# 第 2 章 数据类型与运算符

CHAPTER 2

关系数据库使用二维关系表管理数据，类似于 C 语言的结构体 struct，关系模式的基本数据类型不能脱离关系表独立存在。因此，先要定义关系表，再定义不同数据类型的表字段，才能测试 MySQL 数据类型的存储和管理各种数值的方法。

本章介绍 MySQL 数据类型和运算符，MySQL 各类数值表示与存储的方法，运算符号控制数值操作的规则。主要知识点如下。

2.1 MySQL 数据类型

了解 MySQL 基本数据类型，掌握整数、浮点数和定点数的表示方式和取值范围，日期时间类型的数据格式与相关函数，字符串和二进制类型的分类和功能用途。

第 3 集
微课视频

2.2 MySQL 运算符

掌握算术运算、逻辑运算、位运算、比较运算等基本运算符，及其形式结构与计算规则，掌握运算符运算对象的数据类型和返回结果。

掌握数据库特有的谓词运算符，了解基本功能与使用方法。

2.3 正则表达式

了解 MySQL 正则表达式的基本语法，使用这种强大的文本匹配工具，掌握实现复杂字符串检索和处理的有效方法。

## 2.1 MySQL 数据类型

MySQL 支持多种数据类型，主要有数值类型、日期/时间类型、字符串类型、空间数据类型和 JSON 类型。

（1）数值类型包括整数类型（TINYINT、SMALLINT、MEDIUMINT、INT、BIGINT）、浮点小数数据类型（FLOAT 和 DOUBLE）、定点小数类型（DECIMAL）。

（2）日期/时间类型包括 YEAR、TIME、DATE、DATETIME 和 TIMESTAMP。

（3）字符串类型包括 CHAR、VARCHAR、BINARY、VARBINARY、BLOB、TEXT、ENUM 和 SET 等。字符串类型又分为文本字符串和二进制字符串。

（4）空间数据类型包括几何（GEOMETRYCOLLECTION）、点（POINT）、线串（LINESTRING）、多边形（POLYGON）等，以及上述多值集合。

（5）JSON 类型。JavaScript 对象表示法（Javascript object notation，JSON）是一种开放标准的文件格式和轻量级数据交换格式。JSON 数据类型包括字符串、数字、布尔值、

数组和对象，用于表示复杂数据，在互联网中广泛应用于服务器之间 Web 应用程序的数据交换。

MySQL 提供了多种数值数据类型，不同数据类型管理不同取值范围的数值，在底层则需要相应的存储空间，如表 2-1 所示。

表 2-1 基本数据类型与存储空间

| 数 据 类 型 | 存储空间/B |
|---|---|
| TINYINT | 1 |
| SMALLINT | 2 |
| MEDIUMINT | 3 |
| INT，INTEGER | 4 |
| BIGINT | 8 |
| FLOAT（$f$） | 4（$0 \leqslant f \leqslant 24$）<br>8（$25 \leqslant f \leqslant 53$） |
| FLOAT | 4 |
| DOUBLE [PRECISION]，REAL | 8 |
| DECIMAL（$M$，$D$），NUMERIC（$M$，$D$） | 可变 |
| BIT($M$) | 约（$M$+7）/8 |

数据管理应该根据实际项目需求，选择合适的数据类型，才有利于节省存储空间，提高效率。

### 2.1.1 整数类型

MySQL 提供的主要整数类型有 TINYINT、SMALLINT、MEDIUMINT、INT（INTEGER）、BIGINT。表 2-2 列出了 5 种整数类型和存储空间，以及不同整数类型的无符号最大理论值和有符号理论数值范围。在定义关系表时，整数类型经常用于添加 AUTOINCREMENT 自增约束条件的属性字段。

表 2-2 MySQL整数类型

| 类 型 名 称 | 存储空间/B | 无符号最大理论值 | 有符号理论数值范围 |
|---|---|---|---|
| TINYINT | 1 | $2^8-1$ | $[-2^7, 2^7-1]$ |
| SMALLINT | 2 | $2^{16}-1$ | $[-2^{15}, 2^{15}-1]$ |
| MEDIUMINT | 3 | $2^{24}-1$ | $[-2^{23}, 2^{23}-1]$ |
| INT(INTEGER) | 4 | $2^{32}-1$ | $[-2^{31}, 2^{31}-1]$ |
| BIGINT | 8 | $2^{64}-1$ | $[-2^{63}, 2^{63}-1]$ |

不同整数类型的存储空间决定管理数据的范围，直观数据管理和操作时，需要根据工程数据的数值范围，定义适合的数据类型。

MySQL 的编译运行平台比较丰富，包括支持多种 DBMS 的 Navicat 数据库管理工具。

本书采用支持 Windows 和 Linux 系统、免费开源的 Workbench 8.0 可视化工具，也可以在 MySQL 命令行环境编译执行语句，展示运行结果。

**实例 2-1**　创建表，定义无符号整数类型

在命令行窗口 MySQL>提示符下或 Workbench 的 tab 窗口中输入如下语句。

```
-- 注释：CREATE TABLE 是创建关系表的语句，将在后文介绍相关命令格式
  CREATE TABLE ex_int_unsign(i TINYINT UNSIGNED,
      j SMALLINT UNSIGNED,k MEDIUMINT UNSIGNED,l INT UNSIGNED);
-- 注释：  INSERT INTO 语句将在后文介绍，POWER(x,y)和 POW(x,y)计算 x 的 y 次方
  INSERT INTO ex_int_unsign
      VALUES(POW(2,8)-1,POW(2,16)-1,POW(2,24)-1,POWER(2,32)-1);
-- 注释：  SELECT 语句，将在后文介绍
  SELECT * FROM ex_int_unsign;
```

在命令行环境下，运行结果如下。

```
i              j                 k                  l
255            65535             16777215           4294967295
```

在 Workbench 8.0 可视化环境下，输出结果相同。只是以图形用户界面（graphical user interface，GUI）显示运行结果和程序代码。后续实例运行结果，会选择命令行与 GUI 中的最清晰截图，不再赘述。

MySQL 定义字段为无符号整数类型时，需要首先声明字段变量，数据类型置于字段之右，关键字 UNSIGNED 置于数据类型之右。这种声明规则与高级语言语法不同，需要特别注意并严格遵守。

**实例 2-2**　创建表，定义有符号整数类型

在命令行窗口 MySQL>提示符下或 Workbench 的 tab 窗口中输入如下语句。

```
CREATE TABLE ex_int(i TINYINT,j SMALLINT,k MEDIUMINT,l INT,m BIGINT);
INSERT INTO ex_int VALUES(127,POW(2,15)-1,
   POW(2,23)-1,POWER(2,31)-1,POWER(2,63)-(POWER(2,9)+1));
INSERT INTO ex_int VALUES(-128,-pow(2,15),
   -POW(2,23),-POWER(2,31),-POWER(2,63)-POWER(2,10));
SELECT * FROM ex_int;
```

运行结果如下。

```
i      j          k          l              m
-128   -32768     -8388608   -2147483648    -9223372036854775808
127    32767      8388607    2147483647     9223372036854775807
```

定义 m 列为有符号 BIGINT 数据类型。当为 m 赋值 $2^{64}-1$ 时，同样提示超出管理范围。在插入负数 $-2^{63}-2^{10}$ 或正数 $2^{63}-(2^9+1)$ 时，当前版本可以成功执行插入命令。

当定义字段为无符号 BIGINT 类型时，如果赋值为 $2^{64}-1$，会提示超出管理范围。MySQL 8.0 版本无符号 BIGINT 类型最大数值为 18 446 744 073 709 547 520。

**实例 2-3**　创建表，定义无符号 BIGINT 整数类型

```
CREATE TABLE ex_ubigint(n BIGINT UNSIGNED);
INSERT INTO ex_ubigint VALUES(POWER(2,64)-POWER(2,11)-POWER(2,10));
SELECT * FROM ex_ubigint
```

运行结果如下。

```
18446744073709547520
```

### 2.1.2 浮点数和定点数类型

MySQL 使用浮点数和定点数管理小数数值。浮点数有单精度（FLOAT）和双精度（DOUBLE）两种类型，定点数有 DECIMAL 类型。

MySQL 分别使用 FLOAT(*M*, *D*)和 DOUBLE(*M*, *D*)定义单精度和双精度浮点数。其中 *M* 为精度，表示由整数位和小数位组成的总位数；*D* 为标度，表示小数位数。

**1. 浮点数类型**

IEEE 754 标准规范了单精度浮点数和双精度浮点数两种规格。单精度浮点数用 4 字节（32 位）表示，格式为 1 位符号位+8 位指数+23 位尾数。双精度浮点数用 8 字节（64 位）表示，格式为 1 位符号位+11 位指数+52 位尾数。

IEEE 754 标准还对尾数的格式进行规范：d.dddddd，d 表示数字，小数点左侧只有 1 位且不能为零，意味着二进制表示的尾数小数点左侧只能为 1。显然，此位用于指数位以提高尾数的精度。

因此，用 24 位表示单精度浮点数的尾数，用 53 位表示双精度浮点数的尾数，转换成十进制分别为 $2^{24}-1=16\ 777\ 215$ 和 $2^{53}-1=9\ 007\ 199\ 254\ 740\ 991$。

**实例 2-4** 创建表，定义有符号整数类型

在命令行或 GUI 窗口中逐条输入如下语句。

```
CREATE TABLE ex_float1(i FLOAT DEFAULT NULL);
INSERT INTO ex_float1 VALUES(3.402823466385*POWER(10,38));
INSERT INTO ex_float1 VALUES(-3.402823466385*POWER(10,38));
INSERT INTO ex_float1 VALUES(2.8026*POWER(10,-45));
SELECT * FROM ex_float1;
```

运行结果如下。

```
i
3.40282e38
-3.40282e38
2.8026e-45
```

与整数类型存储原理相同，浮点数和定点数同样需要具体存储空间。表 2-3 列出了 MySQL 中的小数类型和存储空间。

表 2-3 MySQL小数类型和存储空间

| 类 型 名 称 | 说 明 | 存储空间/字节 |
|---|---|---|
| FLOAT | 单精度浮点数 | 4 |
| DOUBLE | 双精度浮点数 | 8 |
| DECIMAL(*M*, *D*)，DEC | 压缩的“严格”定点数 | *M*+2 |

浮点数的优点是在长度一定的情况下，能够表示更广阔的数据范围，缺点是会引起精度问题。两个浮点数值进行减法和比较运算时也容易出问题。

**2. 定点数类型**

MySQL 使用 NUMERIC 和 DECIMAL 类型声明定点数，在 SQL 92 标准中，两种类型实现相同，经常替代转换。FLOAT 和 DOUBLE 类型存储整数变量，会以整数形式显示数值，不会保留小数位，这对于金融货币或科学数值的管理就存在诸多不便。

DECIMAL(*M*, *D*)以字符串的形式保存数值，*M* 定义数值的整数位数，*D* 定义数值的小数位数，轻松解决保留货币数值小数位的问题。DECIMAL 类型存储空间并不固定，其取值范围也是由精度值 *M* 决定，占用 *M*+2 字节。*M* 默认值为 10，最大值是 65；*D* 默认值为 0，最大值是 30。

**实例 2-5**　创建表，定义浮点与定点类型

创建 coord 表，其中字段 x、y、z 数据类型依次为 FLOAT(6, 2)、DOUBLE(6, 2)和 DECIMAL(6, 2)，向表中 3 个字段插入数据 126.622、45.707 和 20.666，语句如下。

```
CREATE TABLE coord(x FLOAT(6,2),y DOUBLE(6,2),z DECIMAL(6,2));
INSERT INTO coord VALUES(126.622,45.707,20.666);
SELECT * FROM coord;
```

在插入数据时，MySQL 给出了一个警告信息。

```
使用 SHOWWARNINGS 语句查看警告信息：
1 row(s)affected,1 warning(s):1265Datatruncated for column'z'at row1
```

可以看到 FLOAT 和 DOUBLE 类型在进行四舍五入时没有给出警告，而 DECIMAL 类型的 z 字段添加数值时，警告数据截断。运行结果如下。

```
x       y       i
126.62  45.71   20.67
```

FLOAT 和 DOUBLE 类型在不指定精度时，默认会按照由计算机硬件和操作系统决定实际的精度，DECIMAL 类型默认为 DECIMAL(10, 0)。

如果插入实际数值的整数位数大于 *M* 值，则会报错。如果数值的小数位数大于 *D* 值，则四舍五入进行处理。所以计算机管理小数时，一定要规定合理的误差值。

## 2.1.3　日期和时间类型

MySQL 提供多种表示日期和时间的数据类型，主要有 YEAR、TIME、DATE、DATETIME 和 TIMESTAMP。当只记录年信息时，可以只使用 YEAR 类型，使用 DATE 类型就显得画蛇添足。每个类型根据使用习惯规定了格式和取值范围，当指定确实不合法的值时系统将“零”值插入数据库。本节将介绍 MySQL 的日期和时间类型的使用方法，如表 2-4 所示。

表 2-4　日期和时间数据类型

| 类 型 名 称 | 格　　式 | 范　　围 | 存储空间/字节 |
|---|---|---|---|
| YEAR | YYYY | 1901～2155 | 1 |
| TIME | HH:MM:SS | –838:59.59～838:59:59 | 3 |
| DATE | YYYY-MM-DD | 0000-01-01～9999-12-3 | 3 |

续表

| 类 型 名 称 | 格　　式 | 范　　围 | 存储空间/字节 |
|---|---|---|---|
| DATETIME | YYYY-MM-DDHH:MM:SS | 0000-01-0100:00:00～<br>9999-12-3123:59:59 | 8 |
| TIMESTAMF | YYYY-MM-DDHH:MM:SS | 1970-01-0100:00-01UTC～<br>2038-01-1903:14:07UTC | 4 |

### 1. YEAR类型

YEAR 类型表示年度信息，需要 1 字节存储空间，赋值时采用如下多种格式。

（1）以 4 位字符串或 4 位数字格式表示，范围为 1901～2155。输入格式为'YYYY'或 YYYY，如输入'2024'或 2024，添加到数据库的值都是 2024。

（2）以 2 位字符串格式表示，范围为'00'～'99'。'00'～'69'和'70'～'99'的值分别被转换为 2000～2069 和 1970～1999 的 YEAR 类型数值。'0'与'00'的作用相同，插入的值自动转换为 2000。

（3）以 2 位数字表示，范围为 1～99。1～69 和 70～99 的值分别被转换为 2001～2069 和 1970～1999 的 YEAR 类型数值。注意，在这里 0 值将被转换为 0000，而不是 2000。

使用 2 位整数或字符格式，在插入数据时稍有不同。使用数字格式 0 插入 YEAR 类型字段时，实际上入库值为 0，而不是期望的 2000。只有使用字符串格式'0'或'00'，入库值才为 2000。插入非法 YEAR 类型值时，系统报错。

**实例 2-6**　创建表，定义 YEAR 类型字段

创建表，定义 YEAR 类型字段，语句如下。

```
CREATE TABLE year(y YEAR);
INSERT INTO year VALUES('0'),('00'),('70'),(0);
SELECT * FROM year;
```

运行结果如下。

```
 y
 2000
 2000
 1970
 0000
```

### 2. TIME类型

TIME 类型表示时间信息，需要 3 字节存储空间。格式为'HH:MM:SS'，其中 HH 表示小时，MM 表示分钟，SS 表示秒。

由于 TIME 类型不仅可以表示一天内的 24 小时，还可以表示某个事件的过去时间，或者两个事件之间的时间间隔，小时字段可以大于 24 或为负，所以 TIME 类型的取值范围为–838:59:59～838:59:59。

TIME 类型赋值时可以采用如下多种格式。

（1）'DHH:MM:SS'格式的字符串，还可以使用多种不太严谨的语法：'HH:MM:SS'、'DHH:MM'、'HH:MM'、'DHH'或'SS'，其中 D 表示日期，取值范围为–34～34。在插入数据时，D 转换为小时，格式为'D*24+HH'。使用'DHH'格式，小时必须为两位数值，小时数小

于 10 时，一定要 0，否则系统报错。

（2）'HHMMSS'格式的没有间隔符的字符串，或者 HHMMSS 格式的数值。例如，'171717'解释为'17:17:17'，'177717'解释为不合法。

TIME 类型赋值时，如果数值既不够 6 位，又没有加冒号分隔，MySQL 从右到左分析赋值，即最右侧两位表示秒，之后从右到左，以此类推。例如，1919 表示为 19 分 19 秒（即 00:19:19），919 表示为 9 分 19 秒（即 00:09:19）。如果数值不够 6 位，但有冒号分隔，MySQL 从左到右分析赋值。例如，19:19 表示为 19 时 19 分（即 19:19:00）

**实例 2-7**　创建表，定义 TIME 类型

创建表，定义 TIME 类型，语句如下。

```
CREATE TABLE time(t TIME DEFAULT NULL);
INSERT INTO time VALUES('34 22:59:59'),('-34 22:59:59');
SELECT * FROM time;
```

运行结果如下。

```
 t
 838:59:59
 -838:59:59
```

向表中插入一系列数据，语句如下。

```
INSERT INTO time VALUES('16:16:16'),('2222'),('121'),('22:22'),('512:12'),
 ('218'),('-310:10');
SELECT * FROM time;
```

运行结果如下。

```
 t
 16:16:16
 00:22:22
 00:01:21
 22:22:00
 512:12:00
 00:02:18
 -310:10:00
```

### 3. DATE类型

DATE类型表示日期信息，需要3字节存储空间。日期格式'YYYY-MM-DD'，其中 YYYY 表示年，MM 表示月，DD 表示日。使用字符串类型或数字类型赋值 DATE 类型，可以采用如下多种格式。

（1）'YYYY-MM-DD'或'YYYYMMDD'格式字符串，取值范围为'0000-01-01'～'9999-12-31'。例如，输入'2023-12-31'或者'20231231'，插入数据库的日期都为 2012-12-31。

（2）'YY-MM-DD'或'YYMMDD'格式字符串，其中 YY 表示年。

使用 YYYY 表示 19**～20**的年份，当表示跨度久远、长期不变的世纪数值时，'19'或'20'常常被忽略，取而代之使用 YY 表示年份。

MySQL 两位 YY 赋值规则是'00'～'69'的年份值转换为'2000'～'2069'，'70'～'99'的年份值转换为'1970'～'1999'。例如，输入'23-12-31'时，插入数据库的日期为 2023-12-31；输入'990101'时，插入数据的日期为 1999-01-01。

（3）YY-MM-DD 或 YYMMDD 的数字格式，00～69 的 YY 值转换为 2000～2069；70～99 的 YY 值转换为 1970～1999。例如。输入 24-01-01 时，插入数据库的日期为 2024-01-01；输入 970707 时，插入数据库的日期为 1997-07-07。

（4）调用 CURRENT_DATE()或 NOW()函数获取当前系统日期。

**实例 2-8** 创建表，定义 DATE 类型

创建表，定义 DATE 类型，语句如下。

```
CREATE TABLE date(d DATE DEFAULT NULL);
INSERT INTO date VALUES('0000-01-01'),('9999-12-31');
SELECT * FROM date;
```

运行结果如下。

```
d
0000-01-01
9999-12-31
```

向表中插入一系列数据，语句如下。

```
INSERT INTO date VALUES(NOW()),(001212),(970707),(CURRENT_DATE());
SELECT * FROM date;
```

运行结果如下。

```
d
2024-07-13
2000-12-12
1997-07-07
2024-07-13
```

CURRENT_DATE()函数只返回当前日期值，不包括时间部分；NOW()函数返回日期和时间值，在保存到数据库时，只保留其日期部分。

4. DATETIME类型

DATETIME 类型同时表示日期和时间信息，需要 8 字节的存储空间。日期格式为'YYYY-MM-DDHH:MM:SS'，其中 YYYY 表示年，MM 表示月，DD 表示日，HH 表示时，MM 表示分，SS 表示秒。使用字符串或数字类型赋值，可以采用如下多种格式。

（1）'YYYY-MM-DDHH:MM:SS'或'YYYYMMDDHHMMSS'字符串格式，取值范围为'0000-01-0100:00:00'～'9999-12-3123:59:59'。

（2）'YY-MM-DDHH:MMSS'或'YYMMDDHHMMSS'字符串格式。

（3）YYYYMMDDHHMMSS 或 YYMMDDHHMMSS 数字格式。

**实例 2-9** 创建表，定义 DATETIME 类型

创建表，定义 DATETIME 类型，语句如下。

```
CREATE TABLE mydatetime(d DATETIME DEFAULT NULL);
INSERT INTO mydatetime VALUES('9999-12-31 23:59:59');
INSERT INTO mydatetime VALUES(now()),(CURRENT_DATE());
SELECT * FROM mydatetime;
```

运行结果如下。

```
d
9999-12-31 23:59:59
```

```
2024-07-13 16:52:50
2024-07-13 00:00:00
```

5. TIMESTAMP类型

TIMESTAMP 类型需要 4 字节存储空间，格式为 YYYY-MM-DDHH:MM:SS，取值范围为'1970-01-0100:00:01'UTC～'2038-01-1903:14:07'UTC，显示宽度固定为 19 个字符。

TIMESTAMP 与 DATETIME 类型除了存储字节和支持的范围不同外，还有一个最大的区别：DATETIME 类型在存储日期数据时，按实际输入的格式存储，即输入什么就存储什么，与时区无关；而 TIMESTAMP 类型以协调世界时（coordinated universal time，UTC）格式存储，检索时参考设置的时区转换显示。

**实例 2-10**　创建表，定义 TIMESTAMP 类型

创建 stamp 表，插入并显示当前日期，更改时区为东 6 区，查看结果。

创建表，插入数据并查看的 SQL 语句如下。

```
CREATE TABLE stamp(s TIMESTAMP NULL DEFAULT NULL);
INSERT INTO stamp VALUES(NOW());
SELECT * FROM stamp;
```

运行结果如下。

```
s
2024-07-13 16:58:15
```

查询结果为插入数据时当前时区的日期，将东 8 区修改为东 6 区，SQL 语句如下。

```
settime_zone='+6:00';
SELECT * FROM stamp;
```

查看插入数据时的日期，由于东 6 区比东 8 区晚 2 小时，时区转换后，查询显示时间，推迟了 2 小时。

```
s
2024-07-13 14:58:15
```

因为 DATE 类型值未包含时间信息，如果为 DATETIME 或 TIMESTAMP 对象分配一个 DATE 类型值，时间部分被设置为 00:00:00。如果为一个 DATE 对象分配一个 DATETIME 或 TIMESTAMP 类型值，时间部分被删除。

## 2.1.4　字符串类型

MySQL 字符串类型包括 CHAR、VARCHAR、TEXT、ENUM 和 SET。字符串类型可以存储字符串数据，以及图片和声音等的二进制数据。字符串比较运算既可区分也可不区分大小写，还可以进行模式匹配查找。MySQL 字符串数据类型如表 2-5 所示。

表 2-5　MySQL字符串数据类型

| 类 型 名 称 | 说　　明 | 存 储 空 间 |
|---|---|---|
| CHAR(*M*) | 固定长度非二进制字符串 | *M*字节，$M<2^8$ |
| VARCHAR(*M*) | 可变长非二进制字符串 | *L*+1字节，*L*=*M*和$M<2^{16}$ |
| TINYTEXT | 非常小的非二进制字符串 | *L*+1字节，$L<2^8$ |
| TEXT | 小的非二进制字符串 | *L*+2字节，$L<2^{16}$ |

续表

| 类 型 名 称 | 说　　明 | 存 储 空 间 |
|---|---|---|
| MEDIUMTEXT | 中等的非二进制字符串 | $L$+3字节，$L<2^{24}$ |
| LONGTEXT | 大的非二进制字符串 | $L$+4字节，$L<2^{32}$ |
| ENUM | 枚举类型，只能有一个枚举字符串值 | 1或2字节，取决于枚举值的数量（最大值为65 535） |
| SET | 一个设置。字符串对象可以有零个或多个SET成员 | 1～4或8字节，取决于集合成员的数量（最多64个成员） |

VARCHAR 和 TEXT 类型是可变长类型，存储空间取决于列值的实际长度，如表 2-5 中 $L$ 参数，而非类型的最大可能尺寸。

例如，一个 VARCHAR(10)列能保存最大长度为 10 个字符的字符串，实际的存储空间是字符串的长度 $L$ 加上 1 字节。对于字符串'abcd'，实际 $L$ 为 4，则存储空间是 5 字节。

### 1. CHAR和VARCHAR类型

CHAR(*M*)定义固定长度字符串，*M* 表示字符串长度，*M* 的范围为 0～255。当实际长度不足时，在右侧填充空格以达到指定的长度。例如，CHAR(4)定义固定 4 个字符的字符串。

VARCHAR(*M*)定义可变长度字符串，*M* 表示字符串最大长度，*M* 的范围为 0～65535。VARCHAR 的实际长度由最长行的大小和使用的字符集确定，而其实际占用的空间（字节数）为字符串的实际长度加 1。例如，VARCHAR(50)定义了一个最大长度为 50 的字符串，如果插入的字符串只有 10 个字符，则实际存储的字符串为 10 个字符和一个字符串结束字符。表 2-6 所示为不同字符串插入 CHAR(4)和 VARCHAR(4)列的差异。

表 2-6　CHAR(4)与VARCHAR(4)存储区别

| 插入值 | CHAR(4) | | VARCHAR(4) | |
|---|---|---|---|---|
| | 存储结果 | 存储空间/字节 | 存储结果 | 存储空间/字节 |
| '' | '' | 4 | '' | 1 |
| 'ap' | 'ap' | 4 | 'ap' | 3 |
| 'app' | 'app' | 4 | 'app' | 4 |
| 'appl' | 'appl' | 4 | 'appl' | 5 |
| 'apple' | 'appl' | 4 | 'appl' | 5 |

定义 CHAR(*n*)之后，固定分配 *n* 字节存储空间，不足 *n* 字节字符串右侧补充空格，如果插入超出 *n* 字节的数据，系统会报错。

定义 VARCHAR(*n*)之后，插入数据的实际字节数如果不超过 *n*，系统分配存储空间为实际字节数加 1，但不会超过 *n*+1 字节。如果插入超出 *n* 字节的数据，系统报错。

### 2. TEXT类型

TEXT 类型包括 TINYTEXT、TEXT、MEDIUMTEXT 和 LONGTEXT 4 种，保存非二进制字符串的文本类型数据，不同类型的存储空间和数据长度各不相同，如表 2-7 所示。

当保存或查询 TEXT 值时，不删除尾部空格。

在 MySQL 中，TEXT 类型存储非二进制的字符串，适用于存储较长的文本数据。TEXT 类型有一个字符集，可以参考字符集中对应的值实现排序和比较运算。TEXT 类型的最大存储长度取决于表的字符集和存储引擎。例如，TEXT 类型最大存储空间为 65 535 字节，如果一个字符使用 1 字节表示，意味着可以存储大约 65 535 个字符。对于 UTF-8 字符集，每个字符最多占用 3 字节，存储字符的数量必然相应减少。

表 2-7　TEXT类型与存储范围

| 数 据 类 型 | 最 大 长 度 | 存储空间/字节 |
|---|---|---|
| TINYTEXT | 255 | $2^8-1$ |
| TEXT | 65 535 | $2^{16}-1$ |
| MEDIUMTEXT | 16M | $2^{24}-1$ |
| LONGTEXT | 4G | $2^{32}-1$ |

### 3. ENUM类型

ENUM 管理字符串对象，语法格式如下。

```
字段名 ENUM('值 1','值 2',…,'值 n')
```

ENUM类型字段名是管理的属性名，括号里定义枚举列表若干具体值，最多管理65 535个成员。赋值时，只能在枚举列表中选择，而且一次只能取一个成员值。

ENUM 数值内部使用整数表示，每个枚举值还有一个索引值，枚举列表成员索引值序号从 1 记起，MySQL 存储的就是这个索引号。ENUM 值依照列索引顺序排列，NULL 值位于所有非空字符串枚举值之前。例如，定义 ENUM 类型字段枚举列表（'first', 'second', 'third'），此字段取值以及索引如表 2-8 所示。

表 2-8　ENUM类型的取值与索引

| 取　　值 | 索　　引 |
|---|---|
| NULL | NULL |
| '' | 0 |
| first | 1 |
| second | 2 |
| third | 3 |

默认值是 ENUM 的必备项。如果将 ENUM 列声明为 NULL，NULL 值则为该列的一个有效值，而且默认值为 NULL。如果 ENUM 列被声明为 NOT NULL，其默认值为允许的枚举列表的第一个元素。

**实例 2-11**　创建表，定义 ENUM 类型

创建表，定义 ENUM 类型，语句如下。

```
CREATE TABLE ex_enum(score INT,level ENUM('A','B','C','D'));
INSERT INTO ex_enum VALUES(80,'B'),(90,1),(55,4),(99,'A');
```

```
SELECT * FROM ex_enum;
```

运行结果如下。

```
score  level
80     B
90     A
55     D
99     A
```

#### 4. SET类型

SET 管理字符串对象，语法格式如下。

```
SET(值 1,值 2,…,值 n)
```

括号里定义 SET 的零或多个不能重复的具体值,最多管理 64 个成员。赋值时与 ENUM 类型不同，ENUM 类型只能从定义成员值中选择其一，SET 类型可从定义的成员值中联合组合选择。

SET 值在内部用整数表示，列表中每个值都有一个索引编号。当创建表时，SET 成员值的尾部空格将自动被删除。

如果插入 SET 字段中的值有重复，则 MySQL 自动删除重复的值；插入 SET 字段的值的顺序并不重要，MySQL 会在保存数据库时，按照定义的顺序显示；如果插入了不正确的值，系统报错。

**实例 2-12** 创建表，定义 SET 类型

创建表，定义 SET 类型，语句如下。

```
CREATE TABLE set1(SET('m','a','n','g','o'));
INSERT INTO set1 VALUES('m,a,n'),('g,o'),('m,o,n'),('a,g,o'),
('g,o,a'),('g,o,o');
SELECT * FROM set1;
```

运行结果如下。

```
m,a,n
g,o
m,n,o
a,g,o
a,g,o
g,o
```

分析结果，如果插入重复值，只保留一个，如输入'g，o，o'，输出结果为'g，o'。如果插入没有按照 SET 定义顺序排列的值，自动参考定义顺序插入，如输入'g，o，a'，输出结果为'a，g，o'。

### 2.1.5 二进制类型

MySQL 支持文本字符串和二进制字符串两类字符型数据。前面讲解了存储文本的字符串类型，本节讲解 MySQL 中存储二进制数据的数据类型。

MySQL 管理二进制数据类型,包括 BIT、BINARY、VARBINARY、TINYBLOB、BLOB、MEDIUMBLOB、LONGBLOB，如表 2-9 所示。

表 2-9　二进制数据类型

| 类　　型 | 说　　明 | 存 储 需 求 |
|---|---|---|
| BIT(*M*) | 位字段类型 | 大约（*M*+7）/8字节 |
| BINARY(*M*) | 固定长度二进制字符串 | *M*字节 |
| VARBINARY(*M*) | 可变长度二进制字符串 | *M*+1字节 |
| TINYBLOB(*M*) | 非常小的BLOB | *L*+1字节，*L*<$2^8$ |
| BLOB(*M*) | 小BLOB | *L*+2字节，*L*<$2^{16}$ |
| MEDIUMBLOB(*M*) | 中等的BLOB | *L*+3字节，*L*<$2^{24}$ |
| LONGBLOB(*M*) | 非常大的BLOB | *L*+4字节，*L*<$2^{32}$ |

### 1. BIT类型

BIT 类型 *M* 参数表示值的位数，可设置范围为 1～64。如果没有定义 *M*，默认值为 1。如果字段赋值长度小于 *M* 位，需要在值左边填充 0。BIT 类型管理比特位值的信息。例如，十进制数字 192 存储为 11000000 二进制形式，可以定义字段类型为 BIT(8)，字段中就不能插入大于二进制 11111111 的数据。

**实例 2-13**　创建表，定义 BIT 类型

创建 bit 表，定义类型为 BIT(8)的字段，向表中插入数据并查看，语句如下。

```
CREATE TABLE bit(b BIT(8));
#下面语句只插入 128，可自行依次插入 128，192，252，255 数值
INSERT INTO bit VALUE(128);
SELECT b,BIN(b+1),BIN(b+0),BIN(b-1),BIN(b-1.2)  FROM bit;
```

使用插入语句依次添加 128，192，252，255 数值之后，运行结果如下。

```
b          BIN(b+1)        BIN(b+0)        BIN(b-1)        BIN(b-1.2)
0x80       10000001        10000000        1111111         1111110
0xC0       11000001        11000000        10111111        10111110
0xFC       11111101        11111100        11111011        11111010
0xFF       100000000       11111111        11111110        11111101
```

BIN()函数功能是将数值转换为二进制，其中 BIN(b+0)表示从字段 b 取值，加上偏移值 0 之后显示二进制。偏移值可以是正整数和负整数，如果是小数，则向负无穷方向取整。例如，+1.8 取整为+1，–1.2 取整为–2。

在 Workbench 8.0 可视化环境下，b 列显示十进制数字 128、192、252、255。

### 2. BINARY和VARBINARY类型

BINARY 和 VARBINARY 类型类似于 CHAR 和 VARCHAR，只是管理二进制的字符串，语法格式如下。

```
字段名 BINARY(M)
字段名 VARBINARY(M)
```

BINARY(*M*)的存储空间是固定的，如果赋值不足最大长度 *M*，需要在右侧填充'\0'直至补齐定义长度。例如，声明字段类型 BINARY(4)，当插入字符串'me'时，存储空间实际内容为'me\0\0'。

VARBINARY(*M*)类型管理可变长度，存储空间根据具体赋值，设置相应长度存储数据。例如，声明字段类型 VARBINARY(10)，如果赋值长度为 7，则实际存储空间是 7+1=8 字节，即实际空间大小设置为字符串实际长度加 1。

**实例 2-14** 创建表，定义 BINARY 和 VARBINARY 类型

创建表，定义 BINARY 和 VARBINARY 类型，语句如下。

```
CREATE TABLE bin(b BINARY(4),vb VARBINARY(10));
INSERT INTO bin VALUES(7,17);
SELECT length(b),length(vb) FROM bin;
```

运行结果如下。

```
length(b)  length(vb)
4          2
```

执行如下查询语句。

```
SELECT b,vb FROM bin;
```

在命令行环境下，运行结果如下。在 Workbench 环境下，显示 BLOB。

```
b              vb
0x37000000     0x3137
```

### 3. BLOB类型

BLOB 类型包括 TINYBLOB、BLOB、MEDIUMBLOB、ELONGBLOB 4 种，管理二进制大对象，存储可变数量的数据，如表 2-10 所示。

表 2-10 BLOB类型与存储空间

| 数 据 类 型 | 存 储 空 间 |
|---|---|
| TINYBLOB | 最大长度为255，$2^8$–1字节 |
| BLOB | 最大长度为65535，$2^{16}$–1字节 |
| MEDIUMBLOB | 最大长度为16777215，$2^{24}$–1字节 |
| LONGBLOB | 最大长度为4294967295或4GB，$2^{32}$–1字节 |

BLOB 类型存储二进制字符串，BLOB 列没有字符集，排序和比较运算参考字段的字节数值。

## 2.1.6 数据类型选型

MySQL 提供丰富的数据类型，根据工程需求选用最适合的数据类型，可以优化存储，提高数据库性能。参照经验，占用存储空间最少的数据类型优势显著。

### 1. 整数和浮点数

如果忽视小数部分，则使用整数类型存储数据。如果需要保留小数部分，则使用浮点数类型。对于浮点数据列，存入的数值会对该列定义的小数位进行四舍五入。例如，如果列值的范围为 1～99999，若使用整数，则 MEDIUMINT UNSIGNED 是最好的类型；若需要存储小数，则使用 FLOAT 类型。

浮点数包括 FLOAT 和 DOUBLE 类型。DOUBLE 类型精度比 FLOAT 类型高，因此，

如要求存储精度较高，应选择 DOUBLE 类型。

### 2. 浮点数和定点数

浮点数（FLOAT、DOUBLE）相对于定点数（DECIMAL）的优势是在长度一定的情况下能表示更大的数据范围。但是，由于浮点数容易产生误差，因此对精确度要求比较高时，建议使用定点数。定点数在 MySQL 中是以字符串存储的，用于定义货币等对精确度要求较高的数据。在数据迁移中，FLOAT(*M*, *D*)是非标准 SQL 定义，数据库迁移可能会出现问题，最好不要这样使用。另外，两个浮点数进行减法和比较运算时也容易出问题，因此在进行计算时一定要小心。如果进行数值比较，最好使用定点数。

### 3. 日期和时间类型

MySQL 对于不同种类的日期和时间有很多的数据类型，如 YEAR 和 TIME。如果只需要记录年份，则使用 YEAR 类型即可；如果只记录时间，使用 TIME 类型即可。

如果需要同时记录日期和时间，则可以使用 TIMESTAMP 或 DATETIME 类型。由于 TIMESTAMP 的取值范围小于 DATETIME 的取值范围，因此存储范围较大的日期最好使用 DATETIME 类型。

TIMESTAMP 也有一个 DATETIME 不具备的属性。默认情况下，当插入一条记录但并没有指定 TIMESTAMP 这个列值时，MySQL 会把 TIMESTAMP 列设为当前的时间。因此，当需要插入记录同时插入当前时间时，使用 TIMESTAMP 是方便的。另外，TIMESTAMP 在空间利用上比 DATETIME 更有效。

### 4. CHAR和VARCHAR

1）CHAR 和 VARCHAR 的区别

CHAR 是固定长度字符串，VARCHAR 是可变长度字符串；CHAR 会自动删除插入数据的尾部空格，VARCHAR 不会删除尾部空格。

因为 CHAR 是固定长度，所以它的处理速度比 VARCHAR 要快，但是它的缺点就是浪费存储空间，所以对于存储空间不大，但在速度上有要求的可以使用 CHAR 类型，反之可以使用 VARCHAR 类型来实现。

2）存储引擎对于选择 CHAR 和 VARCHAR 的影响

对于 MyISAM 存储引擎，最好使用固定长度的数据列代替可变长度的数据列。这样可以使整个表静态化，从而使数据检索更快，用空间换时间。

对于 InnoDB 存储引擎，使用可变长度的数据列，因为 InnoDB 数据表的存储格式不分固定长度和可变长度，因此使用 CHAR 不一定比使用 VARCHAR 更好，但由于 VARCHAR 是按照实际的长度存储，比较节省存储空间。

### 5. ENUM和SET

ENUM 只能取单值，它的数据列表是一个枚举集合，合法取值列表最多允许有 65 535 个成员。因此，在需要从多个值中选取一个时，可以使用 ENUM 类型。例如，性别字段适合定义为 ENUM 类型，每次只能从“男”和“女”中取一个值。

SET 可取多值。它的合法取值列表最多允许有 64 个成员。空字符串也是一个合法的 SET 值。在需要取多个值时，适合使用 SET 类型，如要存储一个人的兴趣爱好，最好使用

SET 类型。

ENUM 和 SET 值是以字符串形式出现的，但在内部，MySQL 以数值的形式存储它们。

6. BLOB和TEXT

BLOB 是二进制字符串，TEXT 是非二进制（文本类型）字符串，两者均可存储大容量的信息。BLOB 类型主要用于存储图片、音频信息等，而 TEXT 类型只能存储纯文本文件。

**同步练习**

2-1 使用整数类型，定义表中存储学生身高和体重的字段，说明哪种整数类型最合适。

2-2 测试不同整数类型的取值范围。

2-3 使用浮点数和定点数类型，定义表中存储圆半径和π值（保留 10 位小数）的字段。

2-4 使用 CHAR 和 VARCHAR 类型定义表，存储五言律诗。

2-5 使用 TEXT 类型定义表，存储唐诗《春江花月夜》。

2-6 使用 ENUM 类型定义表，存储 24 个希腊字母。

2-7 使用 SET 类型定义表，存储 26 个英文字母。

第 4 集
微课视频

2-8 使用二进制类型定义表，存储 IP 地址。

2-9 使用日期类型定义表，存储显示“年月日”日期信息，说明哪种类型最合适。

2-10 使用时间类型定义表，同时显示 24 个时区的“时分秒”的时间信息。

2-11 定义学生表 stu（stu_number，sname，gender，major，birthday），使用 DATE 类型管理出生日期属性。

2-12 定义课程简表 course_s（cno，cname，course_type），使用枚举类型 ENUM 管理课程类型属性 course_type，包括必修（compulsory）、选修（elective）、基础（basic）、通识（general）、实践（practice）几种性质的课程。

2-13 定义学生选课表 sel_course（stuno，courseno，selcost，seltime），使用 DATETIME 类型管理选课时间属性 seltime。

2-14 定义照片表 picture（ID, stuid, pic, input_time），分别使用 BLOB 和字符串数据类型管理 pic 属性。

## 2.2 MySQL 运算符

MySQL 支持算术运算、逻辑运算、位运算以及比较运算。运算符连接表达式中各个操作数，实现相应运算。

### 2.2.1 运算符概述

常见的运算符有算术运算符、逻辑运算符、位运算符、比较运算符。

（1）算术运算符执行数值运算，包括加（+）、减（–）、乘（*）、除（/）、求余（%，

又称模运算）等符号。

（2）逻辑运算符执行逻辑运算，包括逻辑与（AND 或&&）、逻辑或（OR 或||）、逻辑非（NOT 或!）、逻辑异或（XOR）等符号。逻辑运算执行结果均为 1（TRUE）或 0（FALSE）。

（3）位运算符执行二进制比特位运算，包括位与（&）、位或（|）、位非（–）、位异或（～）、左移（<<）、右移（>>）等符号。

（4）比较运算符执行比较运算，包括大于（>）、小于（<）、等于（=）、大于或等于（>=）、小于或等于（<=）、不等于（!=）等符号，以及 IN、BETWEEN AND、ISNULL、GREATEST、LEAST、LIKE、REGEXP 等语句符号。

### 2.2.2　算术运算符

MySQL 算术运算符如表 2-11 所示。

表 2-11　MySQL算术运算符

| 运　算　符 | 功　　能 |
|---|---|
| + | 加法运算，返回和 |
| – | 减法运算，返回差 |
| * | 乘法运算，返回乘积 |
| / | 除法运算，返回商 |
| % | 求余运算，返回余数 |

算术运算基本返回数值类型，有时返回 NULL。编程时，需要关注运算返回值的类型变化，以及异常的问题。

**实例 2-15**　创建表，测试算术运算符

创建表，测试算术运算符，语句如下。

```
CREATE TABLE operator(i INT);
INSERT INTO operator VALUE(31);
SELECT i,i+1.1,i*0.5,i/2,i%2,i/7,i%7,(i+1)*POWER(2,5),i/0 FROM operator;
```

运行结果如下。

```
i   i+1.1   i*0.5   i/2      i%2    i/7      i%7 (i+1)*POWER(2,5)   i/0
31  32.1    15.5    15.5000  1      4.4286   3   1024               NULL
```

整数和小数的运算，运算存储和返回结果是小数形式。即便可以整除，求商运算结果保留 4 位小数。求余运算余数是整数。除法运算中除数不能为 0，如果除以 0，返回结果为 NULL，系统不报错。

### 2.2.3　逻辑运算符

MySQL 逻辑运算符如表 2-12 所示。逻辑运算有 TRUE、FALSE 或 NULL 3 种执行结果，MySQL 分别表示为 1（TRUE）、0（FALSE）和 NULL。

表 2-12　MySQL逻辑运算符

| 运　算　符 | 功　　能 |
| --- | --- |
| AND或&& | 逻辑与 |
| OR或\|\| | 逻辑或 |
| NOT或! | 逻辑非 |
| XOR | 逻辑异或 |

**1. 逻辑与**

所有参与逻辑与运算（AND 或&&）的操作数全部为非零值并且全部不为 NULL 时，运算结果为 1。任意一个操作数为 0 时，运算结果都为 0。其余情况返回值为 NULL。

**2. 逻辑或**

所有参与逻辑或运算（OR 或||）的操作数全部为非 NULL 值，存在任意一个操作数为非零值时，运算结果为 1，否则为 0。当有一个操作数为 NULL，并且其他操作数为非零值时，运算结果为 1，否则为 NULL。当两个操作数均为 NULL 时，运算结果为 NULL。

**实例 2-16**　测试逻辑与和逻辑或运算符

测试逻辑与和逻辑或运算符的语句如下。

```
SELECT 0 OR NULL,2 || 9 ,0 && NULL, 7 AND 7;
```

运行结果如下。

```
0 OR NULL  2 || 9     0 && NULL  7 AND 7
NULL       1          0          1
```

**3. 逻辑非**

逻辑非运算（NOT 或!），当操作数为 0 时，返回值为 1；当操作数为非零值时，返回值为 0；当操作数为 NULL 时，返回值为 NULL。

**4. 逻辑异或**

逻辑异或运算（XOR），当任意一个操作数为 NULL 时，返回值为 NULL；对于非 NULL 的操作数，如果所有操作数都是非零值，或者都是零值，返回结果为 0；如果一个操作数为零值，其他操作数为非零值，返回结果为 1。

**实例 2-17**　测试逻辑非和逻辑异或运算符

测试逻辑非和逻辑异或运算符，语句如下。

```
SELECT NOT 0,! NULL,NULL XOR 1,7 XOR 6;
```

运行结果如下。

```
NOT 0      ! NULL        NULL XOR 1     7 XOR 6
1          NULL          NULL           0
```

## 2.2.4　位运算符

位运算是按照比特位运算二进制的字节数值。MySQL 位运算符包括位与（&）、位或（|）、位取反（～）、位异或（^）、位左移（<<）、位右移（>>），如表 2-13 所示。

表 2-13　MySQL位运算符

| 运　算　符 | 作　　用 |
| --- | --- |
| & | 位与 |
| \| | 位或 |
| ～ | 位取反，反转所有比特 |
| ^ | 位异成 |
| << | 位左移 |
| >> | 位右移 |

**1. 位与运算符**

参与位与运算（&）的两个操作数，按照二进制逐位执行逻辑与运算。测试语句（加粗，下同）和运行结果如下。

```
SELECT 255&192;
255&192
192
```

**2. 位或运算符**

参与位或运算（|）的两个操作数，按照二进制逐位执行逻辑或运算。测试语句和运行结果如下。

```
SELECT 192|255;
192|255
255
```

**3. 位取反运算符**

位取反运算（～）按照操作数的二进制逐位取反，即 1 取反为 0，0 取反为 1。测试语句和运行结果如下。

```
SELECT 192 & ~128;
192 & ~128
64
```

由于位取反运算符～比位与运算符&优先级别高，因此先执行 128（10000000）取反操作，之后取反值（01111111）再和 192（11000000）执行与运算，结果为 64（01000000）。

**4. 位异或运算符**

参与位异或运算（^）的两个操作数，按照二进制逐位执行逻辑异或运算。对应位的二进制数不同时，对应位的结果才为 1。如果两个对应位都为 0 或都为 1，则对应位的结果为 0。测试语句和运行结果如下。

```
SELECT 192^255;
192^255
63
```

192（11000000）与 255（11111111）按位执行异或运算，对应位二进制数不同时，计算结果为 1，相同则为 0，因此运算结果为 63（00111111）。

**5. 位左移运算符**

位左移运算（<<）的语法格式为 expr<<*n*，其中 *n* 是表达式 expr 返回值以二进制形式执行左移的位数。

左移指定位数之后，左侧高比特位移出丢弃，右侧低比特位空出位置用 0 补齐。测试语句和运行结果如下。

```
SELECT 1024<<3;
1024<<3
8192
```

十进制 1024 转换为二进制为 0000010000000000（$2^{10}$），左移 3 位后为 0010000000000000（$2^{13}$），转换回十进制是 8192。左移 1 位，如果没有溢出，相当于执行乘 2 操作。

**6. 位右移运算符**

位右移运算（>>）语法格式为 expr>>*n*，其中 *n* 是表达式 expr 返回值以二进制形式执行右移的位数。

右移指定位数之后，右侧低比特位移出丢弃，左侧高比特位空出位置用 0 补齐。测试语句和运行结果如下。

```
SELECT 1024>>3;
1024>>3
128
```

十进制 1024 转换为二进制为 0000010000000000（$2^{10}$），右移 3 位后为 0000000010000000（$2^{7}$），转换回十进制是 128。右移 1 位，如果没有溢出，相当于执行除 2 操作。

## 2.2.5 比较运算符

MySQL 比较运算符如表 2-14 所示。比较运算有时返回逻辑值，有时返回 NULL。在 SELECT 的查询条件中经常使用比较运算符。

表 2-14 MySQL比较运算符

| 运 算 符 | 功 能 |
|---|---|
| = | 等于 |
| <=> | 安全等于 |
| <>（!=） | 不等于 |
| <= | 小于或等于 |
| >= | 大于或等于 |
| > | 大于 |

**1. 等于运算符**

等于运算符（=）判断等号左右两侧的数字、字符串和表达式是否相等，如果相等，返回值为 1，否则返回值为 0。测试语句和运行结果如下。

```
SELECT'7'=7,7=7,'0.00001'=0,'k'='k',(2+7)=(3+6),NULL=NULL,1=NULL;
'7'=7   7=7 '0.00001'=0   'k'='k'    (2+7)=(3+6)    NULL=NULL   1=NULL
```

```
1       1   0              1          1               NULL        NULL
```

等于运算符（=）遵守如下规则。

（1）参与运算的两个操作数，若有一个或两个为 NULL，返回 NULL 值。

（2）参与运算的两个操作数都是字符串，则按照字符串进行比较。

（3）参与运算的两个操作数都是整数，则按照整数进行比较。

（4）如果判断一个字符串和数字是否相等，则自动将字符串转换为数字。

等于运算符并不是赋值符号，MySQL 有插入语句负责赋值。等于运算符无法判断是否为空值 NULL（不区分大小写），有其他运算符可完成这一功能。

### 2. 安全等于运算符

安全等于运算符（<=>）不仅能实现等号运算符的功能，还可以判断 NULL 值。测试语句运行结果如下。

```
SELECT '7'<=>7,7<=>7,'0.00001'<=>0,'k'<=>'k';
'7'<=>7    7<=>7   '0.00001'<=>0  'k'<=>'k'
 1          1       0               1
SELECT (2+7)<=>(3+6),null<=>NULL,1<=>NULL;
(2+7)<=>(3+6)  null<=>NULL    1<=>NULL
 1              1              0
```

安全等于运算符（<=>）遵守如下规则。

（1）参与运算的两个操作数都是 NULL（不区分大小写）时，返回值为 1。

（2）当一个操作数为 NULL，另一个操作数为非 NULL 值时，返回具体逻辑值。

### 3. 不等于运算符

不等于运算符（<>或!=）判断运算符两侧的数字、字符串、表达式是否不相等。如果不相等，返回值为 1；否则返回值为 0。不等于运算符不能用于判断空值 NULL。测试语句和运行结果如下。

```
SELECT 1<>'1',0<>0.0000,0!=5.5-5.5,'a'!=97,'0'<>48,0<>NULL;
1<>'1'  0<>0.0000  0!=5.5-5.5 'a'!=97  '0'<>48    0<>NULL
0       0          0           1        1         NULL
```

### 4. 小于或等于运算符

小于或等于运算符（<=）判断左边操作数是否小于或等于右边操作数，如果小于或等于，返回值为 1，否则返回值为 0。小于或等于运算符不能判断空值 NULL。测试语句和运行结果如下。

```
SELECT -1<='1',0<=-0.0000,'a'<='b','0'<='a',NULL<=1,NULL<=NULL;
-1<='1' 0<=-0.0000 'a'<='b'    '0'<='A'    NULL<=1     NULL<=NULL
1       1          1           1           NULL        NULL
```

### 5. 小于运算符

小于运算符（<）判断左边操作数是否小于右边操作数，如果小于，返回值为 1，否则返回值为 0。小于运算符不能判断空值 NULL。小于运算符不能判断空值 NULL。测试语句和运行结果如下。

```
SELECT -1<'1',0<-0.0000,'a'<'Z','0'<'a','Z'<'A',NULL<=1;
```

```
-1<'1'  0<-0.0000  'a'<'Z'  '0'<'a'    'Z'<'A'  NULL<=1
1       0          1        1          0        NULL
```

如果操作数是字符或字符串，按照字母表次序比较。

#### 6. 大于或等于运算符

大于或等于运算符（>=）判断左边操作数是否大于或等于右边操作数，如果大于或等于，返回值为 1；否则返回值为 0。大于或等于运算符不能判断空值 NULL。测试语句和运行结果如下。

```
SELECT 'godness'>='god',1.0/3>=0.3333,'#'>='@','NULL'>=NULL;
'godness'>='god'       1.0/3>=0.3333      '#'>='@'        'NULL'>=NULL
1                      1                  1               NULL
```

#### 7. 大于运算符

大于运算符（>）判断左边操作数是否大于右边操作数，如果大于，返回值为 1；否则返回值为 0。大于运算符不能判断空值 NULL。测试语句和运行结果如下。

```
SELECT 'godness'>'god',1.0/3>0.3333,'#'>'@','silly'>'silver','NULL'>
NULL;
'godness'>'god' 1.0/3>0.3333  '#'>'@'  'silly'>'silver'   'NULL'>NULL
1               1             1        0                  NULL
```

### 2.2.6 谓词运算符

MySQL 谓词运算符如表 2-15 所示，以谓词语言形式，结合数据库 DCL 语句实现查询任务。相对于符号运算符，更易读易理解。

表 2-15 MySQL谓词运算符

| 运 算 符 | 功 能 |
|---|---|
| ISNULL | 判断一个值是否为NULL |
| IS NULL | 与ISNULL相同 |
| IS NOT NULL | 判断一个值是否不为NULL |
| LEAST | 当有两个或多个参数时，返回最小值 |
| GREATEST | 当有两个或多个参数时，返回最大值 |
| BETWEEN AND | 判断一个值是否落在两个值之间 |
| IN | 判断一个值是列表中的任意一个值 |
| NOTIN | 判断一个值不是列表中的任意一个值 |
| LIKE | 通配符匹配 |

#### 1. NULL值判断

谓词运算符 IS NULL 和 ISNULL 检验一个值是否为 NULL，如果为 NULL，返回值为 1；否则返回值为 0。

谓词运算符 IS NOT NULL 检验一个值是否为非 NULL，如果为非 NULL，返回值为 1；否则返回值为 0。

测试语句和运行结果如下。

```
SELECT NULL IS NULL,NULL ISNULL,isnull(NULL),ISNULL(NULL),ISNULL(32);
SELECT  isnull(NOT NULL),ISNULL(0),32 ISNULL,32 IS NOT NULL;
NULL IS NULL   ISNULL      isnull(NULL)    ISNULL(NULL)    ISNULL(32)
1              NULL        1               1               0
isnull(NOT NULL)     ISNULL(0)     ISNULL      32 IS NOT NULL
1                    0             32          1
```

（1）IS NULL 和 is null 等效，可以判断左侧数据是否为空值，IS 和 NULL 之间的空格不能缺少。如果 IS 和 NULL 之间没有空格，则无法判断左侧数据是否为空值，系统也不会报错。

（2）IS NULL 和 IS NOT NULL 的返回值正好相反。

（3）ISNULL()和 IS NULL 等效，可以判断括号内数据是否为空值。ISNULL()的 IS 和 NULL 之间不能加空格。如果 ISNULL()括号内没有提供数值，系统会报错。

### 2. BETWEEN AND运算符

谓词运算符 BETWEEN AND 语法格式如下。

```
expr BETWEEN min AND max
```

判断表达式 expr 运算结果是否在设定范围之内，expr 大于或等于 min 且小于或等于 max 则返回值为 1，否则返回值为 0。测试语句和运行结果如下。

```
SELECT 7 BETWEEN 17 AND 27,'g' BETWEEN 'a' AND 'z',9.5 BETWEEN 9 AND 10;
7 BETWEEN 17 AND 27    'g' BETWEEN 'a' AND 'z'    9.5 BETWEEN 9 AND 10
0                      1                          1
```

比较字符串类型时，按字母表顺序进行比较，上述示例中字母 g 在指定字母区间（a～z）内，因此返回值为 1。

### 3. LEAST运算符

谓词运算符 LEAST 语法格式如下。

```
LEAST(值 1,值 2,…,值 n)
```

括号内参数列表可以设置 *n* 个值。当参数是整数或浮点数时，返回其中的最小值。当参数为字符串时，按照首字母以及从左至右的次序比较，返回参照字母表排位在前的字符以及字符串。当参数列表中有 NULL 时，无法判断大小，返回值为 NULL。测试语句和运行结果如下。

```
SELECT LEAST(0.0,0,0.0000),LEAST('seq','eat','each'),LEAST(0.0000,NULL);
LEAST(0.0,0,0.0000)    LEAST('SEQ','EAT','EACH') LEAST(0.0000,NULL)
0.0000                 each                       NULL
```

### 4. GREATEST运算符

谓词运算符 GREATEST 语法格式如下。

```
GREATEST(值 1,值 2,…,值 n)
```

括号内参数列表可以设置 *n* 个值。

当参数为整数或浮点数时，返回其中的最大值。当参数为字符串时，按照首字母以及

从左至右的次序比较,返回参照字母表排位在后的字符以及字符串。当参数列表中有 NULL 时，无法判断大小，返回值为 NULL。测试语句和运行结果如下。

```
SELECT GREATEST(0.0,0,0.0000),GREATEST('seq','eat','set'),
GREATEST(0.0000,NULL);
GREATEST(0.0,0,0.0000) GREATEST('seq','eat','set') GREATEST(0.0000,NULL)
0.0000                 set                         NULL
```

### 5. IN、NOT IN运算符

谓词运算符 IN，判断左侧值是否在 IN 右侧列表集合之中，如果在则返回值为 1；否则返回值为 0。

谓词运算符 NOT IN，判断左侧值是否不在 IN 右侧列表集合之中，如果不在则返回值 1，否则返回值为 0。测试语句和运行结果如下。

```
SELECT NULL IN(7,8,'nine'),9 IN(9,19,NULL);
NULL IN(7,8,'nine')     9 IN(9,19,NULL)
NULL                    1
SELECT 1 NOT IN(0,'1',NULL),NULL IN(7,8,NULL);
1 NOT IN(0,'1',NULL)    NULL IN(7,8,NULL)
0                       NULL
```

在左侧表达式为 NULL 的情况下，或是表中找不到匹配项并且表中一个表达式为 NULL，谓词运算符 IN 的返回值均为 NULL。

### 6. LIKE

谓词运算符 LIKE 语法格式如下。

```
expr LIKE 匹配条件
```

如果 expr 满足匹配条件，返回值为 1（TRUE）；如果不匹配，返回值为 0（FALSE）。若 expr 和匹配条件二者之中有 NULL，运算结果也为 NULL。有以下两种通配符辅助 LIKE 运算。

（1）%通配符，代表任意长度甚至零字符。

（2）_通配符，代表任意一个字符。

测试语句如下，运行结果如图 2-1 所示。

```
SELECT NULL LIKE 'comm','comm' LIKE 'com','comm' LIKE 'c%',
'comm' LIKE '_o__','' LIKE NULL;
```

| NULL LIKE 'comm' | 'comm' LIKE 'com' | 'comm' LIKE 'c%' | 'comm' LIKE '_o__' | '' LIKE NULL |
|---|---|---|---|---|
| NULL | 0 | 1 | 1 | NULL |

图 2-1　LIKE 实例

## 2.2.7　运算符的优先级

MySQL 运算经常需要组合多种运算符实现特定功能。MySQL 运算符具有不同优先级别，在同等条件下，级别高的运算符优先执行。如果级别相同，按表达式顺序从左到右依次排队执行。表 2-16 所示为 MySQL 各类运算符及其优先级。

表 2-16　MySQL各类运算符及其优先级

| 优　先　级 | 运　算　符 |
| --- | --- |
| 最低 | 赋值运算使用的等号（=），:= |
| ↓ | 逻辑或（OR，\|\|） |
| | 逻辑异或（XOR） |
| | 逻辑与（AND，&&） |
| | 逻辑非（NOT） |
| | BETWEEN AND，CASE，WHEN，THEN，ELSE |
| | 比较运算使用的等号（=），<=>，>=，>，<=，<，<>，!=<br>IS，LIKE，REGEXP，RLIKE，IN |
| | 位或（\|） |
| | 位与（&） |
| | 左移（<<），右移（>>） |
| | 加号（+），减号（–） |
| | 乘（*），除（/），取模（%） |
| | 异或（^） |
| | 正号（+），负号（–），位反转（～） |
| 最高 | 逻辑取反（!） |

如果不能准确确定运算符优先级，为了实现运算目标，在不同运算单元中，使用英文圆括号“( )”改变优先级，控制运算符执行次序，提高表达式逻辑顺序的易读性。

**同步练习**

2-15 计算商品零售价×数量的结果。

2-16 计算 210 的二进制与 252 的二进制进行与运算的结果。

2-17 计算 0 按位取反的结果。

2-18 计算 null 和 nuLL 进行安全等于运算的结果。

2-19 查询 Moon’s-day 是否在集合{Sunday，Monday，Tuesday，Wednesday，Thursday，Friday，Saturday}之中。

## 2.3　正则表达式

MySQL 正则表达式是一种强大工具，可以灵活构建关键字与字符组合，实现数据库中模式匹配、数据筛选和验证。MySQL 支持 REGEXP 和 RLIKE 实现正则表达式的匹配，这两个关键字在功能上是等效的。

REGEXP 运算符实现正则表达式模式匹配，语法格式如下。

```
expr REGEXP 匹配条件
```

如果 expr 满足匹配条件，返回值为 1；如果不满足，返回值为 0。若 expr 或匹配条件为 NULL，则结果为 NULL。

REGEXP 运算符在进行匹配时，常用通配符如下。

（1）^，匹配以该字符后面的字符开头的字符串。

（2）$，匹配以该字符后面的字符结尾的字符串。

（3）.，匹配任何一个单字符。

（4）[...]，匹配在方括号内的任何字符。例如，[abc]匹配 a、b 或 c；为了表示字符的范围，使用一个“-”，[a-z]表示匹配任何字母，而[0-9]表示匹配任何数字。

（5）*，匹配零个或多个在它前面的字符。例如，x*表示匹配任何数量的 x 字符；而.*匹配任何数量的任何字符。

如果查询时需要设置烦琐的条件，MySQL 提供更丰富的正则表达式通配符以及匹配组合，如表 2-17 所示。

表 2-17 MySQL正则表达式通配符

| 选　项 | 说　明 | 例　子 | 匹配值示例 |
|---|---|---|---|
| ^ | 匹配文本的首字符 | '^a'匹配首字母是a的字符串 | after、area、apple |
| $ | 匹配文本的结束字符 | 'ch$'匹配以结尾字母是ch的字符串 | beach、each、teach |
| . | 匹配任何单个字符 | 'c.t'匹配c和t之间任何字符的组合 | cat、cut、cute |
| * | 匹配零个或多个之前的字符 | 'e*d'匹配字符d前面有任意个字符e | edit、seed、feed |
| + | 匹配前面的字符一次或多次 | 'go+'匹配首字母是g，字母o在其后至少出现一次的字符串 | goal、good、goo |
| <字符串> | 匹配包含指定字符的文本 | 'hi' | high、chief、white |
| [字符集] | 匹配字符集合中任何字符 | '[jk]'匹配j或k | jar、jack、clock |
| | | [0-9] | 任何数字 |
| [^] | 匹配不在括号中的任何字符 | '[^lmn]'匹配任何不包含m、l、n的字符串 | bear、deer、yak |
| 字符串{*n*} | 匹配前面的字符串至少*n*次 | o{2}匹配两个或更多的o | oo、ooo、oooooo |
| 字符串{*n*,*m*} | 匹配前面的字符串至少*n*次，至多*m*次 | p{1,3}匹配最少1个，最多3个p | pp、ppp |

设置查询学号最后一位是 1 的学生信息，使用 RLIKE 匹配正则表达式的条件，语句如下。

```
SELECT * FROM stu WHERE stu_number RLIKE '1$';
```

输出显示如下内容。

```
stu_number          sname       gender      major
20221601            朱秦          女          电子工程
```

| | | | |
|---|---|---|---|
| 20221621 | 窦章 | 男 | 电子科学 |
| 20221631 | 苗凤 | 女 | 通信工程 |
| 20221641 | 倪汤 | 男 | 电子科学 |

正则表达式主要用来查询和替换符合某个模式（规则）的文本内容。相对于 LIKE 字符匹配，正则表达式更加强大，通过通配符的灵活组合，满足复杂的查询需求。

**同步练习**

2-20 在学生表 stu 中，查找 sname 字段中包含“龙”或“凤”的记录。

2-21 在学生表 stu 中，查询 stu_number 字段中出现字符串'20'最少 1 次、最多 2 次的记录。

2-22 参考商品销售简表 sale_s( serial_number, name_goods, price, quantity, shop )，查询 shop 字段中包含字母 a～e 和数字 1、2 以外字符的商铺记录。