# MySQL数据类型

# 数据类型的要求和基本原理

- 表示为字符串的数字或日期不容易执行操作，相加、相减或相乘这个数据将是一个复杂而又笨重的任务，它要求设计者首先把字符转化成数值类型，然后计算。
- 数据分类解决了所有这些问题使系统能够根据数据类型来操作数据。
- 强大的数据分类把每个类型与特定的行为联系在一起，执行这些行为可以预防认为错误。最长见的错误是字符与数字相加。
- 数据类型还可以更有效的利用空间，导致了更小的存储请求，同时提高了性能。例如把1234567898765432l识别为一个（8字节）数字，而不是一个（17字节，1个字符一个字节）字符串。

# MySQL数据类型

- 数值类型
- 数值类型可以大致划分为两个类型，一个是整数，另一个是浮点数或小数。

# MySQL数据类型

| 类型 | 大小 | 范围（有符号） | 范围（无符号） | 用途 |
| --- | --- | --- | --- | --- |
| TINYINT | 1字节 | （-128,127） | (0,255) | 小整数值 |
| SMALLINT | 2字节 | （-32768，32767） | （0，65535） | 大整数值 |
| MEDIUMINT | 3字节 | （-8388608，8388607） | （0，16777215） | 大整数值 |
| INT或INTERGER | 4字节 | （-2147483648，2147483647） | （0，4294967295） | 大整数值 |
| BIGINT | 8字节 | （-9223372036854775808，9223372036854775807） | （0，18446744073709551615） | 极大整数值 |
| FLOAT | 4字节 | | | 单精度浮点数值 |
| DOUBLE | 8字节 | | | 双精度浮点数值 |
| DECIMAL<br>小数值 | 对DECIMAL(M,D)如果M>D 为M+2，否则D+2。Thiz版本中为+1 | | | |

# INT类型

- 在MySQL中支持的5个主要整数类型是TINYINT，SMALLINT，MEDIUMINT，INT和BIGINT。这些类型在很大程度上是相同的，只有他们存储的值的大小是不同的
- 我们来实地做一下

# INT类型

```
mysql> create table data (fti TINYINT,fsi SMALLINT,fmi MEDIUMINT,fi INT,fbi BIGI
NT);
Query OK, 0 rows affected (0.59 sec)

mysql> INSERT INTO data VALUES(123456789,123456789,123456789,123456789,123456789
87654321);
Query OK, 1 row affected (0.45 sec)

mysql> INSERT INTO data VALUES(-123456789,-123456789,-123456789,-123456789,-1234
5678987654321);
Query OK, 1 row affected (0.00 sec)

mysql> select * from data;
+------+--------+----------+------------+---------------------+
| fti  | fsi    | fmi      | fi         | fbi                 |
+------+--------+----------+------------+---------------------+
|  127 |  32767 |  8388607 |  123456789 |  12345678987654321 |
| -128 | -32768 | -8388608 | -123456789 | -12345678987654321 |
+------+--------+----------+------------+---------------------+
2 rows in set (0.44 sec)
```

# INT类型

- MySQL以一个可选择的显示宽度指示器的形式对SQL标准进行扩展，这样当从数据库检索一个值时，可以把这个值加长到指定的长度。例如，指定一个字段的类型为INT(6),就可以保证所包含数字少于6个的值从数据库中检查出来时能够自动地用空格填充。
- 请注意，使用一个宽度指示器不会影响字段的大小和它可以存储的值的范围。

# INT类型

```
mysql> create table data2 (age INT(7));
Query OK, 0 rows affected (0.46 sec)

mysql> INSERT INTO data2 VALUES (19);
Query OK, 1 row affected (0.00 sec)

mysql> select age from data2;
+-------+
| age   |
+-------+
|    19 |
+-------+
1 row in set (0.09 sec)
```

# INT类型

- 来看下面这个例子说明了什么？

```
mysql> create table data2 (id TINYINT);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO data2 VALUES (123456789);
Query OK, 1 row affected (0.01 sec)

mysql> select * from data2;
+------+
| id   |
+------+
|  127 |
+------+
1 row in set (0.00 sec)
```

# INT类型

- 公式如下：
- 2的n-1次方-1（有符号的范围）
- 2的n次方-1（无符号的范围）

# INT类型

- MySQL会将不合规定的值在插入表之前自动改为0。

```
mysql> CREATE TABLE data2 (age INT);
Query OK, 0 rows affected (0.42 sec)

mysql> INSERT INTO data2 VALUES ("hello everyone");
Query OK, 1 row affected (0.46 sec)

mysql> select age from data;
ERROR 1054: Unknown column 'age' in 'field list'
mysql> select age from data2;
+-------+
| age   |
+-------+
|     0 |
+-------+
1 row in set (0.00 sec)
```

# INT类型

- UNSIGNED修饰符规定字段只保存正的值，它可以增大这个字段支持的值的范围。
- ZEROFILL修饰符规定0（不是空格）可以用于填补输出值。使用这个值可以防止MySQL存储负值。

# INT类型

```
mysql> create table data3(fi INT,fiu INT UNSIGNED,fiz INT ZEROFILL,fiuz INT UNSI
GNED ZEROFILL);
Query OK, 0 rows affected (0.53 sec)

mysql> INSERT INTO data3 VALUES(10,10,10,10)
    -> ,(-10,-10,-10,-10),
    -> (2147483647,2147483647,2147483647,2147483647),
    -> (3004005006,3004005006,3004005006,3004005006);
Query OK, 4 rows affected (0.50 sec)
Records: 4  Duplicates: 0  Warnings: 4

mysql> select * from data3;
+------------+------------+------------+------------+
| fi         | fiu        | fiz        | fiuz       |
+------------+------------+------------+------------+
|         10 |         10 | 0000000010 | 0000000010 |
|        -10 |          0 | 0000000000 | 0000000000 |
| 2147483647 | 2147483647 | 2147483647 | 2147483647 |
| 2147483647 | 3004005006 | 3004005006 | 3004005006 |
+------------+------------+------------+------------+
4 rows in set (0.47 sec)
```

# FLOAT,DOUBLE,DECIMAL类型

- MySQL支持3个浮点类型是FLOAT，DOUBLE和DECIMAL类型。FLOAT数值类型用于表示单精度浮点类型，而DOUBLE数值类型用于表示双精度浮点数值。
- 与整数类型一样，这些类型也带有附加参数：一个显示宽度指示器和一个小数点指示器。
- 例如：语句FLOAT（5，2）规定显示的值的宽度为5位数字，小数点后面带有2位数字。

# FLOAT,DOUBLE,DECIMAL类型

- 对于小数点后面的数字个数超过了允许的数目的值，系统会自动将它四舍五入为最接近它的值，然后插入它。

# FLOAT,DOUBLE,DECIMAL类型

```
mysql> create table data6 (test FLOAT(3,1));
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO data6 VALUES(123.456);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT test FROM data6;
+--------+
| test   |
+--------+
| 123.5  |
+--------+
1 row in set (0.00 sec)
```

# FLOAT,DOUBLE,DECIMAL类型

- UNSIGNED和ZEROFILL修饰符也可以被FLOAT，DECIMAL和DOUBLE数据类型接受，它们的结果和INT类型中介绍的一样。

# 字符串类型

- 字符串类型
- CHAR和VARCHAR类型
- CHAR后面的括号中必须用一个大小修饰符来定义。这个大小修饰符范围为0到255，指定了要存储的值的长度——例如：CHAR(10)指定了一个长度为10个字符的值。比指定长度小的值将会用空格适当填补；比指定长度大的值将被自动截短。

# 字符串类型

```
mysql> CREATE TABLE data8 (alphabet CHAR(10));
Query OK, 0 rows affected (0.44 sec)

mysql> INSERT INTO data8 VALUES ('abcdefghijklmno');
Query OK, 1 row affected (0.46 sec)

mysql> INSERT INTO data8 VALUES ('abc');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM data8;
+------------+
| alphabet   |
+------------+
| abcdefghij |
| abc        |
+------------+
2 rows in set (0.00 sec)
```

# 字符串类型

- CHAR类型认可一个可选择的BINARY修饰符，当用于比较预算时，这个修饰符使CHAR类型以一个二进制方式（不是传统的区分大小写）起作用。

# 字符串类型

```
mysql> CREATE TABLE data9 (name CHAR(5));
Query OK, 0 rows affected (0.48 sec)

mysql> INSERT INTO data9 VALUES('HOHO');
Query OK, 1 row affected (0.04 sec)

mysql> SELECT * FROM data9 WHERE name = 'hoho';
+------+
| name |
+------+
| HOHO |
+------+
1 row in set (0.15 sec)

mysql> ALTER TABLE data MODIFY name CHAR(5) BINARY;
ERROR 1054: Unknown column 'name' in 'data'
mysql> ALTER TABLE data9 MODIFY name CHAR(5) BINARY;
Query OK, 1 row affected (0.03 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM data9 WHERE name = 'hoho';
Empty set (0.00 sec)

mysql> SELECT * FROM data9 WHERE name = 'HOHO';
+------+
| name |
+------+
| HOHO |
+------+
```

# 字符串类型

- 因为现在字段使用一个二进制类型，所以MySQL执行一个二进制比较，很显然由于不匹配，它以失败告终。

# 字符串类型

- CHAR类型的一个变体是VARCHAR类型，它用于变长字符串，它也必须带有一个范围在0~255之间的大小指示器。然而，CHAR类型和VARCHAR类型之间的差别在于MySQL处理这个指示器的方式：CHAR把这个大小视为值得准确大小（用空格填补比较短的值，所以达到了这个大小），而VARCHAR类型把它视为最大值并且只使用了存储字符串实际上需要的字节数（增加了一个额外的字节记录长度）。因而，较短的值当被插入一个语句为VARCHAR类型的字段时，将不会用空格填补（然而，较长的值仍然被截短）。

# 日期和时间类型

- 当处理日期和时间值时，MySQL带有5个不同的数据类型可供选择，它们可以被分为简单的日期或时间类型和混合日期/时间类型。MySQL带有内置智能来识别并把多样化的输入格式转变为一个标准格式，以便于使用和操作。

# 日期和时间类型

- DATE,TIME和YEAR类型
- MySQL用DATE和YEAR类型描述简单的日期值，而使用TIME类型描述时间值。
- 这些值可以描述为字符串或不带分隔符的整数序列。如果描述为字符串，DATE类型的值应该使用连字号作为分隔符分隔开，而TIME类型的值应该使用冒号作为分隔符。

# 日期和时间类型

```
mysql> CREATE TABLE data10 (birthday DATE);
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO data10 VALUES ('2004-04-01'),(20040401);
Query OK, 2 rows affected (0.04 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM data10;
+------------+
| birthday   |
+------------+
| 2004-04-01 |
| 2004-04-01 |
+------------+
2 rows in set (0.00 sec)
```

# 日期和时间类型

```
mysql> CREATE table data11 (showtime TIME);
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO data11 VALUES ('11:11:11'),('11:11'),(111111);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM data11;
+----------+
| showtime |
+----------+
| 11:11:11 |
| 11:11:00 |
| 11:11:11 |
+----------+
3 rows in set (0.00 sec)
```

# 日期和时间类型

- 让我们来看下面的例子说明了什么？

```
mysql> CREATE TABLE data12 (
    -> f_data DATE,
    -> f_time TIME
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO data12 VALUES ('1978-4-6',123412),(650503,'3:4:1');
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM data12;
+------------+----------+
| f_data     | f_time   |
+------------+----------+
| 1978-04-06 | 12:34:12 |
| 2065-05-03 | 03:04:01 |
+------------+----------+
2 rows in set (0.00 sec)
```

# 日期和时间类型

- MySQL还对日期的年份中的两个数字的值，或是语句为YEAR类型的输入字段的两个数字输入执行这种类型的最大限度的通译。因为所有的YEAR类型的值必须用4个数字存储，所有MySQL试图根据值的数字范围把两个数字的年份值转换为4个数字的值：把在00~69范围内的值转换为2000~2069范围内，而把70~99范围内的值转换到1970~1999之内。如下例：

# 日期和时间类型

```
mysql> CREATE TABLE data13 (test YEAR);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO data13 VALUES (2003),(04),(9),(53),(96);
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT test FROM data13;
+------+
| test |
+------+
| 2003 |
| 2004 |
| 2009 |
| 2053 |
| 1996 |
+------+
5 rows in set (0.00 sec)
```

# 日期和时间类型

- DATETIME和TIMESTAMP类型
- 除了日期和时间数据类型，MySQL还支持这个两种类型的一个混合类型：DATETIME和TIMESTAMP数据类型，它们可以把日期和时间作为一个单值得组成成分来存储。

# 日期和时间类型

```
mysql> CREATE TABLE data14 (f_datatime DATETIME,f_timestamp TIMESTAMP);
Query OK, 0 rows affected (0.18 sec)

mysql> INSERT INTO data14 VALUES ('1999-11-11 11:11:11','2002-11-11 11:11:11');
Query OK, 1 row affected (0.44 sec)

mysql> INSERT INTO data14 VALUES(19991111111111,20021111111111);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO data14 VALUES (NOW(),NULL);
Query OK, 1 row affected (0.03 sec)

mysql> SELECT * FROM data14;
+---------------------+----------------+
| f_datatime          | f_timestamp    |
+---------------------+----------------+
| 1999-11-11 11:11:11 | 20021111111111 |
| 1999-11-11 11:11:11 | 20021111111111 |
| 2005-03-17 14:38:18 | 20050317143818 |
+---------------------+----------------+
3 rows in set (0.04 sec)
```

# 日期和时间类型

- 这两种类型通常用于自动存储包含当前日期和时间的时间戳，并且对执行大量数据库事务和需要建立一个调试和审查用途的审计跟踪的应用程序可以派上用场。
- 如果一个行中第一个字段语句为TIMESTAMP类型，而且这个字段没有被明确地指定值或者被指定了一个NULL值，MySQL将会自动用当前的时间和日期来填充它。

# 日期和时间类型

```
mysql> CREATE TABLE data15 (f_timestamp TIMESTAMP);
Query OK, 0 rows affected (0.19 sec)

mysql> INSERT INTO data15 VALUES ();
Query OK, 1 row affected (0.45 sec)

mysql> INSERT INTO data15 VALUES (NULL);
Query OK, 1 row affected (0.02 sec)

mysql> SELECT f_timestamp FROM data15;
+----------------+
| f_timestamp    |
+----------------+
| 20050317145641 |
| 20050317145655 |
+----------------+
2 rows in set (0.09 sec)
```

# 复合类型

- MySQL有两个复合类型： ENUM类型与SET类型。
- 它们的值必须从一个预先定义好的字符串集合中选取。一个ENUM类型从一个允许值集合中只选择单个值；SET类型允许从可选集合中选择多个值。

# 复合类型

- ENUM通常用于互斥的数据值。
- 如男人与女人。

# 复合类型

```
mysql> CREATE TABLE data16 (sex ENUM('M','F'));
Query OK, 0 rows affected (0.32 sec)

mysql> INSERT INTO data16 VALUES ('M'),('m'),('F'),('yy'),(NULL);
Query OK, 5 rows affected (0.03 sec)
Records: 5  Duplicates: 0  Warnings: 1

mysql> SELECT * FROM data16;
+------+
| sex  |
+------+
| M    |
| M    |
| F    |
|      |
| NULL |
+------+
5 rows in set (0.00 sec)
```

# 复合类型

- 可以看出，ENUM类型可以包含列于集合允许成员中的惟一的值或NULL值，任何不使用这些值而使用其它值得操作都将使MySQL插入一个空的字符串。如果插入值得大小写与在字段语句中相对应的大小写不匹配，那么插入的大小写将被自动改变为与字段语句相匹配，在这个实例中可以清晰看到这一点。

# 复合类型

- 虽然ENUM的元素只能指定为字符串，但是它们在系统内部可以存储为数字并且从1开始用数字作索引。一个ENUM类型最多可以包含65536个元素，其中一个元素为MySQL保留，用来保存错误信息，整个错误值用索引0或一个空字符串表示。

# 复合类型

```
mysql> SELECT COUNT(*) FROM data16 WHERE sex = 'M';
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
1 row in set (0.08 sec)

mysql> SELECT COUNT(*) FROM data16 WHERE sex = 1;
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM data16 WHERE sex = 2;
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.00 sec)
```

# 复合类型

```
mysql> SELECT COUNT(*) FROM data16 WHERE sex = 0;
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM data16 WHERE sex = '';
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.00 sec)
```

# 复合类型

- SET类型
- SET类型与ENUM类型相似又不完全相同，SET类型允许从预先定义的字符串集合中选取任意数目的值。语句为SET类型的字段可以包含0个、1个或1个以上从允许值集合中选取得值。

# 复合类型

```
mysql> CREATE TABLE data17 (type SET ('a','b','c','d','f'));
Query OK, 0 rows affected (0.19 sec)

mysql> INSERT INTO data17 VALUES ('a');
Query OK, 1 row affected (0.11 sec)
```

```
mysql> INSERT INTO data17 VALUES ('b,c');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT type FROM data17
    -> ;
+------+
| type |
+------+
| a    |
| b,c  |
+------+
2 rows in set (0.06 sec)
```

# 复合类型

- 与ENUM类型一样，任何企图使用一个不在预先定义集合的值的操作都会使MySQL插入一个空字符。

# 复合类型

```
mysql> INSERT INTO data17 VALUES ('j');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT type FROM data17 ;
+-------+
| type  |
+-------+
| a     |
| b,c   |
|       |
+-------+
```

# 复合类型

- 然而，如果试图插入一个既包含合法元素又包含非法元素的记录，MySQL将会除去非法的部分，保留合法的部分。

# 复合类型

```
mysql> INSERT INTO data17 VALUES ('a,b,c,j');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT type FROM data17 ;
+-------+
| type  |
+-------+
| a     |
| b,c   |
|       |
| a,b,c |
+-------+
4 rows in set (0.00 sec)
```

# 复合类型

- 一个SET类型最多可以包含64项。在SET类型中，值被存储为一个分离的位序列，这些位表示与它相对应的选取项。在SET类型中相同的元素重复2次是不可能的。

# 复合类型

```
mysql> INSERT INTO data17 VALUES ('a,a,a,b');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT type FROM data17 ;
+--------+
| type   |
+--------+
| a      |
| b,c    |
|        |
| a,b,c  |
| a,b    |
+--------+
5 rows in set (0.00 sec)
```

# 复合类型

- 这样也使查找包含错误数据的记录的位置变得很简单，需要做的就是查找包含空字符串或二进制为0的行。下面的两个命令是等价的。

# 复合类型

```
mysql> SELECT COUNT(*) FROM data17 WHERE type = '';
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM data17 WHERE type = 0;
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.00 sec)
```