第 3 章 数据表设计

学习目标：

- 熟悉数据类型的使用，能够区分 SQL 语句中不同类型数据的表示方式。
- 掌握数据表的相关约束的使用方法，能够在数据表中设置默认值约束、非空约束、唯一约束和主键约束。
- 掌握字段自动增长的设置，能够在创建数据表或修改数据表时为字段设置自动增长。
- 了解字符集和校对集的概念，能够说出字符集与校对集之间的联系。
- 掌握字符集和校对集的设置，能够设置服务器、数据库、数据表和字段的字符集和校对集。

在数据库中，数据表用来组织和存储各种数据，它是由表结构和数据组成的。在设计表结构时，经常需要根据实际需求，选择合适的数据类型、约束、字符集和校对集，以及为主键字段设置自动增长。本章围绕数据类型、表的约束、自动增长、字符集与校对集进行详细讲解。

3.1 数据类型

使用 MySQL 存储数据时，不同的数据类型决定了 MySQL 存储数据方式的不同。MySQL 提供了多种数据类型，其中主要包括数值类型、日期和时间类型、字符串类型。只有掌握这些数据类型的用法，才能正确编写 SQL 语句。作为一名合格的数据库开发工程师，要建立规范编写 SQL 语句的意识，养成规范化书写的好习惯。本节针对这些数据类型进行详细讲解。

3.1.1 数值类型

现实生活中有各种各样的数字，如考试成绩、商品价格等。在 MySQL 中，如果希望保存数字，可以将数字保存为数值类型，这样可以很方便地进行数学计算，而如果将数字保存为字符串类型，则不利于数学计算。数值类型主要包括整数类型、浮点数类型、定点数类型、BIT(位)类型，下面分别进行讲解。

1. 整数类型

整数类型用于保存整数，根据取值范围的不同，整数类型主要包括 TINYINT、SMALLINT、MEDIUMINT、INT 和 BIGINT。整数类型又分为无符号(UNSIGNED)和有

符号(SIGNED)两种情况,无符号不能保存负数,而有符号可以保存负数。整数类型的字节数和取值范围如表 3-1 所示。

表 3-1 整数类型的字节数和取值范围

类 型 名	字 节 数	无符号数取值范围	有符号数取值范围
TINYINT	1	0～255	-128～127
SMALLINT	2	0～65 535	-32 768～32 767
MEDIUMINT	3	0～16 777 215	-8 388 608～8 388 607
INT	4	0～4 294 967 295	-2 147 483 648～2 147 483 647
BIGINT	8	0～$2^{64}-1$	-2^{63}～$2^{63}-1$

从表 3-1 中可以看出,不同整数类型所占用的字节数和取值范围都是不同的。其中,占用字节数最小的是 TINYINT,占用字节数最大的是 BIGINT。不同整数类型的取值范围可以根据字节数计算出来,如 TINYINT 类型的整数占用 1 字节,1 字节是 8 位,那么 TINYINT 类型无符号数的最大值就是 2^8-1(即 255),有符号数的最大值就是 2^7-1(即 127)。同理,可以算出其他不同整数类型的取值范围。

需要注意的是,整数类型默认情况下是有符号的,如果使用无符号数据类型,需要通过 UNSIGNED 关键字修饰数据类型。例如,描述数据表中的 age(年龄)字段,可以使用“age TINYINT UNSIGNED”表明 age 字段是无符号的 TINYINT 类型。

在实际应用中选择数据类型时的注意事项如下。

(1) 若一个数据将来可能参与数学计算,推荐保存为整数、浮点数或定点数类型;若只用来显示,则推荐保存为字符串类型。例如,商品库存可能需要进行增加、减少或求和等操作,可以保存为整数类型;用户的身份证号、电话号码一般不需要计算,可以保存为字符串类型。

(2) 数据表的主键推荐使用整数类型,与字符串相比,整数类型的处理效率更高,查询速度更快。

(3) 当插入值的数据类型与字段的数据类型不一致,或使用 ALTER TABLE 修改字段的数据类型时,MySQL 会尝试尽可能将现有的值转换为新类型。例如,字符串'123'、'-123'、'1.23'与数字 123、-123、1.23 可以互相转换;1.5 转换为整数时,会被四舍五入,结果为 2。

下面通过案例演示整数类型的使用及注意事项,具体示例如下。

(1) 在 dms 数据库中创建 my_int 数据表,选取 INT 和 TINYINT 两种类型测试,具体 SQL 语句及执行结果如下。

```
mysql>USE dms;
Database changed
mysql>CREATE TABLE my_int (
    ->  int_1 INT,
    ->  int_2 INT UNSIGNED,
    ->  int_3 TINYINT,
    ->  int_4 TINYINT UNSIGNED
    -> );
Query OK, 0 rows affected (0.04 sec)
```

上述 SQL 语句中，定义 int_1 字段的数据类型为有符号的 INT 类型；定义 int_2 字段的数据类型为无符号的 INT 类型；定义 int_3 字段的数据类型为有符号的 TINYINT 类型；定义 int_4 字段的数据类型为无符号的 TINYINT 类型。

（2）添加数据进行测试。当数值在合法的取值范围内时，可以正确添加，反之则添加失败，并提示错误信息，具体 SQL 语句及执行结果如下。

```
#添加成功测试
mysql>INSERT INTO my_int VALUES (1000, 1000, 100, 100);
Query OK, 1 row affected (0.00 sec)
#添加失败测试
mysql>INSERT INTO my_int VALUES (1000, -1000, 100, 100);
ERROR 1264 (22003): Out of range value for column 'int_2' at row 1
```

从上述执行结果可以看出，由于-1000 超出了无符号 INT 类型的取值范围，所以导致数据添加失败，并提示 int_2 字段超出取值范围的错误信息。

2. 浮点数类型

浮点数类型用于保存小数。浮点数类型有两种，分别是 FLOAT（单精度浮点数）和 DOUBLE（双精度浮点数）。DOUBLE 的精度比 FLOAT 高，但是 DOUBLE 消耗的内存是 FLOAT 的两倍，DOUBLE 的运算速度比 FLOAT 慢。

对于 FLOAT 类型，当一个数字的整数部分和小数部分加起来超过 6 位时就有可能损失精度；对于 DOUBLE 类型，当一个数字的整数部分和小数部分加起来超过 15 位时就有可能损失精度。浮点数在进行数学计算时可能会损失精度。因此，浮点数类型适合将小数作为近似值存储而不是作为精确值存储。

为了帮助读者更好地理解，下面选取单精度浮点数进行演示，具体示例如下。

（1）创建数据表 my_float，具体 SQL 语句及执行结果如下。

```
mysql>CREATE TABLE my_float (
    ->  f1 FLOAT,
    ->  f2 FLOAT
    -> );
Query OK, 0 rows affected (0.01 sec)
```

上述 SQL 语句中，在创建数据表 my_float 时，定义 f1 字段和 f2 字段的数据类型都为 FLOAT。

（2）添加数据进行测试。添加未超出精度的数据，具体 SQL 语句及执行结果如下。

```
#第 1 条语句
mysql>INSERT INTO my_float VALUES (111111, 1.11111);
Query OK, 1 row affected (0.00 sec)
```

（3）添加超出精度的数据，具体 SQL 语句及执行结果如下。

```
#第 2 条语句
mysql>INSERT INTO my_float VALUES (1111111, 1.111111);
```

```
Query OK, 1 row affected (0.00 sec)
#第 3 条语句
mysql>INSERT INTO my_float VALUES (1111114, 1111115);
Query OK, 1 row affected (0.00 sec)
#第 4 条语句
mysql>INSERT INTO my_float VALUES (11111149, 11111159);
Query OK, 1 row affected (0.00 sec)
```

(4) 查询 my_float 数据表中的数据,具体 SQL 语句及执行结果如下。

```
mysql>SELECT * FROM my_float;
+----------+----------+
| f1       | f2       |
+----------+----------+
| 111111   | 1.11111  |
| 1111110  | 1.11111  |
| 1111110  | 1111120  |
| 11111100 | 11111200 |
+----------+----------+
4 rows in set (0.00 sec)
```

从上述执行结果可以看出,第一条语句添加的数据没有超出精度,数据原样输出,结果为 111111 和 1.11111;第二条语句中 f1 字段的第 7 位为 1,四舍五入后为 0,f2 字段的第 7 位为 1,四舍五入后为 0,结果为 1111110 和 1.11111;第三条语句中 f1 字段的第 7 位为 4,四舍五入后为 0,f2 字段的第 7 位为 5,四舍五入后进位,结果为 1111110 和 1111120;第四条语句的 f1 和 f2 字段都为 8 位,第 8 位被忽略,第 7 位四舍五入,结果为 11111100 和 11111200。

3. 定点数类型

定点数类型用于保存确切精度的小数,如金额。MySQL 中的定点数类型使用 DECIMAL 或 NUMERIC 表示,两者被视为相同的类型。以 DECIMAL 为例,定点数类型的定义方式如下。

```
DECIMAL(M,D)
```

上述定义中,M 表示整数部分加小数部分的总长度,取值范围为 0～65,默认值为 10,超出范围会报错;D 表示小数点后可存储的位数,取值范围为 0～30,默认值为 0,且必须满足 D≤M。例如,DECIMAL(5,2)表示能够存储总长度为 5,并且包含 2 位小数的任何值,它的取值范围是-999.99～999.99,系统会自动根据存储的数据来分配存储空间。若不允许保存负数,可通过 UNSIGNED 关键字将定点数类型修饰为无符号数据类型。

为了帮助读者更好地理解,下面以 DECIMAL 定点数类型进行测试,具体示例如下。

(1) 创建数据表 my_decimal,具体 SQL 语句及执行结果如下。

```
mysql>CREATE TABLE my_decimal (
    ->  d1 DECIMAL(4,2),
```

```
    ->  d2 DECIMAL(4,2)
    ->);
Query OK, 0 rows affected (0.01 sec)
```

上述 SQL 语句中，在创建数据表 my_decimal 时，定义字段名 d1、d2，数据类型都为 DECIMAL，DECIMAL(4,2)表示的取值范围是-99.99～99.99。

(2) 添加数据进行测试，当添加的小数部分超出范围时，会四舍五入并出现警告，具体 SQL 语句及执行结果如下。

```
mysql>INSERT INTO my_decimal VALUES (1.234, 1.235);
Query OK, 1 row affected, 2 warnings (0.00 sec)
#查看警告
mysql>SHOW WARNINGS;
+--------+------+-----------------------------------------+
| Level  | Code | Message                                 |
+--------+------+-----------------------------------------+
| Note   | 1265 | Data truncated for column 'd1' at row 1 |
| Note   | 1265 | Data truncated for column 'd2' at row 1 |
+--------+------+-----------------------------------------+
2 rows in set (0.00 sec)
#查询结果
mysql>SELECT * FROM my_decimal;
+------ +------+
| d1     | d2   |
+------ +------+
| 1.23   | 1.24 |
+------ +------+
1 row in set (0.00 sec)
```

上述 SQL 语句中，由于 DECIMAL(4,2)只能保存小数点后 2 位，第一个值 1.234 的小数点后第 3 位为 4，四舍五入后为 0，结果为 1.23；第二个值 1.235 的小数点后第 3 位为 5，四舍五入后进位，结果为 1.24。从警告信息可以看出，因为小数部分超出范围，出现了 Data truncated(数据截断)警告。

(3) 添加数据进行测试，当添加的小数部分四舍五入导致整数部分进位时，会插入失败，具体 SQL 语句及执行结果如下。

```
mysql>INSERT INTO my_decimal VALUES (99.99, 99.999);
ERROR 1264 (22003): Out of range value for column 'd2' at row 1
```

上述 SQL 语句中，第一个值 99.99 在 DECIMAL(4,2)的取值范围内，第二个值 99.999 的小数点后第三位为 9，四舍五入后进位，结果为 100.00，整数部分超出了取值范围，因此数据插入失败，出现 Out of range value(超出取值范围)错误。

4. BIT(位)类型

BIT(位)类型用于存储二进制数据，该类型的定义方式如下。

```
BIT(M)
```

上述定义中,M 表示位数,范围为 1～64。

下面演示如何用 BIT(位)类型字段保存字符 A。为了方便演示,本案例会用到 MySQL 中的 ASCII()、BIN()和 LENGTH()函数。其中,ASCII()函数用于查看指定字符的 ASCII 码,BIN()函数用于将十进制数转换为二进制数,LENGTH()函数用于获取字符串长度。首先查询出字符 A 的二进制和长度,然后根据获取到的长度创建数据表,并将字符 A 的 ASCII 码值添加到数据表中,最后将数据表中的查询结果转为二进制数字显示,具体步骤如下。

(1) 查询字符 A 的 ASCII 码值,将获取的 ASCII 码值转换为二进制,并计算出长度,具体 SQL 语句及执行结果如下。

```
mysql>SELECT ASCII('A');
+-------------+
| ASCII('A')  |
+-------------+
| 65          |
+-------------+
1 row in set (0.00 sec)
```

上述 SQL 语句中,使用 ASCII()函数查看字符 A 的 ASCII 码;从执行结果可以看出,字符 A 的 ASCII 码值为 65。

(2) 将获取到的字符 A 的 ASCII 码值转换为二进制,并计算出长度,具体 SQL 语句及执行结果如下。

```
mysql>SELECT BIN(65), LENGTH(BIN(65));
+---------+------------------+
| BIN(65) | LENGTH(BIN(65))  |
+---------+------------------+
| 1000001 | 7                |
+---------+------------------+
1 row in set (0.01 sec)
```

上述 SQL 语句中,使用 BIN()函数将 65 转换为二进制数,使用 LENGTH()函数获取字符串长度。从执行结果可以看出,字符 A 对应的二进制数为 1000001,长度为 7。

(3) 创建数据表 my_bit,并添加数据,具体 SQL 语句及执行结果如下。

```
#创建数据表 my_bit
mysql>CREATE TABLE my_bit (b BIT(7));
Query OK, 0 rows affected (0.03 sec)
#添加数据
mysql>INSERT INTO my_bit VALUES (65);
Query OK, 1 row affected (0.01 sec)
```

上述 SQL 语句中,在创建数据表 my_bit 时,设置字段名为 b,数据类型为 BIT,并且存储位数为 7;向数据表 my_bit 添加值为 65 的一条记录。

（4）查询数据，验证数据是否添加成功，具体 SQL 语句及执行结果如下。

```
mysql>SELECT * FROM my_bit;
+-------------+
| b           |
+-------------+
| 0x41        |
+-------------+
1 row in set (0.00 sec)
```

从上述执行结果可以看出，b 字段的值为十六进制数 0x41。

（5）将十六进制数转换为二进制数显示，具体 SQL 语句及执行结果如下。

```
mysql>SELECT BIN(b) FROM my_bit;
+----------+
| BIN(b)   |
+----------+
| 1000001  |
+----------+
1 row in set (0.00 sec)
```

从上述结果可以看出，b 字段的值转换为二进制数的结果为 1000001。

多学一招：MySQL 中的直接常量

直接常量是指在 MySQL 中直接编写的字面常量，如数字 123、字符串'abc'等，常用于在 INSERT 语句中编写插入的数据。直接常量有多种语法形式，具体如下。

（1）十进制数。十进制数语法近似于日常生活中的数字，如 123、1.23、-1.23，以及科学记数法 1E2、1E-2(E 不分大小写)。

（2）二进制数。在二进制字符串前加前缀 b，形如“b'1000001'”。通过“SELECT b'1000001';”语句可以查看二进制数转换为十六进制数的结果，即 0x41。

（3）十六进制数。十六进制数有两种表示方式，形如“x'41'”和“0x41”。其中，十六进制数 41 对应十进制数 65。通过“SELECT HEX(65);”可以查看十进制数 65 转换为十六进制数的结果，即 41。

（4）字符串。MySQL 支持单引号和双引号定界符，如'abc'和"abc"。若要在单引号或双引号字符串中书写单引号或双引号，则需要在单引号或双引号前面加上反斜线“\”转义，即“\'”和“\"”，这种方式称为转义字符。常用的转义字符如表 3-2 所示。

表 3-2　常用的转义字符

转义字符	含　义	转义字符	含　义
\0	空字符(NUL)	\t	制表符(HT)
\r	回车符(CR)	\b	退格(BS)
\n	换行符(LF)	\'	单引号
\"	双引号	\%	%(常用于 LIKE 条件)
\\	反斜线	_	_(常用于 LIKE 条件)

表 3-2 中的转义字符区分大小写,例如,\b 会被当成退格符,但\B 则被当成字符 B。

当字符串用双引号定界符时,该字符串中的单引号不需要转义;同理,当字符串用单引号定界符时,该字符串中的双引号不需要转义。

(5) 布尔值。布尔值有 TRUE 和 FALSE 两个(不区分大小写),通常用于逻辑判断,表示事物的"真"和"假"。在 SELECT、INSERT 等语句中使用布尔值时,TRUE 会转换为 1,FALSE 会转换为 0。

(6) NULL 值。NULL 值通常用来表示没有值、值不确定等含义。例如,在插入一条商品数据时,暂时不知道该商品的库存量,可将库存量设为 NULL,以后再修改。

3.1.2 日期和时间类型

为了方便在数据库中存储日期和时间,MySQL 提供了一些表示日期和时间的数据类型,分别是 YEAR、DATE、TIME、DATETIME 和 TIMESTAMP。MySQL 中的日期和时间类型如表 3-3 所示。

表 3-3 日期和时间类型

类型名	字节数	范 围	格 式	描 述
YEAR	1	1901～2155	YYYY	年份值
DATE	3	1000-01-01～9999-12-31	YYYY-MM-DD	日期值
TIME	3	-838:59:59～838:59:59	HH:MM:SS	时间值或持续时间
DATETIME	8	1000-01-01 00:00:00～9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	日期和时间值
TIMESTAMP	4	1970-01-01 00:00:01～2038-01-19 03:14:07	YYYY-MM-DD HH:MM:SS	日期和时间值,保存为时间戳

在表 3-3 中,日期格式 YYYY-MM-DD 中的 YYYY 表示年,MM 表示月,DD 表示日;时间格式 HH:MM:SS 中的 HH 表示小时,MM 表示分钟,SS 表示秒数。

下面针对不同的日期和时间类型分别进行讲解。

1. YEAR 类型

YEAR 类型用于存储年份数据。在 MySQL 中,可以使用以下 3 种格式指定 YEAR 类型的值。

(1) 4 位字符串或数字,可表示的年份范围为'1901'～'2155'或 1901～2155。例如,输入'2022'或 2022,插入数据库中的值均为 2022。

(2) 两位字符串'00'～'99'。其中,'00'～'69'的值会被转换为 2000—2069 的年份,'70'～'99'的值会被转换为 1970—1999 的年份。例如,输入'22',插入数据库中的值为 2022。

(3) 数字 1～99。其中,1～69 的值会被转换为 2001—2069 的年份,70～99 的值会被转换为 1970—1999 的年份。例如,输入 22,插入数据库中的值为 2022。

下面演示 YEAR 类型的使用,具体示例如下。

```
#创建数据表
mysql>CREATE TABLE my_year (y YEAR);
```

```
#添加年份数据
mysql>INSERT INTO my_year VALUES (2022), ('22'), (22);
Query OK, 3 rows affected (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 0
#查询数据
mysql>SELECT * FROM my_year;
+-------+
| y     |
+-------+
| 2022  |
| 2022  |
| 2022  |
+-------+
3 rows in set (0.00 sec)
```

上述 SQL 语句中，创建了数据表 my_year，设置 y 字段的数据类型为 YEAR；由查询结果可知，当添加年份数据为 2022、'22'或 22 时，输出结果都为 2022。

2. DATE 类型

DATE 类型用于存储日期数据，通常用于保存年、月、日，日期数据中的分隔符“-”也可以使用“.”“,”“/”等符号替代。在 MySQL 中，可以使用以下 4 种格式指定 DATE 类型的值。

（1）字符串'YYYY-MM-DD'或者'YYYYMMDD'。例如，输入'2022-01-02'或'20220102'，插入数据库中的日期都为 2022-01-02。

（2）字符串'YY-MM-DD'或者'YYMMDD'。YY 表示的是年，范围为'00'～'99'，其中'00'～'69'的值会被转换为 2000～2069 的值，'70'～'99'的值会被转换为 1970～1999 的值。例如，输入'22-01-02'或'220102'，插入数据库中的日期都为 2022-01-02。

（3）数字 YYMMDD。例如，输入 220102，插入数据库中的日期为 2022-01-02。

（4）使用 CURRENT_DATE 或者 NOW()表示当前系统日期。如需查看当前系统日期，可通过“SELECT CURRENT_DATE;”或“SELECT NOW();”进行查看。

下面演示 DATE 类型的使用，具体示例如下。

```
#创建数据表
mysql>CREATE TABLE my_date (d DATE);
#添加日期数据
mysql>INSERT INTO my_date VALUES ('2022-01-02');
Query OK, 1 row affected (0.01 sec)
mysql>INSERT INTO my_date VALUES (CURRENT_DATE);
Query OK, 1 row affected (0.01 sec)
mysql>INSERT INTO my_date VALUES (NOW());
Query OK, 1 row affected (0.01 sec)
#查询数据
mysql>SELECT * FROM my_date;
```

```
+-------------+
| d           |
+-------------+
| 2022-01-02  |
| 2022-08-16  |
| 2022-08-16  |
+-------------+
3 rows in set (0.00 sec)
```

上述 SQL 语句中,创建了数据表 my_date,设置 d 字段的数据类型为 DATE;添加日期数据'2022-01-02'、CURRENT_DATE 和 NOW()。由查询结果可知,当添加的日期为'2022-01-02'时,输出结果为 2022-01-02;当添加的日期为 CURRENT_DATE 和 NOW()时,输出结果为当前系统日期。

3. TIME 类型

TIME 类型用于存储时间数据,通常用于保存时、分、秒。在 MySQL 中,可以使用以下 3 种格式指定 TIME 类型的值。

(1) 字符串'HHMMSS'或者数字 HHMMSS。例如,输入'345454'或 345454,插入数据库中的时间为 34:54:54(34 小时 54 分 54 秒)。

(2) 字符串'D HH:MM:SS'。其中,D 表示日,可以取 0～34,插入数据时,小时的值等于(D×24+HH)。例如,输入'2 11:30:50',插入数据库中的时间为 59:30:50;输入'11:30:50',插入数据库中的时间为 11:30:50;输入'34 22:59:59',插入数据库中的时间为 838:59:59。

(3) 使用 CURRENT_TIME 或 NOW()表示当前系统时间。

下面演示 TIME 类型的使用,具体示例如下。

```
#创建数据表
mysql>CREATE TABLE my_time (t TIME);
Query OK, 0 rows affected (0.02 sec)
#添加时间数据
mysql>INSERT INTO my_time VALUES ('345454');
Query OK, 1 row affected (0.00 sec)
mysql>INSERT INTO my_time VALUES ('2 11:30:50');
Query OK, 1 row affected (0.00 sec)
mysql>INSERT INTO my_time VALUES (CURRENT_TIME);
Query OK, 1 row affected (0.00 sec)
mysql>INSERT INTO my_time VALUES (NOW());
Query OK, 1 row affected (0.00 sec)
#查询数据
mysql>SELECT * FROM my_time;
+-----------+
| t         |
```

```
+----------+
| 34:54:54     |
| 59:30:50     |
| 10:14:11     |
| 10:14:11     |
+----------+
4 rows in set (0.00 sec)
```

上述 SQL 语句中，创建了数据表 my_time，设置 t 字段的数据类型为 TIME；添加时间数据'345454'、'2 11:30:50'、CURRENT_TIME 和 NOW()。由查询结果可知，当添加的时间为'345454'时，输出结果为 34:54:54；当添加的时间为'2 11:30:50'时，输出结果为 59:30:50；当添加的时间为 CURRENT_DATE 和 NOW()时，输出结果为当前系统时间。

4. DATETIME 类型

DATETIME 类型用于存储日期和时间数据，通常用于保存年、月、日、时、分、秒。在 MySQL 中，可以使用以下 4 种格式指定 DATETIME 类型的值。

（1）字符串'YYYY-MM-DD HH:MM:SS'或者'YYYYMMDDHHMMSS'。例如，输入字符串'2022-01-22 09:01:23'或'20220122090123'，插入数据库中的 DATETIME 值都为 2022-01-22 09:01:23。

（2）字符串'YY-MM-DD HH:MM:SS'或者'YYMMDDHHMMSS'。其中 YY 表示年，取值范围为'00'～'99'。与 DATE 类型中的 YY 相同，'00'～'69'的值会被转换为 2000～2069 的值，'70'～'99'的值会被转换为 1970～1999 的值。

（3）数字 YYYYMMDDHHMMSS 或者 YYMMDDHHMMSS。例如，输入数字 20220122090123 或者 220122090123，插入数据库中的 DATETIME 值都为 2022-01-22 09:01:23。

（4）使用 NOW()表示当前系统的日期和时间。

下面演示 DATETIME 类型的使用，具体示例如下。

```
#创建数据表
mysql>CREATE TABLE my_datetime (d DATETIME);
Query OK, 0 rows affected (0.02 sec)
#添加日期和时间数据
mysql>INSERT INTO my_datetime VALUES ('2022-01-22 09:01:23');
Query OK, 1 row affected (0.01 sec)
mysql>INSERT INTO my_datetime VALUES (NOW());
Query OK, 1 row affected (0.00 sec)
#查询数据
mysql>SELECT * FROM my_datetime;
+----------------------+
| d                    |
+----------------------+
| 2022-01-22 09:01:23  |
| 2022-08-16 10:22:13  |
+----------------------+
2 rows in set (0.00 sec)
```

上述 SQL 语句中,创建了数据表 my_datetime,设置 d 字段的数据类型为 DATETIME;添加日期和时间数据'2022-01-22 09:01:23'和 NOW()。由查询结果可知,当添加的日期和时间为'2022-01-22 09:01:23'时,输出结果为 2022-01-22 09:01:23;当添加的日期和时间为 NOW()时,输出结果为当前系统的日期和时间。

5. TIMESTAMP 类型

TIMESTAMP 类型用于存储日期和时间数据,它的格式与 DATETIME 类似,都用于存储日期和时间数据。在使用时,TIMESTAMP 类型与 DATATIME 类型存在一些区别,具体如下。

(1) TIMESTAMP 类型的取值范围比 DATATIME 类型小。

(2) TIMESTAMP 类型的值和时区有关,如果插入的日期时间为 TIMESTAMP 类型,系统会根据当前系统所设置的时区,对日期时间进行转换后存放;从数据库中取出 TIMESTAMP 类型的数据时,系统会将数据转换为对应时区的时间后显示。因此,TIMESTAMP 类型可能会导致两个不同时区的环境下取出的同一个日期和时间的显示结果不同。

(3) TIMESTAMP 类型可以使用 CURRENT_TIMESTAMP 表示系统当前日期和时间。

下面演示 TIMESTAMP 类型的使用,具体示例如下。

```
#创建数据表
mysql>CREATE TABLE my_timestamp (t TIMESTAMP);
Query OK, 0 rows affected (0.02 sec)
#添加当前系统日期和时间
mysql>INSERT INTO my_timestamp VALUES (CURRENT_TIMESTAMP);
Query OK, 1 row affected (0.01 sec)
#查询数据
mysql>SELECT * FROM my_timestamp;
+---------------------+
| t                   |
+---------------------+
| 2022-08-16 10:29:22 |
+---------------------+
1 row in set (0.00 sec)
```

上述 SQL 语句中,创建了数据表 my_timestamp,设置 t 字段的数据类型为 TIMESTAMP;添加当前系统日期和时间数据 CURRENT_TIMESTAMP。由查询结果可知,输出结果为当前系统的日期和时间。

3.1.3 字符串类型

对于一些文本信息类的数据,如姓名、家庭住址等,在 MySQL 中适合保存为字符串类型。MySQL 中常用的字符串类型如表 3-4 所示。

表 3-4　字符串类型

类　型　名	类 型 说 明
CHAR	固定长度的字符串
VARCHAR	可变长度的字符串
BINARY	固定长度的二进制字符串
VARBINARY	可变长度的二进制字符串
BLOB	普通二进制数据
TEXT	普通文本数据
ENUM	枚举类型
SET	字符串对象，可以有零个或多个值

接下来，针对各字符串类型进行详细介绍，具体如下。

1. CHAR 和 VARCHAR

CHAR 和 VARCHAR 类型的字段用于存储字符串数据，CHAR 类型的字段用于存储固定长度的字符串，其中固定长度可以是 0～255 中的任意整数值；VARCHAR 类型的字段用于存储可变长度的字符串，其中可变长度可以是 0～65535 中的任意整数值。

在 MySQL 中，定义 CHAR 类型的方式如下。

```
CHAR(M)
```

上述定义方式中，M 指的是字符串的最大长度。CHAR 类型的字段会根据 M 分配存储空间，无论有没有被存满，都会占用存满时的存储空间。

在 MySQL 中，定义 VARCHAR 类型的方式如下。

```
VARCHAR(M)
```

上述定义方式中，M 指的是字符串的最大长度。VARCHAR 类型的字段会根据实际保存的字符个数来决定实际占用的存储空间。例如，VARCHAR(255)表示最多可以保存 255 个字符，在 UTF-8 字符集下，当保存 255 个中文字符时，这些中文字符占用 255×3=765 字节，此外，VARCHAR 还会多占用 1～3 字节来存储一些额外的信息。

为了对比 CHAR 和 VARCHAR 之间的区别，下面以 CHAR(4)和 VARCHAR(4)为例进行比较，如表 3-5 所示。

表 3-5　比较 CHAR(4)和 VARCHAR(4)

添 加 值	CHAR(4)值	存储空间	VARCHAR(4)值	存 储 空 间
''	'　　'	4 字节	''	1 字节
'ab'	'ab　'	4 字节	'ab'	3 字节
'abcd'	'abcd'	4 字节	'abcd'	5 字节
'abcdefgh'	'abcd'	4 字节	'abcd'	5 字节

从表 3-5 可以看出,对于 CHAR(4),无论添加值的长度是多少,所占用的存储空间都是 4 字节,而 VARCHAR(4)占用的字节数为实际长度加 1。

需要注意的是,在向 CHAR 和 VARCHAR 类型的字段中插入字符串时,如果插入的字符串尾部存在空格,CHAR 类型的字段会去除空格后进行存储,而 VARCHAR 类型的字段会保留空格完整地存储字符串。

2. BINARY 和 VARBINARY

BINARY 和 VARBINARY 类型类似于 CHAR 和 VARCHAR,不同之处在于 BINARY 和 VARBINARY 类型存储的是二进制字符串。

在 MySQL 中,定义 BINARY 类型的方式如下。

```
BINARY(M)
```

上述定义中,M 是指二进制数据的最大字节长度。BINARY(M)类型的长度是固定的,如果未指定 M,表示只能存储 1 字节。例如,BINARY(8)表示最多能存储 8 字节。如果字段值不足 M 字节,将在数据的后面填充 0 以补齐指定长度。

在 MySQL 中,定义 VARBINARY 类型的方式如下。

```
VARBINARY(M)
```

上述定义中,M 是指二进制数据的最大字节长度。VARBINARY 类型必须指定 M,否则会报错。此外,VARBINARY 除了存储数据本身外,还会多占用 1～2 字节来存储一些额外的信息。

下面演示 BINARY 和 VARBINARY 类型的使用,具体示例如下。

(1) 创建数据表 my_binary,准备 f1、f2 和 f3 这 3 个字段进行测试,具体 SQL 语句及执行结果如下。

```
mysql>CREATE TABLE my_binary (
    ->  f1 BINARY,
    ->  f2 BINARY(3),
    ->  f3 VARBINARY(10)
    -> );
```

(2) 添加数据进行测试,当添加超出存储的最大字节长度时,数据添加失败,具体 SQL 语句及执行结果如下。

```
mysql>INSERT INTO my_binary (f1) VALUES ('我');
ERROR 1406 (22001): Data too long for column 'f1' at row 1
```

上述 SQL 语句中,因为添加的是 1 个汉字,1 个汉字大于 1 字节,而 f1 只能存储 1 字节,所以会添加失败。

(3) 添加数据进行测试,当添加未超出存储的最大字节长度时,数据添加成功,具体 SQL 语句及执行结果如下。

```
#第 1 条语句
mysql> INSERT INTO my_binary (f1, f2) VALUES ('a', 'a');
Query OK, 1 row affected (0.00 sec)
#第 2 条语句
mysql> INSERT INTO my_binary (f2) VALUES ('我');
Query OK, 1 row affected (0.01 sec)
#第 3 条语句
mysql> INSERT INTO my_binary (f1, f2) VALUES ('a', 'abc');
Query OK, 1 row affected (0.01 sec)
#第 4 条语句
mysql> INSERT INTO my_binary (f2, f3) VALUES ('ab', 'ab');
Query OK, 1 row affected (0.01 sec)
```

上述 SQL 语句中，第一条语句中的 f1 和 f2 的值都为一个字母，因为一个字母占 1 字节，f1 长度为 1 字节，f2 长度为 3 字节，所以都添加成功；第二条语句中的值为汉字，f2 长度能够容纳一个汉字，所以添加成功；第三条语句中的 f1 值为一个字母，f2 值为 3 个字母，因为一个字母占 1 字节，f1 长度是 1 字节，f2 长度是 3 字节，所以都添加成功；第四条语句的 f2 值和 f3 值都为 2 个字母，因为一个字母占 1 字节，f2 长度是 3 字节，f3 长度是 10 字节，所以都添加成功。

3. TEXT 系列

TEXT 系列的数据类型包括 TINYTEXT、TEXT、MEDIUMTEXT 和 LONGTEXT，通常用于存储文章内容、评论等较长的字符串。TEXT 系列类型具体如表 3-6 所示。

表 3-6　TEXT 系列类型

数 据 类 型	类型说明	存 储 范 围
TINYTEXT	短文本数据	0～L+1 字节，其中 $L<2^8$
TEXT	普通文本数据	0～L+2 字节，其中 $L<2^{16}$
MEDIUMTEXT	中等文本数据	0～L+3 字节，其中 $L<2^{24}$
LONGTEXT	超大文本数据	0～L+4 字节，其中 $L<2^{32}$

在表 3-6 中，L 表示给定字符串值的实际长度（以字节为单位）。

4. BLOB 系列

BLOB 系列的数据类型包括 TINYBLOB、BLOB、MEDIUMBLOB 和 LONGBLOB，通常用于存储图片、PDF 文档、音频和视频等二进制数据，BLOB 类型具体如表 3-7 所示。

表 3-7　BLOB 类型

数据类型	类型描述	存储范围
TINYBLOB	短二进制数据	0～L+1 字节，其中 $L<2^8$
BLOB	普通二进制数据	0～L+2 字节，其中 $L<2^{16}$

续表

数据类型	类型描述	存储范围
MEDIUMBLOB	中等二进制数据	0～L+3 字节,其中 $L<2^{24}$
LONGBLOB	超大二进制数据	0～L+4 字节,其中 $L<2^{32}$

需要注意的是,BLOB 类型的数据是根据二进制编码进行比较和排序的,而 TEXT 类型的数据是根据文本模式进行比较和排序的。

下面演示 BLOB 系列数据类型的使用,具体示例如下。

(1) 创建数据表 my_blob,具体 SQL 语句及执行结果如下。

```
#创建数据表
mysql>CREATE TABLE my_blob (
    ->  id INT,
    ->  img MEDIUMBLOB
    -> );
Query OK, 0 rows affected (0.07 sec)
```

上述 SQL 语句中,在创建数据表 my_blob 时,设置字段名为 id,数据类型为 INT;设置字段名为 img,数据类型为 MEDIUMBLOB。

(2) 添加数据进行测试,具体 SQL 语句及执行结果如下。

```
mysql>INSERT INTO my_blob (id) VALUES (9001);
Query OK, 1 row affected (0.00 sec)
```

上述 SQL 语句中,向数据表 my_blob 添加 id 值为 9001 的一条数据。

(3) 查询数据,具体 SQL 语句及执行结果如下。

```
mysql>SELECT * FROM my_blob;
+------+-------------+
| id   | img         |
+------+-------------+
| 9001 | NULL        |
+------+-------------+
1 row in set (0.00 sec)
```

从上述执行结果可以看出,查询出了一条 id 为 9001、img 为 NULL 的记录。

5. ENUM 类型

ENUM 类型又称为枚举类型,占用 1～2 字节的存储空间,当 ENUM 类型包含 1～255 个成员时,需要 1 字节的存储空间;当 ENUM 类型包含 256～65535 个成员时,需要 2 字节的存储空间。枚举列表中的每个成员都有一个索引值,索引值从 1 开始,依次递增。

在 MySQL 中,定义 ENUM 类型的方式如下。

```
ENUM('值 1', '值 2', '值 3', …, '值 n')
```

上述定义方式中，('值 1', '值 2', '值 3', …, '值 n')称为枚举列表，该列表中的每一项，称为成员，ENUM 类型的数据只能从成员中选取单个值，不能一次选取多个值。枚举列表最多可以有 65535 个值，每个值都有一个顺序编号，实际保存在记录中的是顺序编号，而不是列表中的值，因此不必担心过长的值占用空间。但在使用 SELECT、INSERT 等语句进行操作时，仍然使用列表中的值，而不是使用顺序编号值。

下面演示 ENUM 类型的使用，具体示例如下。

(1) 创建数据表 my_enum，具体 SQL 语句及执行结果如下。

```
mysql>CREATE TABLE my_enum (gender ENUM('male', 'female'));
Query OK, 0 rows affected (0.04 sec)
```

上述 SQL 语句中，在创建数据表 my_enum 时，设置字段名为 gender，数据类型为 ENUM，枚举列表中包含 male 和 female 两个成员。

(2) 添加两条测试数据，并查询数据是否添加成功，具体 SQL 语句及执行结果如下。

```
#添加数据
mysql>INSERT INTO my_enum VALUES ('male'), ('female');
Query OK, 1 row affected (0.01 sec)
#查询数据，查询结果为 female
mysql>SELECT * FROM my_enum WHERE gender='female';
+---------+
| gender  |
+---------+
| female  |
+---------+
1 row in set (0.01 sec)
```

上述 SQL 语句中，向数据表 my_enum 添加值为 male、female 的两条记录；并使用 SELECT 语句按要求查询出了一条 gender 为 female 的记录。

(3) 添加枚举列表中不存在的值进行测试，具体 SQL 语句及执行结果如下。

```
mysql>INSERT INTO my_enum VALUES('m');
ERROR 1265 (01000): Data truncated for column 'gender' at row 1
```

从上述执行结果可以看出，当添加枚举列表中不存在的值时，会提示错误信息。

6. SET 类型

SET 类型用于保存字符串对象，可以有零个或多个值，每个值都必须从创建表时指定的允许值列表中选择，其定义格式与 ENUM 类型类似，定义 SET 类型的方式如下。

```
SET('值 1', '值 2', '值 3', …, '值 n')
```

上述定义方式中，SET 类型占用 1、2、3、4 或 8 字节，这取决于集合成员的数量，SET 类型的列表中最多可以有 64 个成员。

SET 与 ENUM 类型的区别在于，SET 类型可以从列表中选择一个或多个值来保存，多

个值之间用逗号“,”分隔,而 ENUM 类型只能从列表中选择一个值来保存。

SET 和 ENUM 类型的优势在于,规范了数据本身,限定只能添加规定的数据项,查询速度比 CHAR、VARCHAR 类型快,节省存储空间。

在使用 SET 类型与 ENUM 类型时的注意事项如下。

(1) ENUM 和 SET 类型列表中的值都可以使用中文,但必须设置支持中文的字符集。例如“CREATE TABLE my_enum (gender ENUM ('男', '女')) CHARSET=GBK;”。

(2) ENUM 和 SET 类型在填写列表、插入值、查找值等操作时,都会自动忽略末尾的空格。

下面演示 SET 类型的使用,具体示例如下。

(1) 创建数据表 my_set,具体 SQL 语句及执行结果如下。

```
mysql>CREATE TABLE my_set (hobby SET('book', 'game', 'code'));
Query OK, 0 rows affected (0.01 sec)
```

上述 SQL 语句中,在创建数据表 my_ set 时,设置字段名为 hobby,数据类型为 SET,SET 列表中包含 book、game 和 code 这 3 个成员。

(2) 添加 3 条测试记录,并查询数据是否添加成功,具体 SQL 语句及执行结果如下。

```
#添加数据
mysql>INSERT INTO my_set VALUES (''), ('book'), ('book,code');
Query OK, 3 rows affected (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 0
#查询数据,查询结果为"book,code"
mysql>SELECT * FROM my_set WHERE hobby='book,code';
+------------+
| hobby      |
+------------+
| book,code  |
+------------+
1 rows in set (0.00 sec)
```

需要注意的是,当添加重复的 SET 类型成员时,MySQL 会自动删除重复的成员。如果向 SET 类型的字段插入 SET 成员中不存在的值时,MySQL 会抛出错误。当添加没有被定义的值时,也会报错。

多学一招:JSON 数据类型

JSON 是一种轻量级的数据交换格式,由 JavaScript 语言发展而来。在 MySQL 5.7.8 版本中,开始提供了 JSON 数据类型;在 MySQL 8 版本中,JSON 类型提供了可以自动验证的 JSON 文档和优化的存储结构,使得在 MySQL 中存储和读取 JSON 类型的数据更加方便和高效。

MySQL 中 JSON 类型值常见的格式有两种,分别为 JSON 数组和 JSON 对象,示例如下。

```
#JSON 数组
["abc", 10, null, true, false]
#JSON 对象
{"k1": "value", "k2": 10}
```

从上述示例可知,JSON 数组使用“[”和“]”表示,多个值之间使用逗号分隔,例如,10

和 null 之间使用逗号分隔;JSON 对象使用“{”和“}”表示,保存的数据是一组键值对,例如,k1 和 k2 是键名(或称为属性名),value 和 10 是键名对应的值。

下面演示 JSON 数据类型的使用,具体示例如下。

(1) 创建数据表 my_json,具体 SQL 语句及执行结果如下。

```
mysql>CREATE TABLE my_json (j1 JSON, j2 JSON);
Query OK, 0 rows affected (0.01 sec)
```

上述 SQL 语句中,在创建数据表 my_json 时,设置两个字段名 j1 和 j2,它们的数据类型都为 JSON。

(2) 添加数据进行测试,并查询数据是否添加成功,具体 SQL 语句及执行结果如下。

```
#添加数据
mysql>INSERT INTO my_json
    -> VALUES ('{"k1": "value", "k2": 10}', '["run", "sing"]');
Query OK, 1 row affected (0.00 sec)
#查询数据
mysql>SELECT * FROM my_json;
+-----------------------------+------------------+
| j1                          | j2               |
+-----------------------------+------------------+
| {"k1": "value", "k2": 10}   | ["run", "sing"]  |
+-----------------------------+------------------+
1 row in set (0.00 sec)
```

从上述示例结果可以看出,JSON 数据类型的字段以字符串的方式添加数据成功。

3.2　表的约束

为了防止在数据表中插入错误的数据,MySQL 定义了一些维护数据库中数据完整性和有效性的规则,这些规则即表的约束。表的约束作用于表中的字段上,可以在创建数据表或修改数据表的时候为字段添加约束。表的约束起着规范作用,确保程序对数据表的正确使用。技术中如此,生活中也是如此。我们在生活中需要遵循一些规则和道德准则,通过这些规则和道德准则可以帮助我们建立良好的人际关系,保持社会秩序的和谐。

数据表常见的约束有默认值约束、非空约束、唯一约束、主键约束和外键约束,其中外键约束涉及多表操作,将在第 6 章进行讲解。本节主要讲解如何设置默认值约束、非空约束、唯一约束和主键约束。

3.2.1　设置默认值约束

在实际开发中,有时需要为字段设置默认值。例如,在新增数据时,为了方便操作,希望一部分字段可以省略,直接使用默认值,这时就可以为这部分字段设置默认值约束。

默认值约束用于给数据表中的字段指定默认值,当在数据表中插入一条新记录时,如果没有给这个字段赋值,那么数据库系统会自动为这个字段插入指定的默认值。

在 MySQL 中,可以通过 DEFAULT 关键字设置字段的默认值约束。设置默认值约束的方式有两种,分别为创建数据表时设置默认值约束和修改数据表时添加默认值约束,当数据表中的某字段不需要设置默认值时,可以通过修改数据表的语句删除,具体如下。

1. 创建数据表时设置默认值约束

创建数据表时给字段设置默认值约束,基本语法格式如下。

```
CREATE TABLE 表名(
  字段名 数据类型 DEFAULT 默认值,
  …
);
```

上述语法格式中,“DEFAULT 默认值”表示设置默认值约束。

2. 修改数据表时添加默认值约束

在创建数据表时如果没有给字段设置默认值约束,可以在修改数据表时通过 ALTER TABLE 语句的 MODIFY 子句或 CHANGE 子句为字段添加默认值约束,基本语法格式如下。

```
#语法 1,MODIFY 子句
ALTER TABLE 表名 MODIFY 字段名 数据类型 DEFAULT 默认值;
#语法 2,CHANGE 子句
ALTER TABLE 表名 CHANGE [COLUMN] 字段名 字段名 数据类型 DEFAULT 默认值;
```

上述两种语法格式添加默认值约束的效果相同。需要注意的是,BLOB、TEXT 和 JSON 字段的数据类型不支持默认值约束。

3. 删除默认值约束

当数据表中的某字段不需要设置默认值时,可以通过 ALTER TABLE 语句的 MODIFY 子句或 CHANGE 子句以重新定义字段的方式删除字段的默认值约束,基本语法如下。

```
#语法 1,MODIFY 子句
ALTER TABLE 表名 MODIFY 字段名 数据类型;
#语法 2,CHANGE 子句
ALTER TABLE 表名 CHANGE [COLUMN] 字段名 字段名 数据类型;
```

上述语法中,通过 MODIFY 子句或 CHANGE 子句重新定义字段,即可删除默认值约束。

以上内容讲解了默认值约束的创建、修改与删除,下面通过案例演示默认值约束的使用,具体示例如下。

(1) 创建数据表 my_default,准备 name 和 age 两个字段进行测试,为 age 设置默认值约束,设置默认值为 18,具体 SQL 语句及执行结果如下。