Data Types

First of all, we can observe the type of data that will actually be stored. In fact, it is not very diverse.

It can be divided to three types: number, datetime, and text. In database software, we can use these three type and their extensions to set different data types. In addition to these data types, SQL also has some special data types, which will be introduced later.

Number

1. `BIGINT`, `INT`, `MEDIUMINT`, `SMALLINT`, `TINYINT`: Represent integers. The farther to the left, the larger the number that can be stored. The most common setting is `INT`, which can store values ranging from -2147483648 to 2147483647.
2. `DOUBLE`, `FLOAT`, `DECIMAL`: Represent decimals, `DECIMAL` can be set to `DECIMAL(5, 2)`, which means that the value of the column must be 5 digits, and two of the digits occur after the decimal point, such as a value like `666.88`.

It should be noted that the values stored in `DOUBLE` and `FLOAT` are not precise. When you store 2.5 for related operations, it is likely to be calculated as 2.500000002. Therefore, `DECIMAL` is recommended for high-accuracy data system requirements.

Datetime

1. `DATE`, `MONTH`, `YEAR`: Set to date, month, and year.
2. `DATETIME`, `TIMESTAMP`: A relatively complete date data format, in addition to the year, month, and day, it will also include hours, minutes, and seconds, which is more precise than `DATE`. `DATETIME` can accept a purely datetime value format like `8888-01-01 00:00:00`, but `TIMESTAMP` is limited to between `1970-01-01 00:00:01` and `2038-01-19 03:14:07`

Text

1. `CHAR`, `VARCHAR`: Store plain text, the former is suitable for data with fixed text length, such as currency abbreviations. The latter is suitable for data whose length will change, and the length of the text that can be stored needs to be set. The most common default setting is `VARCHAR(45)` for MySQL, which means each item in the column can store 45 characters.

This is a point where many SQL beginners make mistakes when setting the column data type. Setting a size that is too small leads to exceeding the column size limit error.

2. `TEXT`, `LONGTEXT`: If you want to store text data whose maximum length is unknown, you can use `TEXT` related data type settings.

Special Data Types

There are also several special data types: 1. `BINARY`, `BLOB`: Store data of file type, such as images or videos. `BLOB` (Binary Large Object) is not only for those types but also for large unstructured data. However, it is rarely used in practice because it is difficult to manage files through database software. 2. `BOOLEAN`: Store the data of the logical operand. In simple terms, it is the data true or false. The database will replace it with 1 or 0 respectively. Suppose there is a column called `is_alive`, then each value is either true or false. 3. `JSON`: A very common data format in modern data exchange. We will provide more details on this data type in the `SQL Syntax: JSON` chapter.

Column Attribute Functions

In a demonstration statement from the previous page, the column attribute function `NOT NULL` was added to the `id` column. The purpose of the column attribute function is to ensure that when new data is added to the table, the database will process the data format or content in advance.

Through the column attribute function, users can more easily maintain the correctness of the data. Let's take a moment to understand the most common column attribute functions: 1. `NOT NULL`: It means that the value in the field cannot be `NULL`. So when we use `NOT NULL`, we are forcing the column to NOT accept null values, which means the field must always contain a value. In the world of programming languages, `NULL` means nothing. For example, if the value is blank text, it means at least blank text, but it cannot be anything. In the real world, it can be imagined that it is like the contrast between air and vacuum. Although air cannot be seen, it is still there, but the vacuum is really nothing. 2. `AUTO_INCREMENT`: In the database, it will automatically generate the column's values one by one using numbers. It is similar to a serial number. The database software will help us avoid the problem of repeated serial numbers, so you don't need to write your own programs to compare and set the value. 3. `DEFAULT`: You can set the default value of the column because the data to be inserted in the field may be empty in some circumstances.

Suppose we create a new table, and its two columns (`id` and `name`) have different column attribute function settings. This can be written as follows:

```
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',
  PRIMARY KEY (`id`)
);
```

In different database systems, it may appear in different forms. For example, in PostgreSQL, the function of `AUTO_INCREMENT` will be replaced by the `SERIAL` setting, and Oracle will use `IDENTITY`. Therefore, by focusing on what each function can do rather than "what each function is," you can become more adaptable to different database systems.

Create and Update Column

Columns are not static; just like data in the real world, they will change all the time. When we want to add or change columns, we can use the following statements (based on the preceding create table example).

Edit the table above and practice by implementing the following statements.

Create

```
ALTER TABLE `new_schema`.`users`
ADD COLUMN `age` INT NULL AFTER `name`;
```

This can be divided into two parts: 1. `ALTER TABLE`: When changing the column settings, we need to declare that we want to alter the table. 2. `ADD COLUMN`: Use the `ADD COLUMN` keyword, and then add the setting which be used when we create a table. Pay attention to that you can use the `AFTER` to set the column order, This increases readability when viewing the table column setting.

Update

In the SQL, updating a column is more like generate a new rule and overwrite original version, the statement is as follows:

```
ALTER TABLE `new_schema`.`users`
CHANGE COLUMN `id` `id` INT(11) NOT NULL AUTO_INCREMENT,
CHANGE COLUMN `name` `user_name` VARCHAR(45) NOT NULL DEFAULT 'No Name';
```

This statement uses `CHANGE COLUMN` and then adds new rules. It is worth noting that the grammar will first correspond to two consecutive words like `id` - `id` and `name` - `user_name`. The first is the existing column name, and the second is the new column name, followed by the new rules for the column.

So if you don't want to change the column name, it is necessary to keep the two consistent, like `id` `id` shown above.

The new table results will be as shown in the "Results" tab:

```
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',
  PRIMARY KEY (`id`)
);
```