# UPDATE

As the name suggests, whenever you have to update some data in your database, you would use the `UPDATE` statement.

You can use the `UPDATE` statement to update multiple columns in a single table.

The syntax would look like this:

```
UPDATE users SET username='bobbyiliev' WHERE id=1;
```

Rundown of the statement:

- `UPDATE users`: First, we specify the `UPDATE` keyword followed by the table that we want to update.
- `username='bobbyiliev'`: Then we specify the columns that we want to update and the new value that we want to set.
- `WHERE id=1`: Finally, by using the `WHERE` clause, we specify which user should be updated. In our case, it is the user with ID 1.

The most important thing that you need to keep in mind is that if you don't specify a `WHERE` clause, all of the entries inside the `users` table would be updated, and all users would have the `username` set to `bobbyiliev`.

Important: You need to be careful when you use the `UPDATE` statement without a `WHERE` clause as every single row will be updated.

If you have been following along all of the user entries in our `users`

table, it currently have no data in the `about` column:

```
+----+----------+-------+
| id | username | about |
+----+----------+-------+
|  2 | bobby    | NULL  |
|  3 | devdojo  | NULL  |
|  4 | tony     | NULL  |
|  5 | bobby    | NULL  |
|  6 | devdojo  | NULL  |
|  7 | tony     | NULL  |
+----+----------+-------+
```

Let's go ahead and update this for all users and set the column value to `404 bio not found`, For example:

```
UPDATE users SET about='404 bio not found';
```

The output would let you know how many rows have been affected by the query:

```
Query OK, 6 rows affected (0.02 sec)
Rows matched: 6  Changed: 6  Warnings: 0
```

Now, if you were to run a select for all `users`, you would get the following result:

```
+----+----------+--------------------+
| id | username | about              |
+----+----------+--------------------+
|  2 | bobby    | 404 bio not found |
|  3 | devdojo  | 404 bio not found |
|  4 | tony     | 404 bio not found |
|  5 | bobby    | 404 bio not found |
|  6 | devdojo  | 404 bio not found |
|  7 | tony     | 404 bio not found |
+----+----------+--------------------+
```

Let's now say that we wanted to update the `about` column for the user with an id of 2. In this case, we need to specify a `WHERE` clause followed by the ID of the user that we want to update as follows:

```
UPDATE users SET about='Hello World :)' WHERE id=2;
```

The output here should indicate that only 1 row was updated:

```
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Now, if you again run the `SELECT id, username, about FROM users` query, you would see that the user with `id` of 2 now has an updated `about` column data:

```
+----+----------+-------------------+
| id | username | about             |
+----+----------+-------------------+
|  2 | bobby    | Hello World :)    |
|  3 | devdojo  | 404 bio not found |
|  4 | tony     | 404 bio not found |
|  5 | bobby    | 404 bio not found |
|  6 | devdojo  | 404 bio not found |
|  7 | tony     | 404 bio not found |
+----+----------+-------------------+
```

## Updating records using another table

As we've seen in the previous section, you can insert multiple rows in your table using another table. You can use the same principle for the update command.

To do that you simply have to list all needed table in the `update` section, then you have to explain which action you want to perform on the table, and then you need to link the table together.

For example, if you want to update the `about` field in the `users` table using the content of the `about` field in the `prospect_users` table, you would do something like this:

```
update users, prospect_users
set users.about = prospect_users.about
where prospect_users.username = users.username;
```