MySQL数据类型

数据类型名称的规范遵循以下约定。

* 语法描述里的可选信息都用方括号（[]）来表示。
* M表示的是整型的最大显示宽度、浮点类型和小数类型的精度（有效位数）、BIT类型的位数，以及字符串类型的最大长度。在字符串列的定义里，二进制串的长度单位是字节，非二进制串的长度单位是字符。
* 对于那些带有小数部分的数字类型，D代表的是小数精度（即小数点后面的位数）。D必须小于等于M；否则，出错。
* fsp表示的是那些允许带有小数秒的时态类型的小数秒精度。

在对各种数据类型的描述里都包含有一种或多种下面的信息。

**含义**：对该类型的简短描述。

**可用属性**：指的是一些可能会与CREATE TABLE或ALTER TABLE语句里的数据类型有关联的可选属性关键字。这些属性将按字母表的顺序依次列出，但这并不表示在CREATE TABLE或ALTER TABLE语句（参考附录E）里也必须按同样的顺序列出它们。在各个数据类型的描述里并没有把包含那些可应用于所有（或近乎所有）数据类型的全局属性列出来。这些全局属性都集中列在下面，而没有将它们单独列在各个类型的描述里。

* **NULL或NOT NULL**可用于任何类型。
* **DEFAULT default\_value**可用在除下面几种类型列以外的所有类型列的定义里：拥有AUTO\_INCREMENT属性的整数列、BLOB和TEXT列，以及空间类型列。除了TIMESTAMP和DATETIME（自MySQL 5.6.5起）两种类型外，其他类型的默认值都必须是常数。例如，不能为DATE列指定DEFAULT CURDATE()。

**允许长度**：适用于字符串数据类型的列，它指的是该类型列的值被允许达到的最大长度。

**取值范围**：适用于数字类型和时态（日期/时间）类型，它指的是该类型所能表示的最大值。对于整数类型，因为存在有符号和无符号两种情况，所以需要指定两个取值范围，并且每种情况的取值范围也不同。

**零值**：适用于时态类型，它指的是如果有非法值插入该类型列里时，实际存入的那个零值。（必须将SQL模式设置为允许这种值；否则，有非法值插入时，会出现错误。）

**默认值**：类型定义没有明确设定DEFAULT属性时的默认值。这只在没有启用严格SQL模式时才适用。如果在严格模式中没有给出DEFAULT子句，并且能接受NULL值，那么会使用默认的NULL来定义列；否则，用非默认值定义。更多相关信息请参考3.2.3节。

**存储空间要求**：存储有关类型的值所需要使用的字节或字符个数。对于某些类型，该值是固定的。对于其他类型，该数字会因存储在列里的值的长度的不同而有所差异。

**比较方式**：对于字符串类型，这个值表明了比较操作进行的方式（这对分组、排序和索引操作也有影响）。二进制串类型是使用每个字节的数字值逐个字节地比较。非二进制串类型会根据字符集排序规则逐字符进行比较。

**同义词**：类型名称的同义词。

**备注**：与该类型有关的杂项说明。

注意：特有属性一般放在通用属性的前面。

# 数字类型

MySQL提供了精确值和近似值数字数据类型。不同的数字类型有着不同的取值范围，因此要根据想要表示的数值范围来加以选择。此外，还有一种BIT类型，专门用来表示位域值。

**整数和定点数**类型都是**精确值数据类型**。**FLOAT和DOUBLE**类型都是**近似值数据类型**。对于精确值类型，数值在存储和给定时是完全一致的，只要数值和计算结果没有超出这些类型的取值范围，计算过程就将精确地进行而不会产生任何舍入误差。对于近似值类型，计算会产生舍入误差。

对于整数类型的列，**如果为其指定了AUTO\_INCREMENT属性，就必须对它进行索引**。把NULL值插入一个AUTO\_INCREMENT列将导致下一个序列值被插入该列里。通常情况下，新序列值比该列的当前最大值大1。第3章对AUTO\_INCREMENT列的精确行为作了详细的描述（AUTO\_INCREMENT属性也可以用于浮点类型列，只是这种做法不太常见。）

属性ZEROFILL和UNSIGNED可以用于**除BIT类型以外**的数字类型。如果为某列指定了ZEROFILL属性，那么该列会用前导零将值的长度补齐到列的显示宽度。如果为某列指定了UNSIGNED属性，那么该列将不允许出现负值。（SIGNED其实也是一种可用属性，但因为数字类型在默认情况下就带有正负号，所以使用该属性没有什么实际效果。）

适用于整数和浮点数据类型的SERIAL DEFAULT VALUE属性是NOT NULL AUTO\_INCREMENT UNIQUE的缩写形式。

在某些情况下，指定一种属性会导致另外一种属性也被启用。**为数字类型指定ZEROFILL属性会导致该列被自动设定为UNSIGNED**。**指定AUTO\_INCRE  
MENT属性会导致该列被自动设定为NOT NULL**。

请注意，语句DESCRIBE和SHOW COLUMNS会报告说某个AUTO\_  
INCREMENT列的默认值是NULL，尽管你无法把一个NULL值插入这个列里去。这说明，你可以通过在创建新行时将AUTO\_INCREMENT列的值设置为NULL的方式来生成该列的默认值（即下一个序列编号）。

## 整型

|  |
| --- |
| ==================================================  **INYINT[(M)]**  **含义**：非常小的整数。M是最大显示宽度，1～255。如果省略，M的默认值是4（UNSIGNED列的默认值为3）。  **可用属性**：AUTO\_INCREMENT、SERIAL DEFAULT VALUE、UNSIGNED、ZEROFILL。  **取值范围**：有符号数的取值范围是-128～127（-27～27-1），无符号数的取值范围是0～255（0～28-1）。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为0。  **存储空间要求**：1个字节。  **同义词**：INT1[(M)]。此外，BOOL和BOOLEAN等同于TINYINT(1)。  ==================================================  ==================================================  **SMALLINT[(M)]**  **含义**：小整数。M是最大显示宽度，1～255。如果省略，M的默认值是6（UNSIGNED列的默认值为5）。  **可用属性**：AUTO\_INCREMENT、SERIAL DEFAULT VALUE、UNSIGNED、ZEROFILL。  **取值范围**：有符号数的取值范围是-32768～32767（-215～215-1）；无符号数的取值范围是0～65535（0～216-1）。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为0。  **存储空间要求**：2个字节。  **同义词**：INT2[(M)]。  ==================================================  **MEDIUMINT[(M)]**  **含义**：中等大小的整数。M是最大显示宽度，1～255。如果省略，M的默认值是9（UNSIGNED列的默认值为8）。  **可用属性**：AUTO\_INCREMENT、SERIAL DEFAULT VALUE、UNSIGNED、ZEROFILL。  **取值范围**：有符号数的取值范围是-8388608～8388607（-223～223-1）；无符号数的取值范围是0～16777215（0～224-1）。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为0。  **存储空间要求**：3个字节。  **同义词**：INT3[(M)]和MIDDLENT[(M)]。  ==================================================  **INT[(M)]**  **含义**：标准大小的整数。M是最大显示宽度，1～255。如果省略，M的默认值是11（UNSIGNED列的默认值为10）。  **可用属性**：AUTO\_INCREMENT、SERIAL DEFAULT VALUE、UNSIGNED、ZEROFILL。  **取值范围**：有符号数的取值范围是-2147483648～2147483647（-231～231-1）；无符号数的取值范围是0～4294967295（0～232-1）。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带NOT NULL属性，则默认值为0。  **存储空间要求**：4个字节。  **同义词**：INTERGER[(M)]和INT4[(M)]  ==================================================  **BIGINT [(M)]**  **含义**：大整数。M是最大显示宽度，1～255。如果省略，M的默认值是20。  **可用属性**：AUTO\_INCREMENT、SERIAL DEFAULT VALUE、UNSIGNED、ZEROFILL。  **取值范围**：有符号数的取值范围是-9223372036854775808～9223372036854775807（-263～263-1）；无符号数的取值范围是0～18446744073709551615（0～264-1）。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带NOT NULL属性，则默认值为0。  **存储空间要求**：8个字节。  **同义词**：INT8[(M)]。  **备注**：当SERIAL作为一种数据类型的名字时，它是BIGINT UNSIGNED NOT NULL AUTU\_INCREMENT UNIQUE的缩写。 |

## 定点类型

|  |
| --- |
| ==================================================  **DECIMAL[(M,[D])]**  **含义**：定点数。M是有效位数，1～65。D是小数点后面的精确位数，0～30。如果D等于0，列值就没有小数点或小数部分。如果同时省略M和D，则默认值分别为10和0。  **可用属性**：UNSIGNED、ZEROFILL。  **取值范围**：给定的DECIMAL列的取值范围由M和D的值以及是否给定UNSIGNED属性来决定。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为0。  **存储空间要求**：取决于小数点左右两边的数字个数。无论哪一边，都是每9位数字需要4个字节，最后剩下的数字需要1～4个字节。每个值的存储空间都等于小数点左右两边的存储之和。  **同义词**：NUMERIC[(M,[D])]、DEC[(M,[D])]和FIXED[(M,[D])]。 |

## 浮点类型

|  |
| --- |
| ==================================================  **FLOAT(p)**  **含义**：浮点数。在标准SQL中，精确度p表示至少需要的位数。在MySQL中，p仅用于确定数据类型是单精度的还是双精度的。  p值为0～24的浮点数是单精度浮点数，相当于不带说明符M或D的FLOAT类型。  p值为25～53的浮点数是双精度浮点数，即相当于不带说明符M和D的DOUBLE类型。  0～53以外的p值是非法的。  **可用属性**：UNSIGNED、ZEROFILL。  **取值范围**：参见本节后面对FLOAT和DOUBLE 类型的描述。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为0。  **存储空间要求**：单精度值需要4个字节，双精度值需要8个字节。  ==================================================  **FLOAT[(M,D)]**  **含义**：一种小型浮点数，即单精度浮点数（精度低于DOUBLE类型）。M是有效位数，范围是1～255，D是小数点后面的精确位数，范围是0～30。如果D等于0，则列值没有小数点或小数部分。如果省略M和D，则显示宽度和小数精度不确定；所有值都会以硬件系统所支持的最大精度来存储。  **可用属性**：UNSIGNED、ZEROFILL。  **取值范围**：最小非零值为±1.175494351E-38，最大非零值为±3.402823466E +38。如果该列指定了UNSIGNED属性，则不允许出现负值。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为0。  **存储空间要求**：4个字节。  **同义词**：FLOAT4是不带说明符M或D的FLOAT的同义词。如果启用了SQL模式REAL\_AS\_FLOAT，则REAL[(M,D)]是FLOAT(M,D)的同义词。  ==================================================  **DOUBLE[(M,D)]**  **含义**：一种大型浮点数，即双精度浮点数（精度高于FLOAT）。M和D的含义与FLOAT中的一样。  **可用属性**： UNSIGNED、ZEROFILL。  **取值范围**：最小非零值是±2.2250738585072014E-308，最大非零值是±1.7976931348623157E+ 308。如果为该列指定了UNSIGNED，就不允许出现负值。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带NOT NULL属性，则默认值为0。  **存储空间要求**：8个字节。  **同义词**：DOUBLE PRECISION[(M,D)]是DOUBLE[(M,D)]的同义词；如果没有启用SQL模式REAL\_AS\_FLOAT，则其同义词为REAL[(M,D)]。FLOAT8是不带说明符M或D的DOUBLE的同义词。 |

## BIT类型

|  |
| --- |
| ==================================================  **BIT[(M)]**  **含义**：位域值。M应该是一个范围为1～64的整数，用于表明每个BIT值的二进制位数。如果省略，则M默认为1。  **可用属性**：可用属性：除全局属性外，没有其他特殊属性。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为0。  **存储空间要求**：大约为（M+7）/8个字节。 |

# 字符串类型

MySQL中的字符串类型是通用类型，通常用来存储二进制或者字符（文本）数据。这些字符串类型能够容纳的值的最大长度是不一样的，可以根据你想把它们视为二进制串还是非二进制串来选择。

BINARY、VARBINARY和BLOB类型是**二进制串类型**。二进制串是一个字节序列，其长度单位是字节。二进制串没有字符集的概念，因此对值的比较是以字节值为基础的。

CHAR、VARCHAR和TEXT类型是**非二进制串类型**。非二进制串是一个字符序列。它有字符集和排序规则的概念。字符集定义的是允许数据类型使用的各种字符；排序规则定义的是字符排列顺序。在一个非二进制串列的定义里指定的长度表明的是该列能够容纳的最大字符数。

非二进制串值的长度单位通常为字符，但也可以使用字节为单位。若想获得某个非二进制串的字符长度或字节长度，可以分别使用函数CHAR\_LENGHT()或LENGTH()。如果某个非二进制串只包含单字节字符，且字符长度为n，那么其字节长度也为n；但如果它包含的是多字节字符，那么其字节长度会大于n。这种差异会影响非二进制串列的存储空间。

固定长度的列，如CHAR(M)，所需的存储空间必须能够存储下M个给定字符集里的最宽字符。例如，utf8字符集里的每个字符需要占用1～3个字节，因此CHAR(M)需要占用M×3个字节。

可变长度的列，如VARCHAR(M)，所需的存储空间除了必须能存储下给定值的所有字符，还要能够放得下存储该字符串字节长度的前缀。对于一个字符集为双字节ucs2的VARCHAR(10)列，它需要0（空串）～20个字节（10字符字符串），用于存储字符串里的字符；外加1个字节，用于存储其长度前缀。

除ENUM和SET类型外，非二进制串类型（CHAR、VARCHAR、TEXT）也都可以指定字符集和排序规则：

* 指定字符集的语法是CHARACTER SET charset，其中，charset是字符集名称，如latin1、greek或utf8。CHARSET等同于CHARACTER SET。
* 指定排序规则的语法是COLLATE collation，其中，collation用于指定给定字符集的某个可用排序规则。
* 如果没有设定字符集或排序规则，那么MySQL将根据表的默认设置来确定。如果只设定了字符集而没有设定排序规则，MySQL将使用该字符集的默认排序规则。如果只设定了排序规则而没有设定字符集，MySQL将根据排序规则的名称来确定字符集。如果既设定了字符集又设定了排序规则，则排序规则必须和字符集相兼容。例如，latin1\_bin排序规则和latin1字符集兼容，但是不兼容于utf8字符集。
* binary字符集和BINARY列属性需要特别对待的：如果为非二进制串类型指定了CHARACTER SET binary，那么非二进制串类型会被转换为相对应的二进制串类型。也就是说，CHAR类型将变成BINARY类型，VARCHAR类型将变成VARBINARY类型，TEXT类型将变成BLOB类型。ENUM和SET类型没有相对应的二进制类型，因此CHARACTER SET binary在遇到ENUM和SET类型的时候将变成一个列属性。
* BINARY属性相当于为字符集（以\_bin结尾的排序规则名称）指定二进制排序规则。例如，定义为CHAR(10) CHARACTER SET utf8 BANARY的列将变成CHAR(10) CHARACTER SET utf8 COLLATE utf8\_bin。
* 对于非二进制串类型，ASCII和UNICODE属性分别是CHARACTER SET latin1和CHARACTER SET ucs2的缩写。

服务器支持的字符集和排序规则可以通过SHOW CHARACTER SET和SHOW COLLATION语句来查询。这些语句将显示出每种字符集的默认排序规则。另外，还可以去查看数据库INFORMATION\_SCHEMA里的CHARACTER\_SETS表和COLLATIONS表——它们包含的信息相同。

有些长到无法存储在字符串列里的值，对这些值的处理取决于SQL模式设置。如果没有启用严格SQL模式，则超长值会被截短，以便能刚好存放在该列里。如果被截去的字符不全是空格，那么MySQL还将生成一条警告消息。在严格SQL模式下，如果必须截去非空格字符，那么服务器会产生一个错误，并且不会插入任何值。

**不同的字符串类型值有不同的尾部填充方式**：

* 对于CHAR类型，如有必要，在存储时会使用空格补齐列的整个长度。在检索时，尾部的空格会被去掉。
* 对于BINARY类型，如有必要，在存储时会使用0x00字节补齐列的整个长度。在检索时，不会去掉任何内容。
* 对于VARBINARY、VARCHAR，以及BLOB和TEXT类型，在存储时不会补齐任何内容；在检索时，也不会删除任何内容。
* 对于ENUM和SET类型，在列定义里列出的成员值的任何尾部空格都将被忽略。因此，存储在列里的值没有任何尾部的空格，因为MySQL会把每个值转换为对应于该列成员的内部数值。这同样会影响到比较操作，因为在对ENUM或SET列里的值进行比较的时候，尾部的空格不参加比较。

## 二进制串类型

|  |
| --- |
| ==================================================  **BINARY[(M)]**  **含义**：固定长度（0～M个字节）的二进制串。M应该是一个0～255的整数。如果省略，则M的默认值是1。  **可用属性**：除全局属性外，没有其他特殊属性。  **允许长度**：0～M个字节。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为''（空串）。  **存储空间要求**：M个字节。  **比较方式**：根据数字字节值依次比较各个字节。  ==================================================  **VARBINARY[(M)]**  **含义**：可变长度（0～M个字节）的二进制串。M应该是一个0～65535的整数。  **可用属性**：除全局属性外，没有其他特殊属性。  **允许长度**：0～M个字节（如备注所述，长度有可能小于M）。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为''（空串）。  **存储空间要求**：值本身的长度（以字节计算）再加上1或2个字节的前缀。如果列值的最大长度小于256个字节，则前缀需要1个字节；否则，需要2个字节。  **比较方式**：依次比较各个字节的值。  **备注**：在实际工作中，VARBINARY列的最大长度为65535个字节，在某些情况下，它可能还更小，具体取决于存储引擎内部的行大小限制，以及表里其他列所占用的空间。  ==================================================  **TINYBLOB**  **含义**：小的BLOB（二进制串）值。  **可用属性**：除全局属性外，没有其他特殊属性。  **允许长度**：0～255（0～28-1）个字节。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为''（空串）。  **存储空间要求**：值的长度（单位为字节），外加1个字节（用于记录该长度值）。  **比较方式**：根据数字字节值依次比较各个字节。  ==================================================  **BLOB[(M)]**  **含义**：标准大小的BLOB（二进制串）值。  **可用属性**：除全局属性外，没有其他特殊属性。  **允许长度**：0～65535（0～216-1）个字节。如果给出了长度M，则可以用它来选择一种适当的数据类型，然后丢弃它。如果M的值为1～65535，则数据类型变成BLOB。如果M的值等于或大于65535，则数据类型将根据值的实际长度变成MEDIUMBLOB或LONGBLOB。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为''（空串）。  **存储空间要求**：值的长度（单位为字节），外加2个字节（用于记录该长度值）。  **比较方式**：根据数字字节值依次比较各个字节。  ==================================================  **MEDIUMBLOB**  **含义**：中等大小的BLOB（二进制串）值。  **可用属性**：除全局属性外，没有其他特殊属性。  **允许长度**：0～16777215（0～224-1）个字节。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为''（空串）。  **存储空间要求**：值的长度（单位为字节），外加3个字节（用于记录该长度值）。  **比较方式**：根据数字字节值依次比较各个字节。  **同义词**：LONG VARBINARY。  ==================================================  **LONGBLOB**  **含义**：大的BLOB（二进制串）值。  **可用属性**：除全局属性外，没有其他特殊属性。  **允许长度**：0～4294967295（0～232-1）个字节。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为''（空串）。  **存储空间要求**：值的长度（单位为字节），外加4个字节（用于记录该长度值）。  **比较方式**：根据数字字节值依次比较各个字节。 |

## 非二进制串类型

|  |
| --- |
| ==================================================  **CHAR[(M)]**  **含义**：固定长度（0～M个字符）的非二进制串。M应该是一个0～255的整数。如果省略，M的默认值是1。  **可用属性**：BINARY、CHARACTER SET、COLLATE。  允许长度：0～M个字符。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为''（空串）。  **存储空间要求**：M个字符，长度是M×w个字节，其中，w是列所使用的字符集里的最宽字符需要占用的字节个数。  **比较方式**：根据列所使用的排序规则依次比较各个字符。  **同义词**：NCHAR(M)和NATIONAL CHAR(M)都等同于CHAR(M) CHARACTER SET utf8。  ==================================================  **VARCHAR(M)**  **含义**：可变长度（0～M个字符）的非二进制串。M应该是一个0～65535的整数。  可用属性：BINARY、CHARACTER SET、COLLATE。  **允许长度**：0～M个字符（如备注所述，长度有可能小于M）。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为''（空串）。  **存储空间要求**：值的长度（单位为字节），外加1或2个字节的前缀（用于记录该长度值）。如果列值的最大长度小于256个字节，则前缀需要1个字节；否则，需要2个字节。  **比较方式**：根据列所使用的排序规则依次比较各个字符。  **同义词**：CHAR VARYING(M)、NVRACHAR(M)、NCHAR VARYING(M)和NATIONAL CHAR VARYING(M)都等同于VARCHAR(M) CHARACTER SET utf8。  **备注**：在实际工作中，VARCHAR列的最大长度为65535个字节，在某些情况下，它可能还更小，具体取决于存储引擎内部的行大小限制、列所使用的字符集是单字节字符集还是多字节字符集，以及表里的其他列所占用的空间。  ==================================================  **TINYTEXT**  **含义**：小的TEXT（非二进制串）值。  **可用属性**：BINARY、CHARACTER SET、COLLATE。  **允许长度**：0～255（0～28-1）个字节；如果该值包含的是多字节字符，则字符数会更少。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为''（空串）。  **存储空间要求**：值的长度（单位为字节），外加1个字节（用于记录该长度值）。  **比较方式**：根据列所使用的排序规则依次比较各个字符。  ==================================================  **TEXT[(M)]**  **含义**：标准大小的TEXT（非二进制串）值。  **可用属性**：BINARY、CHARACTER SET、COLLATE。  **允许长度**：0～65535（0～216-1）个字节；如果该值包含的是多字节字符，则字符数会更少。如果给出了长度M，则可以用它来选择一种适当的数据类型，然后丢弃它。如果M的值为1～65535，则数据类型变成TEXT。如果M的值大于或等于65535，则数据类型将根据值的实际长度变成MEDIUMTEXT或LONGTEXT。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为''（空串）。  **存储空间要求**：值的长度（单位为字节），外加2个字节（用于记录该长度值）。  **比较方式**：根据列所使用的排序规则依次比较各个字符。  ==================================================  **MEDIUMTEXT**  **含义**：中等大小的TEXT（非二进制串）值。  **可用属性**：BINARY、CHARACTER SET、COLLATE。  **允许长度**：0～16777215（0～224-1）个字节；如果该值包含的是多字节字符，则字符数会更少。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为''（空串）。  **存储空间要求**：值的长度（单位为字节），外加3个字节（用于记录该长度值）。  **比较方式**：根据列所使用的排序规则依次比较各个字符。  **同义词**：LONG VARCHAR。  ==================================================  **LONGTEXT**  **含义**：大的TEXT（非二进制串）值。  **可用属性**：BINARY、CHARACTER SET、COLLATE。  **允许长度**：0～4294967295（0～232-1）个字节；如果该值包含的是多字节字符，则字符数会更少。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为''（空串）。  **存储空间要求**：值的长度（单位为字节），外加4个字节（用于记录该长度值）。  **比较方式**：根据列所使用的排序规则依次比较各个字符。 |

## ENUM和SET类型

|  |
| --- |
| ==================================================  **ENUM('value1','value2',...)**  **含义**：枚举值；列的取值只能是值列表中的某个成员。  **可用属性**：CHARACTER SET、COLLATE。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则为枚举集合中的第1个成员。  **存储空间要求**：如果成员个数为1～255，则占用1个字节；如果为256～65535，则占用2个字节。  **比较方式**：根据列值的数值进行比较。  **备注**：在数据类型定义里，各成员值里的尾部空格将被忽略。  ==================================================  **SET('value1','value2',...)**  **含义**：集合；列的取值可以是值列表中的0个或者多个成员。  可用属性：CHARACTER SET、COLLATE。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为空集（''）。  **存储空间要求**：1个字节（1～8个成员的集合）、2个字节（9～16个成员）、3个字节（17～24个成员）、4个字节（25～32个成员）或8个字节（33～64个成员）。  **比较方式**：根据列值的数值进行比较。  **备注**：在数据类型定义里，各成员值的所有尾部空格会被忽略。 |

# 时态（日期和时间）类型

MySQL提供了几种表示时态数据的类型。用于表示日期和时间的类型，既可以单独使用，也可以组合使用；并且当不需要完整的日期时，还有一种用于存储年份的类型。对于新建行，有些类型可以被自动初始化成当前的日期和时间；当行里的其他列发生变化时，它们还可以被自动更新为当前的日期和时间。

日期格式里的术语CC、YY、MM、DD分别代表世纪、年、月、日。时间格式里的术语hh、mm、ss分别代表小时、分、秒。此外，MySQL 5.6.4为TIME、DATETIME和TIMESTAMP类型引入了小数秒，并且允许选择性地将小数部分的最大精度设置为6位数（微秒）。时间格式里的术语uuuuuuu表示的是小数部分。（对于MySQL 5.6.4以前的版本，小数秒部分会被忽略。）

在类型语法描述里，fsp表示的是允许使用小数秒部分的类型的小数秒精度。该精度必须是一个0～6的值，其中，0表示的是无小数秒部分，6表示的是精确到微秒。如果没有指定fsp，则默认为0。

自MySQL 5.6.4起，对于那些允许使用小数秒部分的类型，其存储空间要求有所变化。对于非小数秒部分的存储，与前面各个类型的描述是一样的。对于小数秒部分的存储，所有类型都是一样的。下表列出了所需要的字节数。

| **小数秒精度** | **存储空间要求** |
| --- | --- |
| 0 | 0个字节 |
| 1、2 | 1个字节 |
| 3、4 | 2个字节 |
| 5、6 | 3个字节 |

MySQL 5.6.5引入了自动把当前时间戳作为初始值并进行更新的扩展支持。以前，这些属性最多只能用于表的一个TIMESTAMP列；现在，它们可以用于任何TIMESTAMP列，并且还可用于DATETIME列。

|  |
| --- |
| ==================================================  **DATE**  **含义**：日期值，格式为'CCYY-MM-DD'。  **可用属性**：除全局属性外，没有其他特殊属性。  **取值范围**：'1000-01-01'～'9999-12-31'。  **零值**：'0000-00-00'。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为'0000-00-00'。  **存储空间要求**：3个字节。  ==================================================  **DATETIME[(fsp)]**  **含义**：日期和时间值，格式为'CCYY-MM-DD hh:mm:ss[.uuuuuu]'。自MySQL 5.6.4起，允许使用fsp来指定精度为0～6的小数秒。如果省略，则fsp默认为0。  可用属性：在MySQL 5.6.5之前，除全局属性外，无任何属性可用。自MySQL 5.6.5起，DATETIME列可以包含**DEFAULT CURRENT\_TIMESTAMP或ON UPDATE CURRENT\_TIMESTAMP**，或者同时包含这两个属性。它们的含义与TIMESTAMP里的相同。  **取值范围**：'1000-01-01 00:00:00[.000000] '～'9999-12-31 23:59:59[.999999]'。  **零值**：'0000-00-00 00:00:00[.000000]'。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为'0000-00-00 00:00:00[.000000]'。  **存储空间要求**：在MySQL 5.6.4之前，为8个字节；那以后的版本，还需要外加5个字节，用于存储小数秒。  ==================================================  **TIME[(fsp)]**  **含义**：时间值，格式为'hh:mm:ss[.uuuuuu]'（对于负值，格式为'-hh:mm:ss[.uuuuuu]'）。自MySQL 5.6.4起，允许使用fsp来指定精度为0～6的小数秒。如果省略，则fsp默认为0。  可用属性：除全局属性外，没有其他特殊属性。  **取值范围**：'-838:59:59[.000000]'～'838:59:59[.000000]'。  **零值**：'00:00:00[.000000]'。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为'00:00:00[.000000]'。  **存储空间要求**：在MySQL 5.6.4之前，为3个字节；那以后的版本，还需要外加3个字节，用于存储小数秒。  **备注**：当有非法值插入TIME列时，虽然'00:00:00[.000000]'会被用作零值，但是它同时也是列的正常取值范围内的一个合法值。  ==================================================  **TIMESTAMP[(fsp)]**  **含义**：时间戳（日期和时间），格式为 'CCYY-MM-DDhh:mm:ss[.uuuuuu]'。自MySQL 5.6.4起，允许使用fsp来指定精度为0～6的小数秒。如果省略，则fsp默认为0。TIMESTAMP类型有几下几种特殊行为。   * 在往表的TIMESTAMP列里插入NULL值时，会插入当前的日期和时间，除非列被声明为允许NULL值。 * TIMESTAMP列可以有两个自动修改属性。   **自动初始化**：当创建新行时，该列的默认值为当前时间戳。  **自动更新**：当更新行时，修改该行里的任何其他列的值，都会使TIME  STAMP列被更新为修改发生时的日期和时间值。  在MySQL 5.6.5之前，每个表最多只能有一个TIMESTAMP列可以拥有这些属性，并且可以指定哪一个列会有此“殊荣”。自MySQL 5.6.5起，任何TIMESTAMP列都可以拥有这些属性，并且可以随意组合。  **可用属性**：在MySQL 5.6.5之前，一个表最多只能有一个TIMESTAMP列可以拥有DEFAULT CURRENT\_TIMESTAMP或ON UPDATE CURRENT\_TIMESTAMP属性，或者同时拥有这两个属性。不能为一个TIMESTAMP列设置一种属性，又为另一个TIMESTAMP列设置其他属性；同样，也不能将一个属性用于多个TIMESTAMP列。自MySQL 5.6.5起，所有的TIMESTAMP列都可以拥有这些属性，并且可以随意组合。  在创建新行时，如果没有为TIMESTAMP列提供值，则DEFAULT CURRENT\_ TIMESTAMP属性会使其被设置为当前的日期和时间。当行里的其他列值发生变化时，ON UPDATE CURRENT\_TIMESTAMP属性会使TIMESTAMP列被更新为当前的日期和时间。CURRENT\_TIMESTAMP()和NOW()函数都可以认为等同于CURRENT\_TIMESTAMP属性。  如果列定义里的初始关键字TIMESTAMP包含有一个fsp值，那么在用于指定自动属性的子句DEFAULT或ON UPDATE里（如果存在的话），也必须使用相同的值。  **可以指定一个DEFAULT常量**，从而为TIMESTAMP列分配一个固定的日期和时间值或零值。  如果指定NULL属性，则允许TIMESTAMP列存储NULL值。**如果没有指定此属性，则将NULL存储到TIMESTAMP列时，会将其设置为当前的日期和时间**。  **取值范围**：'1970-01-01 00:00:01[.000000]'～2038年（大约）。  **零值**：'0000-00-00 00:00:00[.000000]'。  **默认值**：在创建行时，如果TIMESTAMP列被设置为当前的日期和时间，则DESCRIBE和SHOW COLUMNS语句会把默认值显示为CURRENT\_TIMESTAMP；否则，会显示日期和时间的默认值。请参考对“可用属性”项的讨论。  **存储空间要求**：在MySQL 5.6.4之前，为4个字节；那以后的版本，还需要外加4个字节，用于存储小数秒。  ==================================================  **YEAR[(M)]**  **含义**：年份值。如果给出的话，M只能是2或者4，分别代表格式YY或CCYY。如果省略，则M的默认值为4。  可用属性：除全局属性外，没有其他特殊属性。  **取值范围**：对于YEAR(4)，范围是1901～2155，以及0000；对于YEAR(2)，范围相同，但是只显示最后两位数字。如果想要避免存储在YEAR(2)里的值出现歧义，则需要将这些值的范围限制在1970～2069；另一种选择是，使用YEAR(4)。  零值：对于YEAR(4)，零值为0000；对于YEAR(2)，零值为00。  **默认值**：如果列可以为NULL，则默认值为NULL；如果带有NOT NULL属性，则默认值为0000或00。  **存储空间要求**：1个字节。  **备注**：自MySQL 5.6.6起，YEAR(2)已被弃用。对于这样的列，可以使用YEAR(4)代替。 |

# MySQL处理数据值

## 处理列默认值

除了timestamp和datetime外，默认值必须为常量。他不能是表达式也不能引用其他列。如果某个列没有显示的包含default 子句，并且该列允许为NULL值，其默认值为NULL。否则，不允许为NULL，并且列表里没有显示DEFAULT子句，mysql处理方式如下：

* 如果没有启用了sql的严格模式，该列被设置成隐式默认值。
* 如果启用了严格模式，会出现错误。

列隐含默认值：

* 数字列（除了AUTO\_INCREAMENT）:0
* 时态类型：'0000-00-00），对于timestamp,datetime(mysql5.6)有点特殊
* 字符型（不包括enum,set）:空字符串。Enum(枚举第一个元素)，Set（空集合）

## 处理无效值

默认情况下，MySQL会按照一下规则来处理越界值和其他非正常值：

* 对于数值列或者time列，超出合法曲直的的那些值会被截断到取值范围最近的断点，并把结果存储起来。
* 除了time以外的其他时态类型列，非法值会被转化与该类型相一致的'零'值
* 对于字符串列（不包含ENUM或者SET类型），过长部分会被截取
* 给ENUM或者SET类型列进行赋值的时候，需要根据列定义里给出的合法取值列表进行。如果不是枚举类型的值赋值给ENUM列，那么列的值会变成出错成员（即与零值成员相对应的空字符串）。如果把包含非集合成员的子字符串的值赋给SET列，那么这些字符串会被清理，剩余成员被赋值

注意：发生上述转化过程，会发生警告。用show warnings 语句查看。

使用严格模式执行严格检查：

|  |
| --- |
| SET sql\_mode='STRICT\_ALL\_TABLES'  SET sql\_mode='STRICT\_TRANS\_TABLES' |

STRICT\_ALL\_TABLES：插入和修改第一行某个值无效或者缺失，会产生一个错误，语句会终止。如果插入和修改多行时候，如果发生错误，会抛出一个错误，并且语句终止执行，会导致部分更新。

STRICT\_TRANS\_TABLES：插入和修改第一行某个值无效或者缺失，会产生一个错误，语句会终止。多行操作也会导致部分更新的情况。

ERROR\_FOR\_DIVISION\_BY\_ZERO：阻止被除数为0的数据进入数据库。

NO\_ZERO\_DATE：阻止0日期值进入数据库。

NO\_ZERO\_IN\_NDATE：阻止月日部分为0 不完整日期值进入数据库。

注意：使用insert或者update语句中使用ignore关键字，那么因为无效导致错误的语句，只会发生警告。

# MySQL处理序列

## 通用的AUTO\_INCREAMENT属性

|  |
| --- |
| 1. 每个表只能由一个列具有auto\_increament属性，并且它应该为整数数据类型 2. 列必须建立索引 3. 列必须拥有NOT NULL约束条件 4. 把NULL值插入auto\_increament列将引发MySQL自动生成下一个序列编号。 5. 要获取最近生成的序号值，可以调用last\_insert\_id()函数（当前会话有效） 6. 插入一行时候，如果没有指定自增长列，默认插入NULL 7. 默认情况下，把0插入到auto\_increament列，等效于插入null；如果sql模式启用了NO\_AUTO\_VALUE\_ON\_ZERO模式会存储0。 8. 如果插入一行，auto\_increament列指定值如果在当前表中存在，则报错，否则成功插入，并把auto\_increament设置为当前值。 9. 对于某些存储引擎，从顶端删除的auto\_increament值可以被重用 |

## 存储引擎特有的AUTO\_INCREAMENT属性

### MyISAM表的AUTO\_INCREAMENT属性

|  |
| --- |
| 1. MyISAM表的序列一般是单调的，行被删除后，序列不能被重用，除以下情况：truncate table\_name 或者使用复合索引生成多个序列 2. MyISAM序列默认值为1，可以通过create table 的auto\_increament=n显示指定初始值。并且可以通过ALTER TABLE t\_name auto\_increament=n修改计数器的值。 |

### InnoDB表的AUTO\_INCREAMENT属性

|  |
| --- |
| 1. InnoDB序列默认值为1，可以通过create table 的auto\_increament=n显示指定初始值。并且可以通过ALTER TABLE t\_name auto\_increament=n修改计数器的值。 2. 从序列顶端删除的值通常不能重用；truncate table清空表，那么序列从1开始。 3. 事务回滚在序列中可能导致断裂 4. 不能使用复合索引生成多个独立的序列 |

### MEMORY表的AUTO\_INCREAMENT属性

|  |
| --- |
| 1. MEMORY序列默认值为1，可以通过create table 的auto\_increament=n显示指定初始值。并且可以通过ALTER TABLE t\_name auto\_increament=n修改计数器的值。 2. 从序列顶端删除的值通常不能重用；truncate table清空表，那么序列从1开始。 3. 不能使用复合索引生成多个独立的序列 |

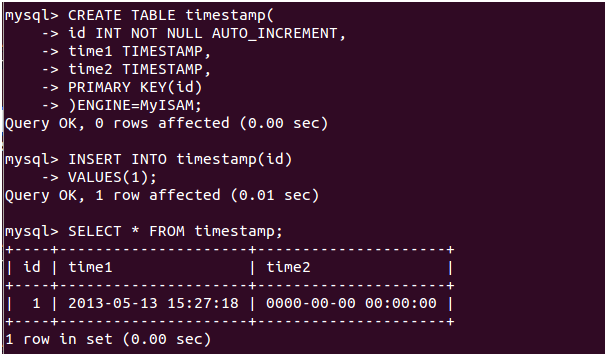
## 使用AUTO\_INCREAMENT属性要考虑的问题

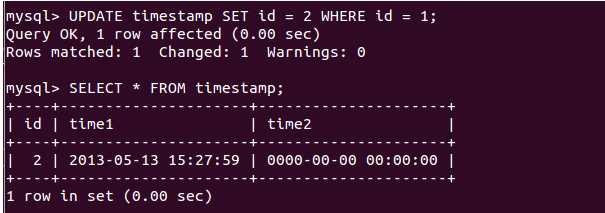
|  |
| --- |
| 1. AUTO\_INCREAMENT不支持非正数，可以把AUTO\_INCREAMENT列设置成UNSIGNED类型,得到双倍序列编号 2. AUTO\_INCREAMENT不能得到无穷尽的序列编号。 3. 使用TRUNCATE table 语句来清理某个表的内容，会使得计数器从1开始。 |

# MySQL 5.6 中 TIMESTAMP 的变化

在MySQL 5.6.6之前，TIMESTAMP的默认行为：

* TIMESTAMP列如果没有明确声明NULL属性，默认为NOT NULL。（而其他数据类型，如果没有显示声明为NOT NULL，则允许NULL值。）设置TIMESTAMP的列值为NULL，会自动存储为当前timestamp。
* 表中的第一个TIMESTAMP列，如果没有声明NULL属性、DEFAULT或者 ON UPDATE，会自动分配 DEFAULT CURRENT\_TIMESTAMP和ON UPDATE CURRENT\_TIMESTAMP 属性。
* 表中第二个TIMESTAMP列，如果没有声明为NULL或者DEFAULT子句，默认自动分配'0000-00-00 00:00:00′。插入行时没有指明改列的值，该列默认分配'0000-00-00 00:00:00′，且没有警告。



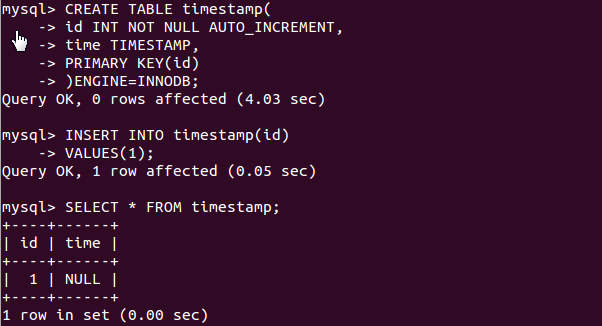


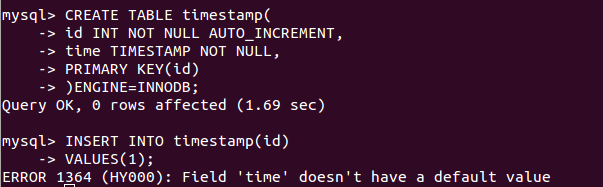
从MySQL5.6.6开始这种默认设置的方法被废弃了。在MySQL启动时会出现以下警告：

|  |
| --- |
| [Warning] TIMESTAMP with implicit DEFAULT value is deprecated.  Please use --explicit\_defaults\_for\_timestamp server option  (seedocumentation for more details). |
| 关闭警告，在my.cnf中加入：  [mysqld]  explicit\_defaults\_for\_timestamp=true |

重启MySQL后错误消失，这时TIMESTAMP的行为如下：

* TIMESTAMP如果没有显示声明NOT NULL，是允许NULL值的，可以直接设置改列为NULL，而没有默认填充行为。
* TIMESTAMP不会默认分配DEFAULT CURRENT\_TIMESTAMP 和 ON UPDATE CURRENT\_TIMESTAMP属性。
* 声明为NOT NULL且没有默认子句的TIMESTAMP列是没有默认值的。往数据表中插入列，又没有给TIMESTAMP列赋值时，如果是严格SQL模式，会抛出一个错误，如果严格SQL模式没有启用，该列会赋值为'0000-00-00 00:00:00′，同时出现一个警告。（这和MySQL处理其他时间类型数据一样，如DATETIME）





创建时间的列：

create\_ts timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP ,

update\_ts timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,