In the previous section, we introduced the `WHERE` clause. This section will provide you with additional practice using the `WHERE` clause, and it will help you learn how to retrieve data using different conditions easily.

Sample data

Search results for the entire chapter will reference this sample dataset:

id	name	age	height
1	John	50	180
2	May	40	NULL
3	Tim	10	170
4	Jay	20	155

To help solidify the new concepts, you can follow along by editing the provided table below:

402: DB-Fiddle subscription required

Basic Conditions

Equal To, Greater Than, and Less Than

Let's start by reviewing a sample of the simplest `WHERE` usage and its results:

```
SELECT * FROM `new_schema`.`users` WHERE id = 1;
```

id	name	age	height
1	John	50	180

In addition to the `=` symbol, SQL also supports the greater than and less than symbols:

```
SELECT * FROM `new_schema`.`users` WHERE id > 2;
```

id	name	age	height
3	Tim	10	170
4	Jay	20	155

```
SELECT * FROM `new_schema`.`users` WHERE id <= 1;
```

id	name	age	height
1	John	50	180

Suppose you want to use a negative condition, that is not equal then we can use `!=`:

```
SELECT * FROM `new_schema`.`users` WHERE id != 1;
```

id	name	age	height
2	May	40	NULL
3	Tim	10	170
4	Jay	20	155

NULL

In SQL, if you want to determine that the value in a column is equal to `NULL`, it cannot be written as `columnA = NULL`. Because `NULL` is a special value for the database, it cannot be used in a general value comparison. Instead, we can use the following:

```
SELECT * FROM `new_schema`.`users` WHERE height IS NULL;
```

id | name | age | height ---------- | ----- | ----- | ----- | 2 | May | 40 | NULL

For the opposite condition, you only need to add one more keyword, NOT.

```
SELECT * FROM `new_schema`.`users` WHERE height IS NOT NULL;
```

id	name	age	height
1	John	50	180
3	Tim	10	170
4	Jay	20	155

Multiple Conditions

The previous section focused on the data extraction statement using a single condition. What should we do if we want to use multiple conditions? In this situation, we need to use logical operators, which is to combine `AND` and `OR` keywords in SQL statements to meet our requirements.

AND

```
SELECT * FROM `new_schema`.`users` WHERE age < 40 AND height > 160;
```

id	name	age	height
3	Tim	10	170

In this statement, the `AND` keyword is used to ensure that each record satisfies both the condition on the left side (`age < 40`) and the condition on the right side (`height > 160`).

OR

```
SELECT * FROM `new_schema`.`users` WHERE age < 40 OR height > 160;
```

id	name	age	height
1	John	50	180
3	Tim	10	170
4	Jay	20	155

This query uses the `OR` keyword to ensure if the data that meets one of the conditions (`age < 40` on the left or `height > 60` on the right), it will be fetched.

Pay attention to the record for May. Because May's height is `NULL`, it will not be treated as an empty value or 0 in the operation. It would be directly excluded from the result!

AND + OR

```
SELECT * FROM `new_schema`.`users` WHERE id < 4 AND (age > 30 OR height > 175);
```

id	name	age	height
1	John	50	180
2	May	40	NULL

This query is more complicated but common in the real world. It needs to combine multiple conditions to form this SQL statement.

Therefore, you can use parentheses to separate the logical blocks to be operated independently. Assuming that there are no parentheses, the current condition `(age > 30 OR height > 175)`, which would have been wrapped together and calculated together, will be disassembled and operated separately, thus affecting the final data result.

Note: You can refer to Operator Precedence on MySQL for more details about why the parentheses are important here.

Range Conditions

Range query is also one of the common types of data searching problems. The three most common types of ranges in SQL are `IN`, `BETWEEN`, and `LIKE`.

IN

```
SELECT * FROM `new_schema`.`users` WHERE `id` IN (1, 3);
```

id	name	age	height
1	John	50	180
3	Tim	10	170

Using the `IN` keyword will make SQL only compare the value of the specified column absolutely precise, such as in this case, only records with an id equal to 1 or 3 will be fetched.

And just like the earlier example statement for `IS NULL`, you can add a `NOT` keyword before `IN` to obtain the opposite result.

For example:

```
SELECT * FROM `new_schema`.`users` WHERE id NOT IN (1, 4);
```

id	name	age	height
2	May	40	NULL
3	Tim	10	170

BETWEEN

```
SELECT * FROM `new_schema`.`users` WHERE height BETWEEN 160 AND 190;
```

id	name	age	height
1	John	50	180
3	Tim	10	170

`BETWEEN` needs to be used with the `AND` keyword to fetch the data within a specific range. And like `IN`, there is also a reverse query mode with `NOT`.

LIKE

`LIKE` is a powerful SQL keyword that supports regular expressions which are often used for text comparison in many programming languages.

We are going to introduce the three most commonly used search and comparison methods

```
SELECT * FROM `new_schema`.`users` WHERE name LIKE '%a%';
```

id	name	age	height
2	May	40	NULL
4	Jay	20	155

The matching logic used in this sentence is to find records with names that contain the letter "a". Use two `%` symbols (one on each side), and put the word you want to search in the middle.

Note: The percent sign (`%`) will match zero, one, or multiple characters. To match exactly one character we could use an underscore (`_`).

```
SELECT * FROM `new_schema`.`users` WHERE name LIKE 'J%';
```

id	name	age	height
1	John	50	180
4	Jay	20	155

In addition to the normal contain comparison mode, you can also use another syntax to find the data whose name starts with J. The approach is leaving only a `%` symbol and putting it at the end of the searched string.

Notice the percent sign (`%`) is a wildcard; it will match zero, one, or multiple characters. To match exactly one character, we could use an underscore (`_`).

So if you want to find out the data which's name ends with y, Just write:

```
SELECT * FROM `new_schema`.`users` WHERE name LIKE '%y';
```

id	name	age	height
2	May	40	NULL
4	Jay	20	155