1. Common SQL keywords for statistics such as `COUNT`, `SUM`, `AVG`, and others.
2. Usage for common SQL functions and practical considerations.

What is the function?

A function is a technique commonly used in programming languages. Its purpose is to wrap complex logical operations and replace them with a single word or phrase. Usually, it can also include a set of parentheses, and inside the parentheses are the data that would be used for complex operations.

For example, suppose there is a function called `eat()`, the details of which may include chewing, swallowing, digesting, etc., but we don't need to know the details of how it works, just know that we can call `eat(cake)`, `eat(noodles)`, to quickly complete complex logic.

Therefore, we can find that the word we used to call a keyword, like JSON_EXTRACT, is actually a function.

Overview

In the previous chapters, I believe you will have an impression of the keyword `COUNT()`, with which you can make simple statistics in SQL, and such keywords in SQL are called `Aggregate Functions`. SQL's aggregate functions are of course more than just `COUNT()`. This chapter will introduce you to some common aggregate functions.

Sample data

id	name	age	height
1	John	40	150
2	May	30	140
3	Tim	25	180
4	Jay	40	160

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, 180),
  (4, 'Jay', 40, 160);
```

Query SQL

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

Result

Query #1 Execution time: 0ms

id	name	age	height
1	John	40	150
2	May	30	140
3	Tim	25	180
4	Jay	40	160

Counting: COUNT

The first one to introduce is the most familiar one to warm up. `COUNT()` can help us make a simple calculation about how many pieces of eligible data records exist:

```
SELECT COUNT(*) AS `user_count` FROM `new_schema`.`users` WHERE id > 1;
```

Result:

user_count
3

This tells us that there are 3 users with an id greater than 1. Specifically, May, Tim, and Jay.

Total: SUM

`SUM` can help us to aggregate the final results of a specific column. It is a very common function. In practice, it will be used to aggregate different data such as order price, user points...etc.

```
SELECT SUM(`age`) AS `sum_of_user_ages` FROM `new_schema`.`users`;
```

Result:

sum_of_user_ages
135

Average: AVG

Since there is an aggregate, of course, there will also be an average. The average is often used in various statistical reports, and it is also a very common SQL aggregate function.

```
SELECT AVG(`height`) AS `avg_user_height` FROM `new_schema`.`users`;
```

Result:

avg_user_height
157.5

Minimum & Maximum: MIN & MAX

Since there are count, sum, and average, of course, the minimum and maximum are indispensable. After learning the previous aggregate functions, the language used to obtain the minimum and maximum values of a specific column will feel natural:

```
SELECT MIN(`height`) AS `user_min` FROM `new_schema`.`users`;
```

Result:

user_min
140

```
SELECT MAX(`height`) AS `user_max` FROM `new_schema`.`users`;
```

Result:

user_max
180

In addition to the most common aggregate functions mentioned in this chapter, it is recommended that you can refer to MySQL Aggregate Functions to see what other statistical tools are available in the typical database software!

Other Functions

In addition to the aggregate functions that will be used in statistics, there are many functions that can be used for convenient operations in SQL. Here is an example of the word combination function that is commonly seen. In SQL, you can use `CONCAT()` to manipulate the presented data:

```
SELECT CONCAT(`id`, '-', `name`) AS `identification`, `age` FROM
`new_schema`.`users`;
```

Result:

identification	age
1-John	40
2-May	30
3-Tim	25
4-Jay	40

But at this time, you may have one question:

Since there is a column name `identification`, is it possible to use a statement similar to `WHERE identification LIKE '%J%'` to find out the data that meet the conditions?

The answer is no, but also yes.

What does it mean?

The reason it is impossible is because the `WHERE` keyword only supports columns that already exist in this table, and we can see that `identification` is a column that is manually combined after being picked up in `SELECT`. The column is processed by us, so we cannot use `WHERE` to filter the column.

This is a problem that is easy to encounter when you want to quickly use `WHERE` to finish tasks when maintaining SQL codes which has complex structure and long history.

But the reason why it is possible is because, for this situation, SQL supports a special keyword `HAVING` to help developers filter the conditions for combined columns, which are written as follows:

```
SELECT CONCAT(`id`, '-', `name`) AS `identification`, `age` FROM `new_schema`.`users`
HAVING `identification` LIKE '%J%';
```

Result:

identification	age
1-John	40
4-Jay	40

I believe you may have a thought here:

> WHERE looks the same as HAVING, so I'll just use HAVING forever!

Unfortunately, if we only use `HAVING`, as the amount of data increases, the performance may be much slower than `WHERE` in different situations, which will affect the efficiency of our database and relative application. So, it is not good to just use `HAVING` instead of `WHERE`. In practice, we still suggest that you use `WHERE` to query data with different conditions as much as possible.