employee (Id, `Name`, Department, Salary, Gender, Age, City) VALUES (1009, 'Pranaya Kumar', 'IT', 50000, 'Male', 28, 'Mumbai');

INSERT INTO Employee (Id, `Name`, Department, Salary, Gender, Age, City) VALUES (1010, 'Hina Sharma', 'HR', 75000, 'Female', 26, 'Mumbai');

CREATE TABLE Projects (

ProjectId INT PRIMARY KEY AUTO\_INCREMENT,

Title VARCHAR(200) NOT NULL,

ClientId INT,

EmployeeId INT,

StartDate DATETIME,

EndDate DATETIME,

FOREIGN KEY (EmployeeId) REFERENCES Employee(Id)

);

INSERT INTO Projects ( Title, ClientId, EmployeeId, StartDate, EndDate) VALUES

('Develop ecommerse website from scratch', 1, 1003, NOW(), DATE\_ADD(NOW(), INTERVAL 30 DAY)),

('WordPress website for our company', 1, 1002, NOW(), DATE\_ADD(NOW(), INTERVAL 45 DAY)),

('Manage our company servers', 2, 1007, NOW(), DATE\_ADD(NOW(), INTERVAL 45 DAY)),

('Hosting account is not working', 3, 1009, NOW(), DATE\_ADD(NOW(), INTERVAL 7 DAY)),

('MySQL database from my desktop application', 4, 1010, NOW(), DATE\_ADD(NOW(), INTERVAL 15 DAY)),

('Develop new WordPress plugin for my business website', 2, NULL, NOW(), DATE\_ADD(NOW(), INTERVAL 10 DAY)),

('Migrate web application and database to new server', 2, NULL, NOW(), DATE\_ADD(NOW(), INTERVAL 5 DAY)),

('Android Application development', 4, 1004, NOW(), DATE\_ADD(NOW(), INTERVAL 30 DAY)),

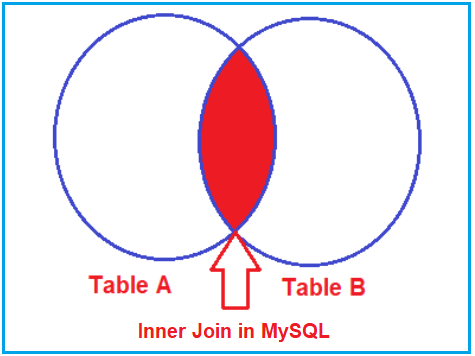
('Hosting account is not working', 3, 1001, NOW(), DATE\_ADD(NOW(), INTERVAL 7 DAY)),

('MySQL database from my desktop application', 4, 1008, NOW(), DATE\_ADD(NOW(), INTERVAL 15 DAY)),

('Develop new WordPress plugin for my business website', 2, NULL, NOW(), DATE\_ADD(NOW(), INTERVAL 10 DAY));

**Inner Join in MySQL**

An INNER JOIN in MySQL is created by using the INNER JOIN keyword. An INNER JOIN is the most common type of join used in real-time applications. The Inner Join in MySQL is used to return only the matching rows from both the tables involved in the join by removing the non-matching records. The following diagram is the pictorial representation of Inner Join.

**Note:** The INNER JOIN returns the rows in the result set where the column value in a row of table1 is equal to the column value in a row of table2. In INNER JOIN the ON clause defines the columns and condition to be evaluated.

SELECT Id as EmployeeID, Name, Department, City, Title as Project, ClientId

FROM Employee

INNER JOIN Projects

ON Employee.Id = Projects.EmployeeId;

Instead of using the **INNER JOIN** keyword, you can also use the **JOIN** keyword as shown in the below SQL Query, and also you will get the same output as the previous one.

SELECT Id as EmployeeID, Name, Department, City, Title as Project, ClientId

FROM Employee

JOIN Projects

ON Employee.Id = Projects.EmployeeId;

**Note:** In MySQL, JOIN or INNER JOIN means the same i,e. they perform the Inner Join Operation. But it recommended to use the **INNER JOIN**keyword as it explicitly specifies the type of join we are performing.

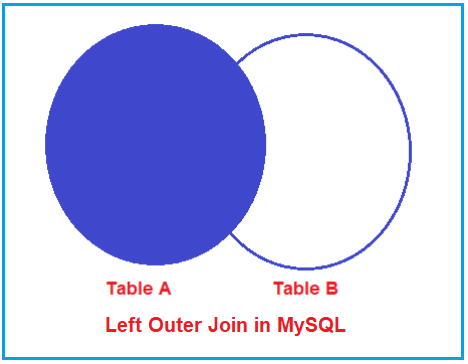
**Outer Join in MySQL**

Unlike INNER JOIN, the OUTER JOIN returns matched data rows as well as unmatched data rows from both the tables involved in the join. Outer join is again classified into three types.

1. **Left outer join**
2. **Right outer join**
3. **Full outer join**

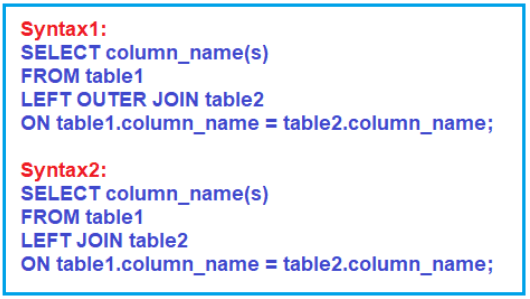
**Left Outer Join in MySQL**

The LEFT OUTER JOIN in MySQL retrieves all the matching rows from both the tables as well as non-matching rows from the left side table. In this case, the un-matching data will take a null value. The most obvious question is which is the left table and which is the right table? The answer is, the table mentioned to the left of the LEFT OUTER JOIN keyword is the left table, and the table mentioned to the right of the LEFT OUTER JOIN keyword is the right table. The following diagram shows the pictorial representation of Left Outer Join in MySQL.



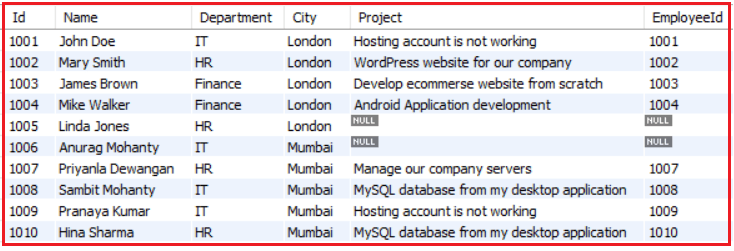
**Syntax to use Left Outer Join in MySQL:**

Please have a look at the following image which shows how to use Left Outer Join in MySQL. In MySQL, you can use either the LEFT OUTER JOIN or LEFT JOIN keyword to perform Left Outer Join. If you use only the LEFT JOIN keyword, then it is also going to perform LEFT OUTER JOIN in MySQL.



**Example to Understand Left Outer Join in MySQL**

Let us understand how to implement Left Outer Join in MySQL with an Example. Now, we need to write a query to retrieve EmployeeId, Name, Department, City, and Title as Project from the Employee and Projects tables. So, basically, we want the query to produce the following output.



Now using LEFT OUTER JOIN, we can join the 2 tables and display the information of employees and the projects they are working on, we also get the details of employees who are not working on any project. The Following is the Left Outer Join Query which is joining the Employee and Projects tables and will give the results shown in the above image.

**SELECT** Id **as** EmployeeID, Name, Department, City, Title **as** Project, ClientId

**FROM** Employee

**LEFT** **OUTER** **JOIN** Projects

**ON** Employee.Id = Projects.EmployeeId;

Or

**SELECT** Id **as** EmployeeID, Name, Title **as** Project, ClientId

**FROM** Employee

**LEFT** **OUTER** **JOIN** Projects

**ON** Employee.Id = Projects.EmployeeId;

If we look at the output, here, we got all 10 rows (i.e. all the rows from the Employee Table) including the row that has a null value for the EmployeeId column in the Projects table. So, this proofs that the Left Outer Join will retrieve all the rows from the Left-hand side Table including the rows that have a null foreign key value in the right-hand side table.

Instead of using the **LEFT OUTER JOIN** keyword, you can also use the **LEFT** **JOIN** keyword as shown in the below SQL Query, and also you will get the same output as the previous one.

**SELECT** Employee.Id **as** Id, Name, Department, City, Title **as** Project, Projects.EmployeeId

**FROM** Employee

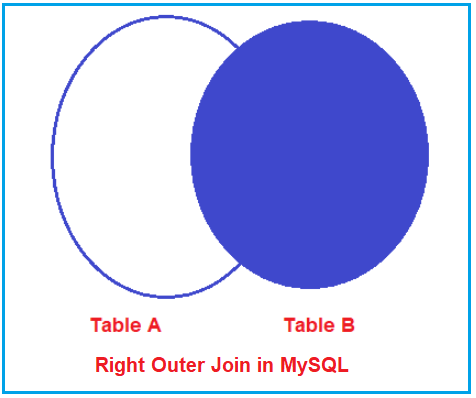
**LEFT** **JOIN** Projects

**ON** Employee.Id = Projects.EmployeeId;

**Note:** If you want to select all the records from the Left side table including the records that have a null foreign key value then you need to use the Left Outer Join. Or in other words, if you want to retrieve all the matching rows from both the tables as well as the non-matching rows from the left side table then you need to use Left Outer Join in MySQL.

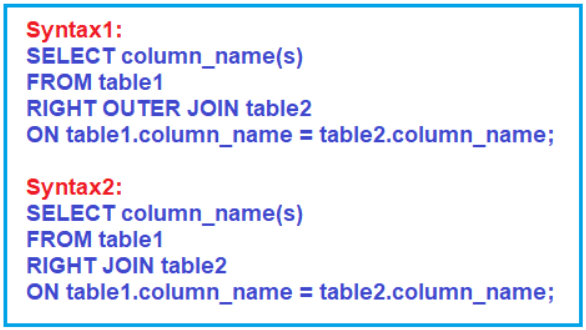
**Right Outer Join in MySQL**

The RIGHT OUTER JOIN in MySQL retrieves all the matching rows from both the tables as well as non-matching rows from the right-side table. In this case, the un-matching data will take a null value. The most obvious question is which is the left table and which is the right table? The answer is, the table mentioned to the left of the RIGHT OUTER JOIN keyword is the left table, and the table mentioned to the right of the RIGHT OUTER JOIN keyword is the right table. The following diagram shows the pictorial representation of Right Outer Join in MySQL.



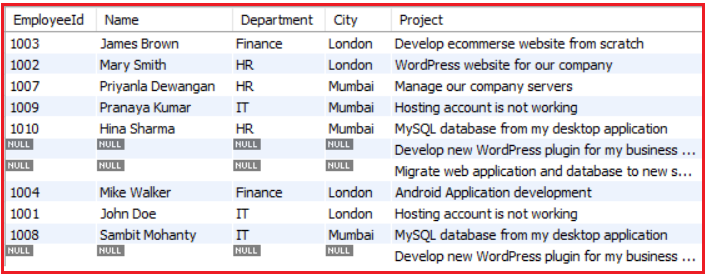
**Syntax to use Right Outer Join in MySQL:**

Please have a look at the following image which shows how to use Right Outer Join in MySQL. In MySQL, you can use either the RIGHT OUTER JOIN or RIGHT JOIN keyword to perform Right Outer Join operations. If you use only the RIGHT JOIN keyword, then it is also going to perform RIGHT OUTER JOIN in MySQL.



**Example to Understand Right Outer Join in MySQL**

Let us understand how to implement Right Outer Join in MySQL with an Example. Now, we need to write a query to retrieve EmployeeId, Name, Department, City, and Title as Project from the Employee and Projects tables. So, basically, we want the query to produce the following output.



Now using RIGHT OUTER JOIN, we can join the 2 tables and display the information of employees and the projects they are working on, we also get the details of projects which are not yet assigned to any employee. The Following is the Right Outer Join Query which is joining the Employee and Projects tables and will give the results shown in the above image.

**SELECT** Employee.Id **as** EmployeeId, Name, Department, City, Title **as** Project

**FROM** Employee

**RIGHT** **OUTER** **JOIN** Projects

**ON** Employee.Id = Projects.EmployeeId;

If we look at the output, here, we got all 11 rows (i.e. all the rows from the Projects Table). If you further notice, here, we got all the matching records from both the tables Employee and Projects as well as all the non-matching rows from the right-side table i.e. the Projects Table. So, this proofs that the Right Outer Join will retrieve all the rows from the Left-hand side Table including the rows that have a null foreign key value in the left-hand side table.

Instead of using the **RIGHT OUTER JOIN** keyword, we can also use the **RIGHT** **JOIN** keyword as shown in the below SQL Query, and also you will get the same output as the previous one.

**SELECT** Employee.Id **as** EmployeeId, Name, Department, City, Title **as** Project

**FROM** Employee

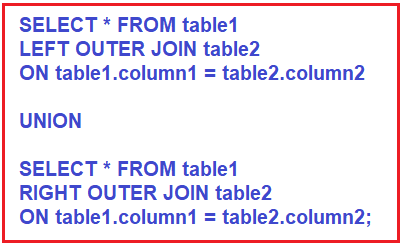
**RIGHT** **JOIN** Projects

**ON** Employee.Id = Projects.EmployeeId;

**Note:** If you want to select all the records from the right-side table including the records that have a null foreign key value then you need to use the Right Outer Join. Or in other words, if you want to retrieve all the matching rows from both the tables as well as the non-matching rows from the right-side table then you need to use Right Outer Join in MySQL.

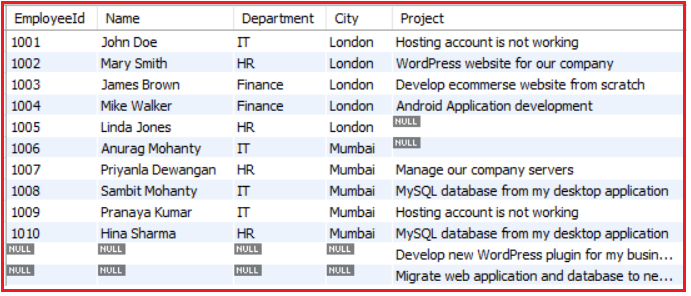
**Full Outer Join in MySQL**

The FULL OUTER JOIN retrieves all the matching rows from both the tables as well as non-matching rows from both the tables involved in the Join. In this case, the un-matching data will take a null value. MySQL doesn’t support FULL OUTER JOIN; we will achieve the FULL OUTER JOIN using UNION Operator in MySQL. The syntax to achieve Full Outer Join in MySQL using the UNION operator is given below.



**Example to Understand Full Outer Join in MySQL**

Let us understand how to implement Full Outer in MySQL with an Example. Now, we need to write a query to retrieve EmployeeId, Name, Department, City, and Title as Project from the Employee and Projects tables. So, basically, we want the query to produce the following output.



Now execute the following SQL statement. You will get all the data rows from the left as well as from the right table. The data rows from the left and right table column where the values don’t match will contain NULL means no value.

**SELECT** Employee.Id **as** EmployeeId, Name, Department, City, Title **as** Project

**FROM** Employee

**LEFT** **OUTER** **JOIN** Projects

**ON** Employee.Id = Projects.EmployeeId

**UNION**

**SELECT** Employee.Id **as** EmployeeId, Name, Department, City, Title **as** Project

**FROM** Employee

**RIGHT** **OUTER** **JOIN** Projects

**ON** Employee.Id = Projects.EmployeeId;

**Note:** If you want to select all the records from both the right and left-side table then you need to use the Full Outer Join. Or in other words, if you want to retrieve all the matching rows as well as all non-matching rows from both the tables then you need to use Full Outer Join in MySQL. Further, there is no Full Outer Join Keyword, instead, using the UNION operator we can achieve Full Outer Join.

**Cross Join in MySQL**

The CROSS JOIN is created by using the CROSS JOIN keyword. The CROSS JOIN does not contain an ON clause. Unlike INNER JOIN, the CROSS-JOIN returns matched data rows as well as unmatched data rows from the table. When we combine two or more tables with each other without any condition (where or on) then we call such type of joins as cross join. In Cross Join, each record of a table is joined with each record of the other table involved in the join.

**Example to Understand CROSS JOIN in MySQL:**

A Cross Join in MySQL produces the Cartesian product of the tables involved in the join. The Cartesian product means the number of records present in the first table is multiplied by the number of records present in the second table. Please have a look at the below SQL query which is an example of Cross Join for joining the Employee and Projects Table.

**SELECT** Employee.Id **as** EmployeeId, Name, Department, City, Title **as** Project

**FROM** Employee

**CROSS** **JOIN** Projects;

Or

**SELECT** Employee.Id **as** EmployeeId, Name, Title **as** Project

**FROM** Employee

**CROSS** **JOIN** Projects;

The Employee is the LEFT Table which contains 10 rows and Projects is the RIGHT Table which contains 11 rows. So, when you execute the above query, you will get 110 records in the result set.