MySQL JOINS Tutorial: INNER, OUTER, LEFT, RIGHT, CROSS

What are JOINS?

Joins help retrieving data from two or more database tables.
The tables are mutually related using primary and foreign keys.
Note: JOIN is the most misunderstood topic amongst SQL learners. For sake of simplicity and ease of understanding , we will be using a new Database to practice samples. As shown below

id	first_name	last_name	movie_id
1	Adam	Smith	1
2	Ravi	Kumar	2
3	Susan	Davidson	5
4	Jenny	Adrianna	8
6	Lee	Pong	10

id	title	category
1	ASSASSIN'S CREED: EMBERS	Animations

2	Real Steel(2012)	Animations
3	Alvin and the Chipmunks	Animations
4	The Adventures of Tin Tin	Animations
5	Safe (2012)	Action
6	Safe House(2012)	Action
7	GIA	18+
8	Deadline 2009	18+
9	The Dirty Picture	18+
10	Marley and me	Romance

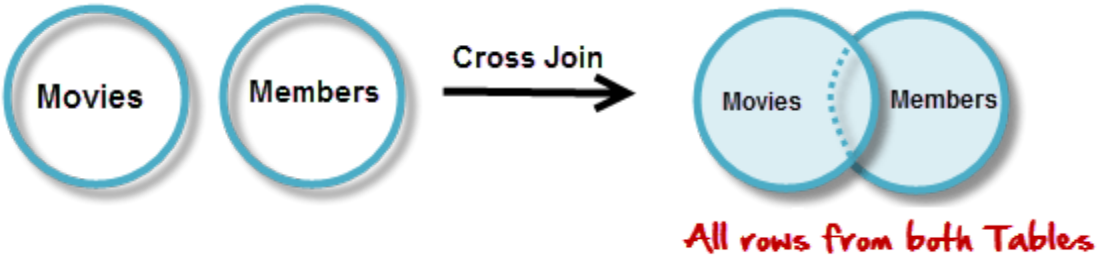
Types of joins

Cross JOIN

Cross JOIN is a simplest form of JOINs which matches each row from one database table to all rows of another.

In other words it gives us combinations of each row of the first table with all records in the second table.

Suppose we want to get all member records against all the movie records, we can use the script shown below to get our desired results.




```
SELECT * FROM `movies` CROSS JOIN `members`
```

Executing the above script in MySQL workbench gives us the following results.

id	title		id	first_name	last_name	movie_id
1	ASSASSIN'S CREED: EMBERS	Animations	1	Adam	Smith	1
1	ASSASSIN'S CREED: EMBERS	Animations	2	Ravi	Kumar	2
1	ASSASSIN'S CREED: EMBERS	Animations	3	Susan	Davidson	5
1	ASSASSIN'S CREED: EMBERS	Animations	4	Jenny	Adriana	8
1	ASSASSIN'S CREED: EMBERS	Animations	6	Lee	Pong	10
2	Real Steel(2012)	Animations	1	Adam	Smith	1

2	Real Steel(2012)	Animations	2	Ravi	Kumar	2
2	Real Steel(2012)	Animations	3	Susan	Davidson	5
2	Real Steel(2012)	Animations	4	Jenny	Adrianna	8
2	Real Steel(2012)	Animations	6	Lee	Pong	10
3	Alvin and the Chipmunks	Animations	1	Adam	Smith	1
3	Alvin and the Chipmunks	Animations	2	Ravi	Kumar	2
3	Alvin and the Chipmunks	Animations	3	Susan	Davidson	5
3	Alvin and the Chipmunks	Animations	4	Jenny	Adrianna	8
3	Alvin and the Chipmunks	Animations	6	Lee	Pong	10
4	The Adventures of Tin Tin	Animations	1	Adam	Smith	1
4	The Adventures of Tin Tin	Animations	2	Ravi	Kumar	2
4	The Adventures of Tin Tin	Animations	3	Susan	Davidson	5
4	The Adventures of Tin Tin	Animations	4	Jenny	Adrianna	8

4	The Adventures of Tin Tin	Animations	6	Lee	Pong	10
5	Safe (2012)	Action	1	Adam	Smith	1
5	Safe (2012)	Action	2	Ravi	Kumar	2
5	Safe (2012)	Action	3	Susan	Davidson	5
5	Safe (2012)	Action	4	Jenny	Adrianna	8
5	Safe (2012)	Action	6	Lee	Pong	10
6	Safe House(2012)	Action	1	Adam	Smith	1
6	Safe House(2012)	Action	2	Ravi	Kumar	2
6	Safe House(2012)	Action	3	Susan	Davidson	5
6	Safe House(2012)	Action	4	Jenny	Adrianna	8
6	Safe House(2012)	Action	6	Lee	Pong	10
7	GIA	18+	1	Adam	Smith	1
7	GIA	18+	2	Ravi	Kumar	2
7	GIA	18+	3	Susan	Davidson	5
7	GIA	18+	4	Jenny	Adrianna	8
7	GIA	18+	6	Lee	Pong	10

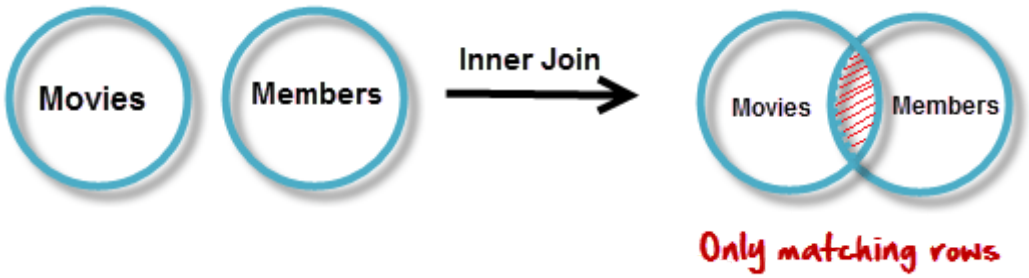
8	Deadline(2009)	18+	1	Adam	Smith	1
8	Deadline(2009)	18+	2	Ravi	Kumar	2
8	Deadline(2009)	18+	3	Susan	Davidson	5
8	Deadline(2009)	18+	4	Jenny	Adrianna	8
8	Deadline(2009)	18+	6	Lee	Pong	10
9	The Dirty Picture	18+	1	Adam	Smith	1
9	The Dirty Picture	18+	2	Ravi	Kumar	2
9	The Dirty Picture	18+	3	Susan	Davidson	5
9	The Dirty Picture	18+	4	Jenny	Adrianna	8
9	The Dirty Picture	18+	6	Lee	Pong	10
10	Marley and me	Romance	1	Adam	Smith	1
10	Marley and me	Romance	2	Ravi	Kumar	2
10	Marley and me	Romance	3	Susan	Davidson	5
10	Marley and me	Romance	4	Jenny	Adrianna	8

```
10  Marley and me     Romance  6   Lee       Pong     10
```

INNER JOIN

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

Suppose , you want to get a 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 the given conditions.




```
SELECT members.`first_name` , members.`last_name` , movies.`title`
FROM members ,movies
WHERE movies.`id` = members.`movie_id`
```

Executing the above script give

first_name	last_name	title
Adam	Smith	ASSASSIN'S CREED: EMBERS
Ravi	Kumar	Real Steel(2012)

```
Susan          Davidson      Safe (2012)

Jenny          Adrianna      Deadline(2009)

Lee            Pong          Marley and me
```

Note the above results script can also be written as follows to achieve the same results.

```
SELECT A.`first_name` , A.`last_name` , B.`title`
FROM `members`AS A
INNER JOIN `movies` AS B
ON B.`id` = A.`movie_id`
```

Outer JOINs

MySQL Outer JOINs return all records matching from both tables .

It can detect records having no match in the joined table. It returns **NULL** values for records of joined tables if no match is found.

Sounds Confusing ? Let's look into an example -

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.

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 A.`title` , B.`first_name` , B.`last_name`
FROM `movies` AS A
LEFT JOIN `members` AS B
ON B.`movie_id` = A.`id`
```

Executing the above script in MySQL workbench gives.You can see that in the returned result which is listed below that for movies which are not rented, member name fields are having NULL values. That means no matching member found members table for that particular movie.

title	first_name	last_name
ASSASSIN'S CREED: EMBERS	Adam	Smith
Real Steel(2012)	Ravi	Kumar
Safe (2012)	Susan	Davidson
Deadline(2009)	Jenny	Adrianna
Marley and me	Lee	Pong
Alvin and the Chipmunks	NULL	NULL
The Adventures of Tin Tin	NULL	NULL
Safe House(2012)	NULL	NULL
GIA	NULL	NULL

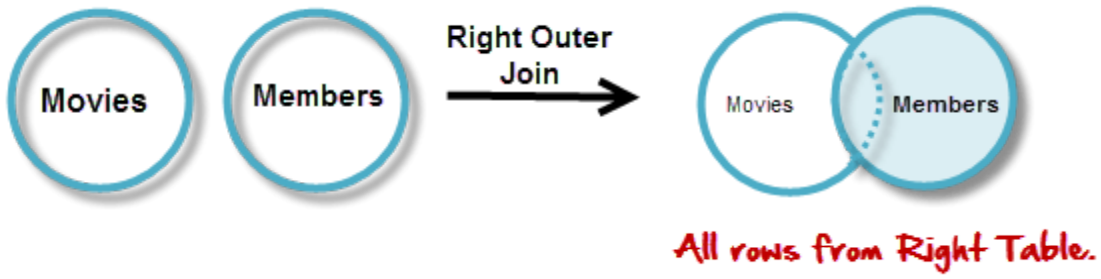
```
The Dirty Picture                NULL            NULL
```

```
Note: Null is returned for non-matching rows on right
```

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




```
SELECT  A.`first_name` , A.`last_name`, B.`title`
FROM `members` AS A
RIGHT JOIN `movies` AS B
ON B.`id` = A.`movie_id`
```

Executing the above script in MySQL workbench gives the following results.

first_name	last_name	title

Adam	Smith	ASSASSIN'S CREED: EMBERS
Ravi	Kumar	Real Steel(2012)
Susan	Davidson	Safe (2012)
Jenny	Adrianna	Deadline(2009)
Lee	Pong	Marley and me
NULL	NULL	Alvin and the Chipmunks
NULL	NULL	The Adventures of Tin Tin
NULL	NULL	Safe House(2012)
NULL	NULL	GIA
NULL	NULL	The Dirty Picture

Note: Null is returned for non-matching rows on left

"ON" and "USING" clauses

In above JOIN query examples, we have used ON clauses to match the records between tables.

USING clauses can also be used for the same purpose. The difference with **USING** is it **needs to have identical names for matched columns in both tables.**

In the "movies" table so far we used its primary key with the name "id". We referred to the same in the "members" table with the name "movie_id".

Let's rename the "movies" tables "id" field to have the name "movie_id". We do this in order to have identical matched field names.

```
ALTER TABLE `movies` CHANGE `id` `movie_id` INT( 11 ) NOT NULL AUTO_INCREMENT;
```

Next let's use USING with above LEFT JOIN example.

```
SELECT A.`title` , B.`first_name` , B.`last_name`
FROM `movies` AS A
LEFT JOIN `members` AS B
USING ( `movie_id` )
```

Apart from using **ON** and **USING with JOINs** you can use many other MySQL clauses like **GROUP BY, WHERE** and even functions like **SUM**, **AVG**, etc.

Why should we use joins?

Now you may think, why do 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, using the output of each in successive queries. Of course, that is possible. But using JOINs, you can get the work done by using only 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 a single JOIN query instead running multiple queries does reduce server overhead. Using multiple queries instead leads to more data transfers between MySQL and applications (software). Further it requires more data manipulations in the application end also.

It is clear that we can achieve better MySQL and application performance by using JOINs.

Summary

- JOINS allow us to combine data from more than one table into a single result set.
- JOINS have better performance compared to sub queries
- INNER JOINS only return rows that meet the given criteria.
- OUTER JOINS can also return rows where no matches have been found. The unmatched rows are returned with the NULL keyword.
- The major JOIN types include Inner, Left Outer, Right Outer, Cross JOINS etc.
- The frequently used clause in JOIN operations is "ON". "USING" clause requires that matching columns be of the same name.
- JOINS can also be used in other clauses such as GROUP BY, WHERE, SUB QUERIES, AGGREGATE FUNCTIONS etc