WHERE

The `WHERE` clause allows you to specify different conditions so that you could filter out the data and get a specific result set.

You would add the `WHERE` clause after the `FROM` clause.

The syntax would look like this:

```
SELECT column_name FROM table_name WHERE column=some_value;
```

WHERE Clause example

If we take the example `users` table from the last chapter, let's say that we wanted to get only the active users. The SQL statement would look like this:

```
SELECT DISTINCT username, email, activem FROM users WHERE active=true;
```

Output:

```
+----------+---------------+--------+
| username | email         | active |
+----------+---------------+--------+
| bobby    | b@devdojo.com |      1 |
| tony     | t@devdojo.com |      1 |
+----------+---------------+--------+
```

As you can see, we are only getting `tony` and `bobby` back as their `active` column is `true` or 1. If we wanted to get the inactive users, we would have to change the `WHERE` clause and set the `active` to `false`:

```
+----------+---------------+--------+
| username | email         | active |
+----------+---------------+--------+
| devdojo  | d@devdojo.com |      0 |
+----------+---------------+--------+
```

As another example, let's say that we wanted to select all users with the username `bobby`. The query, in this case, would be:

```
SELECT username, email, active FROM users WHERE username='bobby';
```

The output would look like this:

```
+----------+---------------+--------+
| username | email         | active |
+----------+---------------+--------+
| bobby    | b@devdojo.com |      1 |
| bobby    | b@devdojo.com |      1 |
+----------+---------------+--------+
```

We are getting 2 entries back as we have 2 users in our database with the username `bobby`.

Operators

In the example, we used the = operator, which checks if the result set matches the value that we are looking for.

A list of popular operators are:

- `!=` : Not equal operator
- `>` : Greater than
- `>=` : Greater than or equal operator
- `<` : Less than operator
- `<=` : Less than or equal operator

For more information about other available operators, make sure to check the official documentation here (`https://dev.mysql.com/doc/refman/8.0/en/non-typed-operators.html`).

AND keyword

In some cases, you might want to specify multiple criteria. For example, you might want to get all users that are active, and the username matches a specific value. This could be achieved with the `AND` keyword.

Syntax:

```
SELECT * FROM users WHERE username='bobby' AND active=true;
```

The result set would contain the data that matches both conditions. In our case, the output would be:

```
+----+----------+-------+----------+--------+---------------+
| id | username | about | birthday | active | email         |
+----+----------+-------+----------+--------+---------------+
|  2 | bobby    | NULL  | NULL     |      1 | b@devdojo.com |
|  5 | bobby    | NULL  | NULL     |      1 | b@devdojo.com |
+----+----------+-------+----------+--------+---------------+
```

If we were to change the `AND` statement to `active=false`, we would not get any results back as none of the entries in our database match that condition:

```
SELECT * FROM users WHERE username='bobby' AND active=false;
```

```
-- Output:
Empty set (0.01 sec)
```

OR keyword

In some cases, you might want to specify multiple criteria. For example, you might want to get all users that are active, or their username matches a specific value. This could be achieved with the `OR` keyword.

As with any other programming language, the main difference between `AND` and `OR` is that with `AND`, the result would only return the values that match the two conditions, and with `OR`, you would get a result that matches either of the conditions.

For example, if we were to run the same query as above but change the `AND` to `OR`, we would get all users that have the username `bobby` and also all users that are not active:

```sql
SELECT * FROM users WHERE username='bobby' OR active=false;
```

Output:

```
+----+----------+-------+----------+--------+---------------+
| id | username | about | birthday | active | email         |
+----+----------+-------+----------+--------+---------------+
|  2 | bobby    | NULL  | NULL     |      1 | b@devdojo.com |
|  3 | devdojo  | NULL  | NULL     |      0 | d@devdojo.com |
|  5 | bobby    | NULL  | NULL     |      1 | b@devdojo.com |
|  6 | devdojo  | NULL  | NULL     |      0 | d@devdojo.com |
+----+----------+-------+----------+--------+---------------+
```

LIKE operator

Unlike the = operator, the `LIKE` operator allows you to do wildcard matching similar to the * symbol in Linux.

For example, if you wanted to get all users that have the `y` letter in them, you would run the following:

```sql
SELECT * FROM users WHERE username LIKE '%y%';
```

Output

```
+----+----------+-------+----------+--------+---------------+
| id | username | about | birthday | active | email         |
+----+----------+-------+----------+--------+---------------+
|  2 | bobby    | NULL  | NULL     |      1 | b@devdojo.com |
|  4 | tony     | NULL  | NULL     |      1 | t@devdojo.com |
+----+----------+-------+----------+--------+---------------+
```

As you can see, we are getting only `tony` and `bobby` but not `devdojo` as there is no `y` in `devdojo`.

This is quite handy when you are building some search functionality for your application.

IN operator

The `IN` operator allows you to provide a list expression and would return the results that match that list of values.

For example, if you wanted to get all users that have the username `bobby` and `devdojo`, you could use the following:

```
SELECT * FROM users WHERE username IN ('bobby', 'devdojo');
```

Output:

```
+----+----------+-------+----------+--------+---------------+
| id | username | about | birthday | active | email         |
+----+----------+-------+----------+--------+---------------+
|  2 | bobby    | NULL  | NULL     |      1 | b@devdojo.com |
|  3 | devdojo  | NULL  | NULL     |      0 | d@devdojo.com |
|  5 | bobby    | NULL  | NULL     |      1 | b@devdojo.com |
|  6 | devdojo  | NULL  | NULL     |      0 | d@devdojo.com |
+----+----------+-------+----------+--------+---------------+
```

This allows you to simplify your `WHERE` expression so that you don't have to add numerous `OR` statements.

IS operator

If you were to run `SELECT * FROM users WHERE about=NULL;` you would get an empty result set as the = operator can't be used to check for NULL values. Instead, you would need to use the `IS` operator instead.

The `IS` operator is only used to check `NULL` values, and the syntax is the following:

```
SELECT * FROM users WHERE about IS NULL;
```

If you wanted to get the results where the value is not NULL, you just need to change `IS` to `IS NOT`:

```
SELECT * FROM users WHERE about IS NOT NULL;
```

BETWEEN operator

The `BETWEEN` operator allows to select value with a given range.The values can be numbers, text, or dates. BETWEEN operator is inclusive: begin and end values are included.

For Example if you want to select those user which have id between 3 and 6.

```
SELECT * FROM users WHERE id BETWEEN 3 AND 6;
```

Output:

```
+----+----------+-------+----------+--------+---------------+
| id | username | about | birthday | active | email         |
+----+----------+-------+----------+--------+---------------+
|  3 | devdojo  | NULL  | NULL     |      0 | d@devdojo.com |
|  5 | bobby    | NULL  | NULL     |      1 | b@devdojo.com |
|  6 | devdojo  | NULL  | NULL     |      0 | d@devdojo.com |
+----+----------+-------+----------+--------+---------------+
```

Conclusion

In this chapter, you've learned how to use the `WHERE` clause with different operators to get different type of results based on the parameters that you provide.

In the next chapter, we will learn how to order the result set.