MySQL JOINS

MySQL JOINS are used with SELECT statement. It is used to retrieve data from multiple tables. It is performed whenever you need to fetch records from two or more tables.

There are three types of [MySQL](https://www.javatpoint.com/mysql-tutorial) joins:

* MySQL INNER JOIN (or sometimes called simple join)
* MySQL LEFT OUTER JOIN (or sometimes called LEFT JOIN)
* MySQL RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)

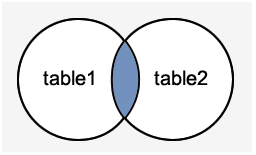
## **MySQL Inner JOIN (Simple Join)**

The [MySQL INNER JOIN](https://www.javatpoint.com/mysql-inner-join) is used to return all rows from multiple tables where the join condition is satisfied. It is the most common type of join.

**Syntax:**

1. **SELECT** columns
2. **FROM** table1
3. **INNER** JOIN table2
4. **ON** table1.**column** = table2.**column**;

**Image representation:**



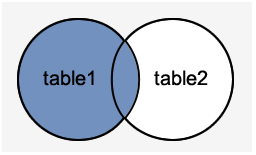
## **MySQL Left Outer Join**

The LEFT OUTER JOIN returns all rows from the left hand table specified in the ON condition and only those rows from the other table where the join condition is fulfilled.

**Syntax:**

1. **SELECT** columns
2. **FROM** table1
3. LEFT [OUTER] JOIN table2
4. **ON** table1.**column** = table2.**column**;

**Image representation:**



## **MySQL Right Outer Join**

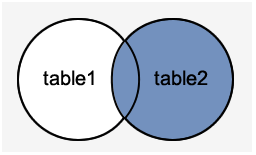
The MySQL Right Outer Join returns all rows from the RIGHT-hand table specified in the ON condition and only those rows from the other table where the join condition is fulfilled.

AD

**Syntax:**

1. **SELECT** columns
2. **FROM** table1
3. RIGHT [OUTER] JOIN table2
4. **ON** table1.**column** = table2.**column**;

**Image representation:**



**Table 1:**

CREATE TABLE employee (

emp\_id VARCHAR(10) PRIMARY KEY,

emp\_name VARCHAR(50),

salary INT,

dept\_id VARCHAR(10),

manager\_id VARCHAR(10));

INSERT INTO employee (emp\_id, emp\_name, salary, dept\_id, manager\_id)

VALUES

('E1', 'Rahul', 15000, 'D1', 'M1'),

('E2', 'Manoj', 15000, 'D1', 'M1'),

('E3', 'James', 55000, 'D2', 'M2'),

('E4', 'Michael', 25000, 'D2', 'M2'),

('E5', 'Ali', 20000, 'D10', 'M3'),

('E6', 'Robin', 35000, 'D10', 'M3');

**Table 2:**

CREATE TABLE projects (

project\_id VARCHAR(10) NOT NULL,

project\_name VARCHAR(50),

team\_member\_id VARCHAR(10),

PRIMARY KEY (project\_id, team\_member\_id));

INSERT INTO projects (project\_id, project\_name, team\_member\_id)

VALUES

('P1', 'Data Migrations', 'E1'),

('P1', 'Data Migrations', 'E2'),

('P1', 'Data Migrations', 'M3'),

('P2', 'ETL Tools', 'E1'),

('P2', 'ETL Tools', 'M4');

**Table 3:**

CREATE TABLE Manager (

manager\_id VARCHAR(10) PRIMARY KEY,

manager\_name VARCHAR(50),

dept\_id VARCHAR(10));

INSERT INTO Manager (manager\_id, manager\_name, dept\_id)

VALUES

('M1', 'Prem', 'D3'),

('M2', 'Shripadh', 'D4'),

('M3', 'Nick', 'D1'),

('M4', 'Cory', 'D1');

**Table 4:**

CREATE TABLE department (

dept\_id VARCHAR(10) PRIMARY KEY,

dept\_name VARCHAR(50)

);

INSERT INTO department (dept\_id, dept\_name)

VALUES

('D1', 'IT'),

('D2', 'HR'),

('D3', 'Finance'),

('D4', 'Admin');

* **Fetch the employee name and the department name they belong to.**
* **Inner Join / Join (Fetch Matching Records only)**

Select e.emp\_name, d.dept\_name

From employee e

Join department d on e.dept\_id = d.dept\_id;

OR

Select e.emp\_name, d.dept\_name

From employee e

Natural Join department d;

* **Fetch all the employee name and their department name they belong to.**
* **Left Join = inner join + any additional information in the left table.**

Select e.emp\_name, d.dept\_name

From employee e

Left Join department d on e.dept\_id = d.dept\_id;

* **Fetch all the department names and their employee name they belong to.**
* **Right Join = inner join + any additional information in the right table.**

Select e.emp\_name, d.dept\_name

From employee e

right Join department d on e.dept\_id = d.dept\_id;

* **Fetch details of All Employee, their manager, their department and the project they work on.**

**Step 1:**

Select e.emp\_name, d.dept\_name

From employee e

Left Join department d on e.dept\_id = d.dept\_id;

**Step 2:**

Select e.emp\_name, d.dept\_name, m.manager\_name

From employee e

Left Join department d on e.dept\_id = d.dept\_id

Inner Join manager m on m.manager\_id = e.manager\_id;

**Step 3:**

Select e.emp\_name, d.dept\_name, m.manager\_name, p.project\_name

From employee e

Left Join department d on e.dept\_id = d.dept\_id

Inner Join manager m on m.manager\_id = e.manager\_id

Left Join projects p on p.team\_member\_id = e.emp\_id;

* Full Join/ Full outer Join = Inner Join

+ all the remaining records from left table

+ all remaining records from right table

select e.emp\_name, d.dept\_name

from employee e

full join department d on d.dept\_id = e.dept\_id;

Query using in MySQL DB:

SELECT e.emp\_name, d.dept\_name

FROM employee e

LEFT JOIN department d ON d.dept\_id = e.dept\_id

UNION

SELECT e.emp\_name, d.dept\_name

FROM employee e

RIGHT JOIN department d ON d.dept\_id = e.dept\_id;

* Cross Join – Cartesian Product

SELECT e.emp\_name, d.dept\_name

FROM employee e

CROSS JOIN department d;

---write a query to fetch the employee name and their corresponding department names

---Also make sure to display the company name and the company location corresponding to each employee.

As we can’t join w.r.t certain columns in the table, we can use cross join

SELECT e.emp\_name, d.dept\_name, c. company\_name, c.location

FROM employee e

RIGHT JOIN department d ON d.dept\_id = e.dept\_id

Cross join company c;

Using WHERE instead of ON in the context of your provided query would result in a different behavior, and it might not produce the intended results. Here are the key differences and drawbacks:

Logical Difference:

ON is used for specifying the join condition between tables. In your query, you are joining employee and department based on the common column dept\_id.

WHERE is used for filtering rows after the tables have been joined. If you use WHERE to specify the join condition, it might not work correctly, as the filtering would occur after the join is already performed.

Readability and Clarity:

Using ON for join conditions is considered good practice because it makes the SQL query more readable and clearly indicates the relationship between the tables. It separates the join conditions from other filtering criteria specified in the WHERE clause.

Potential Ambiguity:

Mixing join conditions in the WHERE clause with other filtering criteria might lead to ambiguity and confusion, especially in more complex queries. It may become challenging to distinguish between join conditions and additional filters.

Maintainability:

When maintaining or modifying the query, using ON for join conditions helps isolate changes related to the join logic, making it easier to understand and modify specific parts of the query.

In summary, it's generally recommended to use ON for specifying join conditions and reserve the WHERE clause for filtering rows based on other criteria after the join has been established. This approach improves query clarity, readability, and maintainability.

In the corrected query, the **ON** keyword is used to specify the condition for the join, ensuring that you are joining rows from the "employee" table where the "dept\_id" matches with the corresponding "dept\_id" in the "department" table.

The original query might still produce the same output if your SQL database system is forgiving and interprets the **WHERE** clause in a join context, but it's considered incorrect syntax. It's safer and more standard to use the **ON** keyword for specifying join conditions.