1. How to find the data without duplicates.
2. How to paginate the result.
3. How to sort the result.
4. How to group the data into specified columns.

We have covered the usage of SQL syntax in some depth through the previous sections, and by now, you can fetch the data you want from a specific table according to your needs.

However, in the workplace, it is not possible to complete the task by pulling out the data only. In fact, there will be a lot of changes in different situations, such as limiting the first 50 results or sorting by specific columns, and this chapter will help you learn the most commonly used auxiliary statements.

Sample Data

id	name	age	height
1	John	40	150
2	May	30	140
3	Tim	25	170
4	Jay	60	185
5	Maria	30	190
6	Tom	53	200
7	Carter	40	145

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

Schema SQL

```
CREATE SCHEMA `new_schema` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

CREATE TABLE `new_schema`.`users` (
  `id` INT NOT NULL AUTO_INCREMENT COMMENT 'This is the primary index',
  `name` VARCHAR(45) NOT NULL DEFAULT 'N/A',
  `age` INT NULL,
  `height` INT NULL,
  PRIMARY KEY (`id`)
);

INSERT INTO `new_schema`.`users` (`id`, `name`, `age`, `height`) VALUES
  (1, 'John', 40, 150),
  (2, 'May', 30, 140),
  (3, 'Tim', 25, 170),
  (4, 'Jay', 60, 185),
  (5, 'Maria', 30, 190),
  (6, 'Tom', 53, 200),
  (7, 'Carter', 40, 145);
```

Query SQL

```
SELECT * FROM `new_schema`.`users`;
```

Result

Query #1 Execution time: 1ms

id	name	age	height
1	John	40	150
2	May	30	140
3	Tim	25	170
4	Jay	60	185
5	Maria	30	190
6	Tom	53	200
7	Carter	40	145

Uniqueness: DISTINCT

When the number of data becomes very large, there is usually a problem that you need to find the results that are not repeated in a specific column. At this time, we can use `DISTINCT` to limit the results to avoid repeated values:

```
SELECT DISTINCT age FROM `new_schema`.`users`;
```

Result:

age
40
30
25
60
53

Notice that while both Maria and May are 30, and both John and Carter are 40, the numbers 30 and 40 each appear only once in the result.

Pagination: LIMIT & OFFSET

`LIMIT` and `OFFSET` are often used when the result has too much data. The syntax logic used by the page feed display function is a very important syntax. These two keywords will provide the functions of limiting the number of items displayed and skipping the first specified number of items.

```
SELECT * FROM `new_schema`.`users` LIMIT 3 OFFSET 1;
```

Result:

id	name	age	height

2	May	30	140
3	Tim	25	170
4	Jay	60	185

From the results, we note particularly that the number after `LIMIT` represents how many records are displayed, and the number after `OFFSET` refers to the number of pieces of data to be offset, that is the result will skip a specific number of results.

Therefore, many SQL programs provide paging functions by dynamically generating the numbers after `OFFSET`. For example, if the requirement is that every 3 rows is a page, one option is to dynamically generate the following queries:

```
SELECT * FROM `new_schema`.`users` LIMIT 3 OFFSET 0;



SELECT * FROM `new_schema`.`users` LIMIT 3 OFFSET 3;



SELECT * FROM `new_schema`.`users` LIMIT 3 OFFSET 6;
```

And the result will be:

First page

id	name	age	height
1	John	40	150
2	May	30	140
3	Tim	25	170

Second page

id	name	age	height
4	Jay	60	185
5	Maria	30	190
6	Tom	53	200

Third page

id	name	age	height
7	Carter	40	145

Sorting: ORDER

`ORDER` is usually used at the end of the query to optimize the search results, so that the data can be sorted according to a certain column.

That is convenient for us to visually analyze the appearance of the data with the column like updated_time, or order_price and to sort them from big to small or small to big.

```
SELECT * FROM `new_schema`.`users` ORDER BY age;
```

Result:

id	name	age	height
3	Tim	25	170
2	May	30	140
5	Maria	30	190
1	John	40	150
7	Carter	40	145

6	Tom	53	200
4	Jay	60	185

As shown in the above results, it will be sorted in ascending by age, from small to large, and this statement is actually an abbreviation of the following sentence:

```
SELECT * FROM `new_schema`.`users` ORDER BY age ASC;
```

Therefore, by replacing the `ASC` the keyword at the back of the `age`, with `DESC`, the data can be sorted in descending order, that is, from large to small:

```
SELECT * FROM `new_schema`.`users` ORDER BY age DESC;
```

Result:

id	name	age	height
4	Jay	60	185
6	Tom	53	200
7	Carter	40	145
1	John	40	150
5	Maria	30	190
2	May	30	140
3	Tim	25	170

Multi-column Sorting

In addition to sorting by a single column, the result can also be sorted by multiple columns. When two records have the same value in the first column, they will be sorted according to the values in the second column, for example:

```
SELECT * FROM `new_schema`.`users` ORDER BY age DESC, height DESC;
```

Result:

id	name	age	height
4	Jay	60	185
6	Tom	53	200
1	John	40	150
7	Carter	40	145
5	Maria	30	190
2	May	30	140
3	Tim	25	170

Grouping: GROUP BY

This is a key concept that was mentioned at the beginning of this series: the SQL programming language focuses on grouping logic. The keyword `GROUP BY` is the embodiment of this concept, and the keyword can be used like this:

```
SELECT `age` FROM `new_schema`.`users` GROUP BY age;
```

Result:

age
40
30
25
60
53

You will find that the result of this statement is very similar to using `DISTINCT`, but `GROUP BY` also supports Aggregate Function statements, so the data can be combined to generate results like reports (to learn more about aggregate functions, see the following chapters!).

For example, suppose we want to find out how many records there are in each age group, then we can write:

```
SELECT COUNT(*), `age` FROM `new_schema`.`users` GROUP BY age;
```

Result:

count(*)	age
2	40
2	30
1	25
1	60
1	53

And here, we can use the various optimizing `SELECT` statements we have learned to optimize the results. For example, we can use `AS` to change the name of the `count(*)` column to make it easier to read, and we can sort the results from small to large:

```
SELECT COUNT(*) AS `age_count`, `age`
FROM `new_schema`.`users`
GROUP BY age
ORDER BY `age_count`;
```

Result:

age_count	age
1	25
1	60
1	53
2	40
2	30

Finally, a quick tip to readers is that when practicing and using SQL, you can divide your thinking into two parts. 1. The first part is how to write a query that meets the given conditions. 2. The second part is how to display the generated result.