JOIN

The `JOIN` clause allows you to combine the data from 2 or more tables into one result set.

As we will be selecting from multiple columns, we need to include the list of the columns we want to choose data from after the `FROM` clause is separated by a comma.

In this chapter, we will go over the following `JOIN` types:

- `CROSS` Join
- `INNER` Join
- `LEFT` Join
- `RIGHT` Join

Before we get started, let's create a new database and two tables that we are going to work with:

- We are going to call the database `demo_joins`:

  ```
  CREATE DATABASE demo_joins;
  ```

- Then, switch to the new database:

  ```
  USE demo_joins;
  ```

- Then, the first table will be called `users`, and it will only have two columns: `id` and `username`:

  ```
  CREATE TABLE users
  (
      id INT PRIMARY KEY AUTO_INCREMENT,
      username VARCHAR(255) NOT NULL
  );
  ```

- Then, let's create a second table called `posts`, and to keep things simple, we will have three two columns: `id`, `user_id` and `title`:

  ```
  CREATE TABLE posts
  (
      id INT PRIMARY KEY AUTO_INCREMENT,
      user_id INT,
      title VARCHAR(255) NOT NULL
  );
  ```

  > The `user_id` column would be used to reference the user's ID that the post belongs to. It is going to be a one to many relations, e.g. one user could have many posts:

Image: One to many relation

- Now, let's add some data into the two tables first by creating a few users:

```
INSERT INTO users
    ( username )
VALUES
    ('bobby'),
    ('devdojo'),
    ('tony'),
    ('greisi');
```

- And finally add some posts:

```
INSERT INTO posts
    ( user_id, title )
VALUES
    ('1', 'Hello World!'),
    ('2', 'Getting started with SQL'),
    ('3', 'SQL is awesome'),
    ('2', 'MySQL is up!'),
    ('1', 'SQL - structured query language');
```

Now that we've got our tables and demo data ready, let's go ahead and learn how to use joins.

CROSS JOIN

The `CROSS` join allows you to put the result of two tables next to each other without specifying any `WHERE` conditions. This makes the `CROSS` join the simplest one, but it is also not of much use in a real-life scenario.

So if we were to select all of the users and all of the posts side by side, we would use the following query:

```
SELECT * FROM users CROSS JOIN posts;
```

The output will be all of your users and all of the posts side by side:

```
+----+----------+----+--------+-----------------+
| id | username | id |user_id | title           |
+----+----------+----+--------+-----------------+
|  4 | greisi   | 1  |      1 | Hello World!    |
|  3 | tony     | 1  |      1 | Hello World!    |
|  2 | devdojo  | 1  |      1 | Hello World!    |
|  1 | bobby    | 1  |      1 | Hello World!    |
|  4 | greisi   | 2  |      2 | Getting started |
|  3 | tony     | 2  |      2 | Getting started |
```

```
|  2 | devdojo  |  2 |      2 | Getting started |
|  1 | bobby    |  2 |      2 | Getting started |
|  4 | greisi   |  3 |      3 | SQL is awesome  |
|  3 | tony     |  3 |      3 | SQL is awesome  |
|  2 | devdojo  |  3 |      3 | SQL is awesome  |
|  1 | bobby    |  3 |      3 | SQL is awesome  |
|  4 | greisi   |  4 |      2 | MySQL is up!    |
|  3 | tony     |  4 |      2 | MySQL is up!    |
|  2 | devdojo  |  4 |      2 | MySQL is up!    |
|  1 | bobby    |  4 |      2 | MySQL is up!    |
|  4 | greisi   |  5 |      1 | SQL            |
|  3 | tony     |  5 |      1 | SQL            |
|  2 | devdojo  |  5 |      1 | SQL            |
|  1 | bobby    |  5 |      1 | SQL            |
+----+---------+----+-------+----------------+
```

As mentioned above, you will highly unlikely run a `CROSS` join for two whole tables in a real-life scenario. If the tables have tens of thousands of rows, an unqualified CROSS JOIN can take minutes to complete.

You would most likely use one of the following with a specific condition.

In MySQL, CROSS JOIN and INNER JOIN are equivalent to JOIN.

INNER JOIN

The `INNER` join is used to join two tables. However, unlike the `CROSS` join, by convention, it is based on a condition. By using an `INNER` join, you can match the first table to the second one.

As we have a one-to-many relationship, a best practice would be to use a primary key for the posts `id` column and a foreign key for the `user_id`; that way, we can 'link' or relate the users table to the posts table. However, this is beyond the scope of this SQL basics eBook, though I might extend it in the future and add more chapters.

As an example and to make things a bit clearer, let's say that you wanted to get all of your users and the posts associated with each user. The query that we would use will look like this:

```sql
SELECT *
FROM users
INNER JOIN posts
ON users.id = posts.user_id;
```

Rundown of the query:

- `SELECT * FROM users`: This is a standard select we've covered many times in the previous chapters.
- `INNER JOIN posts`: Then, we specify the second table and which table we want to join the result set.
- `ON users.id = posts.user_id`: Finally, we specify how we want the data in these two tables to be merged. The `user.id` is the `id` column of the `user` table, which is also the primary ID, and `posts.user_id` is the foreign key in the email address table referring to the ID column in the users table.

The output will be the following, associating each user with their post based on the `user_id` column:

```
+----+----------+----+---------+----------------+
| id | username | id | user_id | title          |
+----+----------+----+---------+----------------+
|  1 | bobby    | 1  |       1 | Hello World!   |
|  2 | devdojo  | 2  |       2 | Getting started |
|  3 | tony     | 3  |       3 | SQL is awesome |
|  2 | devdojo  | 4  |       2 | MySQL is up!   |
|  1 | bobby    | 5  |       1 | SQL            |
+----+----------+----+---------+----------------+
```

Note that the INNER JOIN could (in MySQL) equivalently be written merely as JOIN, but that can vary for other SQL dialects:

```
SELECT *
FROM users
JOIN posts
ON users.id = posts.user_id;
```

The main things that you need to keep in mind here are the `INNER JOIN` and `ON` clauses.

With the inner join, the `NULL` values are discarded. For example, if you have a user who does not have a post associated with it, the user with NULL posts will not be displayed when running the above `INNER` join query.

To get the null values as well, you would need to use an outer join.

Types of INNER JOIN
1. **Theta Join (θ)** :- Theta join combines rows from different tables provided they satisfy the theta condition. The join condition is denoted by the symbol θ.

Here the comparison operators ($\leq$, $\geq$, $<$, $>$, $=$, $\neg$) come into picture.

Notation :- $R_1 \bowtie_\theta R_2$.

For example, suppose we want to buy a mobile and a laptop, based on our budget we have thought of buying both such that mobile price should be less than that of laptop.

```
SELECT mobile.model, laptop.model FROM mobile, laptop WHERE mobile.price <
laptop.price;
```

1. **Equijoin** :- When Theta join uses only equality (=) comparison operator, it is said to be equijoin.

For example, suppose we want to buy a mobile and a laptop, based on our budget we have thought of buying both of the same prices.

```
SELECT mobile.model, laptop.model FROM mobile, laptop WHERE mobile.price =
laptop.price;
```

1. **Natural Join (⋈)** :- Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does.

We can perform a Natural Join only if at least one standard column exists between two tables. In addition, the column must have the same name and domain.

```
SELECT * FROM mobile NATURAL JOIN laptop;
```

LEFT JOIN

Using the `LEFT OUTER` join, you would get all rows from the first table that you've specified, and if there are no associated records within the second table, you will get a `NULL` value.

In our case, we have a user called `graisi`, which is not associated with a specific post. As you can see from the output from the previous query, the `graisi` user was not present there. To show that user, even though it does not have an associated post with it, you could use a `LEFT OUTER` join:

```
SELECT *
FROM users
LEFT JOIN posts
ON users.id = posts.user_id;
```

The output will look like this:

```
+----+----------+------+---------+----------------+
| id | username | id   | user_id | title          |
+----+----------+------+---------+----------------+
|  1 | bobby    |   1  |       1 | Hello World!   |
|  2 | devdojo  |   2  |       2 | Getting started |
|  3 | tony     |   3  |       3 | SQL is awesome |
|  2 | devdojo  |   4  |       2 | MySQL is up!   |
```

```
|  1 | bobby   |    5 |       1 | SQL             |
|  4 | greisi  | NULL |    NULL | NULL            |
+----+---------+------+---------+-----------------+
```

RIGHT JOIN

The `RIGHT OUTER` join is the exact opposite of the `LEFT OUTER` join. It will display all of the rows from the second table and give you a `NULL` value in case that it does not match with an entry from the first table.

Let's create a post that does not have a matching user id:

```
INSERT INTO posts
  ( user_id, title )
VALUES
  ('123', 'No user post!');
```

We specify `123` as the user ID, but we don't have such a user in our `users` table.

Now, if you were to run the `LEFT` outer join, you would not see the post as it has a null value for the corresponding `users` table.

But if you were to run a `RIGHT` outer join, you would see the post but not the `greisi` user as it does not have any posts:

```
SELECT *
FROM users
RIGHT JOIN posts
ON users.id = posts.user_id;
```

Output:

```
+------+---------+----+---------+-----------------+
| id   | username | id | user_id | title           |
+------+---------+----+---------+-----------------+
|    1 | bobby   |  1 |       1 | Hello World!    |
|    2 | devdojo |  2 |       2 | Getting started |
|    3 | tony    |  3 |       3 | SQL is awesome  |
|    2 | devdojo |  4 |       2 | MySQL is up!    |
|    1 | bobby   |  5 |       1 | SQL             |
| NULL | NULL    |  6 |     123 | No user post!   |
+------+---------+----+---------+-----------------+
```

Joins can also be limited with WHERE conditions. For instance, in the preceding example, if we wanted to join the tables and then restrict to only username bobby.

```
SELECT *
FROM users
```

```
RIGHT JOIN posts
ON users.id = posts.user_id
WHERE username = 'bobby';
```

Output:

```
+------+----------+----+---------+----------------+
| id   | username | id | user_id | title          |
+------+----------+----+---------+----------------+
|    1 | bobby    |  1 |       1 | Hello World!   |
|    1 | bobby    |  5 |       1 | SQL            |
+------+----------+----+---------+----------------+
```

Conclusion

Joins are fundamental to using SQL with data. The whole concept of joins might be very confusing initially but would make a lot of sense once you get used to it.

The best way to wrap your head around it is to write some queries, play around with each type of `JOIN`, and see how the result set changes.

For more information, you could take a look at the official documentation here (`https://dev.mysql.com/doc/refman/8.0/en/join.html`).