In more advanced stages of SQL syntax, the ways of using it in various relational database software will be different. But for basic operations, the usage of each software is almost the same, just a little bit different.

Basic syntax

Let's take a look at a basic SQL syntax structure first:

```
SELECT `id`, `name` FROM `new_schema`.`users`;
```

You can see the part of the grammar above, `SELECT` and `FROM` are keywords in SQL, and we will discuss what they mean later. Regarding the other words used in the middle, `id` and `name` are column names, `new_schema` is the schema name, and `users` is the table name.

Sample data

Let's start with a sample table `users`, to help you understand what kind of results will be presented when an SQL statement is executed.

id	name	age
1	John	50
2	May	40
3	Tim	10

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

```
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', 50,
180);
```

```
INSERT INTO `new_schema`.`users` (`id`, `name`, `age`) VALUES (2, 'May', 40);
INSERT INTO `new_schema`.`users` (`id`, `name`, `age`, `height`) VALUES (3, 'Tim', 10,
170);
INSERT INTO `new_schema`.`users` (`id`, `name`, `age`, `height`) VALUES (4, 'Jay', 20,
155);
```

Create data: INSERT

```
INSERT INTO `new_schema`.`users` (`id`, `name`, `age`) VALUES (4, 'Harry',
33);
```

This is the query for adding data to the database. There are three keywords here: 1. `INSERT INTO`: These two can be put together and are mainly used to specify the database table and columns to be added. So this statement will add one new entry into the `users` table and set a value for the `id`, `name`, and `age`. These columns must be prepared before adding data. 2. `VALUES`: The value to be added will be set after this word, and they will need to correspond to the columns specified after `INSERT INTO`. After the query is executed, a record with id equal to 4, name of Harry, age of 33 will be inserted into to the database.

When writing an insert statement, the main point to remember is that the field names and values must be correctly mapped to one another. As such, the `INSERT` statement is relatively simple compared to the `SELECT` statement, which has other, more complex syntax that we will introduce in subsequent chapters.

Create Multiple New Records

To add multiple records, you can add a new set of data directly after `VALUES`. Here is an example:

```
INSERT INTO `new_schema`.`users` (`id`, `name`, `age`) VALUES (4, 'Harry',
33), (5, 'Tom', 30);
```

Result:

id	name	age
1	John	50

2	May	40
3	Tim	10
4	Harry	33
5	Tom	30

Read Data: SELECT

Reading data is one of the most common uses for SQL. Let's take a look at some sample code:

```
SELECT `id`, `name` FROM `new_schema`.`users`;
```

There are two keywords here, `SELECT` and `FROM`: 1. `FROM`: This word refers to operating from which table of which database. In the example, this refers to operating from the `users` table in the `new_schema` database. 2. `SELECT`: This word refers to select which columns to display. In the example, it will display the id and name columns in the result.

Executed result:

id	name
1	John
2	May
3	Tim

Read All

In addition, many developers prefer to use the `SELECT` statement as follows in small projects:

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

Using an asterisk ('*'), instead of listing the column names, will fetch *all* of the columns in the table, and you will get the following result:

id	name	age
1	John	50
2	May	40
3	Tim	10

It's a very convenient statement, and you don't need to specifically list all the columns. However, it should be noted that if the data in the table is very large, the performance of `SELECT *` may be very slow.

Therefore, in a more complex and large database, only use `SELECT *` after careful consideration.

Conditions

As for how to fetch out the desired data according to conditions, you can refer to the next chapter, which discusses the `WHERE` clause in more detail. Here, we will start with a simple example statement and the result:

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

id	name	age
2	May	40

As you can see, the result only shows the data where id is equal to 2.

In a technical article or conversation, an SQL statement would be referred to as a "query," going forward, we may use the term "query" when referring to an SQL statement.

Modify Data: UPDATE

```
UPDATE `new_schema`.`users` SET `name` = 'Andy', `age` = 100 WHERE `id` = 2;
```

There are two keywords we can focus on: 1. `UPDATE`: Following this keyword is the name of the table to be updated. 2. `SET`: Following are the value(s) that should be updated in the existing data. If there are multiple columns, separate them with commas.

We often use the `WHERE` keyword to filter data. Although it is optional and does not have to be written, if it is not added, it will become a full table update, which is a rare situation.

There are also many database software that have default protections to prevent such dangerous operations from taking place.

After the execution is completed (with the `WHERE` clause), it will only update the record whose id is 2 by setting the name to Andy and the age to 100.

Result:

id	name	age
1	John	50
2	Andy	100
3	Tim	10

Remove Data: DELETE

```
DELETE FROM `new_schema`.`users` WHERE `id` = 1;
```

The only new keyword here is `DELETE`. It is connected to `FROM`, which is a keyword we have seen before and refers to which table to operate from.

The operation logic here is similar to `UPDATE`. `WHERE` can be added or not, but in practice, it is unlikely to delete the data of the entire table. Therefore we usually need to include `WHERE` to limit the range of data to be removed.

Result:

id	name	age
2	May	40
3	Tim	10