# MySQL 中的数据类型介绍

## 1、概述

    要了解一个**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "_blank)**，我们也必须了解其支持的数据类型。

[**mysql**](http://lib.csdn.net/base/mysql)支持所有标准的SQL数据类型，主要分3类:

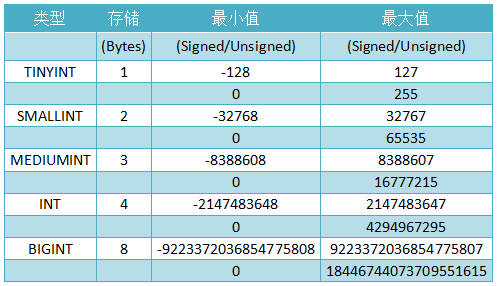
* 数值类型
* 字符串类型
* 时间日期类型

    另一类是几何数据类型，用的不多，也没多介绍。    下面大、小标题后括号内的数组表示其含有的类型个数。下面所有结论都经过本人使用MySql Workbench编写SQL验证过或来自官网。

## 2、数值类型（12）

### 2.1、整数类型（6）

    一张图就能解释清楚了：



    INTEGER同INT。

### 2.2、定点数（2）

    DECIMAL和NUMERIC类型在MySQL中视为相同的类型。它们用于保存必须为确切精度的值。

    使用方式如下：

1. salary DECIMAL(5,2)

    下面的介绍将基于上面这个例子。

    我们看到其中有两个参数，即DECIMAL(M,D)，其中M表示十进制数字总的个数，D表示小数点后面数字的位数，上例中的取值范围为-999.99~999.99。

    如果存储时，整数部分超出了范围（如上面的例子中，添加数值为1000.01），MySql就会报错，不允许存这样的值。

    如果存储时，小数点部分若超出范围，就分以下情况：

* 若四舍五入后，整数部分没有超出范围，则只警告，但能成功操作并四舍五入删除多余的小数位后保存。如999.994实际被保存为999.99。
* 若四舍五入后，整数部分超出范围，则MySql报错，并拒绝处理。如999.995和-999.995都会报错。

    M的默认取值为10，D默认取值为0。如果创建表时，某字段定义为decimal类型不带任何参数，等同于decimal(10,0)。带一个参数时，D取默认值。

* M的取值范围为1~65，取0时会被设为默认值，超出范围会报错。
* D的取值范围为0~30，而且必须<=M，超出范围会报错。

    所以，很显然，当M=65，D=0时，可以取得最大和最小值。

    已经解释很详细了，如还不清楚，请回复。

### 2.3、浮点数（3）

    浮点数是用来表示实数的一种方法，它用 M(尾数) \* B( 基数)的E(指数）次方来表示实数，相对于定点数来说，在长度一定的情况下，具有表示数据范围大的特点。但同时也存在误差问题。

    如果希望保证值比较准确，推荐使用定点数数据类型。

    MySql中的浮点类型有float，double和real。他们定义方式为：FLOAT(M,D) 、 REAL(M,D) 、 DOUBLE PRECISION(M,D)。

    REAL就是DOUBLE ，如果SQL服务器模式包括REAL\_AS\_FLOAT选项，REAL是FLOAT的同义词而不是DOUBLE的同义词。

    “(M,D)”表示该值一共显示M位整数，其中D位位于小数点后面。例如，定义为FLOAT(7,4)的一个列可以显示为-999.9999。MySQL保存值时进行四舍五入，因此如果在FLOAT(7,4)列内插入999.00009，近似结果是999.0001。

    FLOAT和DOUBLE中的M和D的取值默认都为0，即除了最大最小值，不限制位数。允许的值理论上是-1.7976931348623157E+308~-2.2250738585072014E-308、0和2.2250738585072014E-308~1.7976931348623157E+308。M、D范围如下（MySql5.7实测，与IEEE标准计算的实际是不同的，下面介绍）：

* M取值范围为0~255。FLOAT只保证6位有效数字的准确性，所以FLOAT(M,D)中，M<=6时，数字通常是准确的。如果M和D都有明确定义，其超出范围后的处理同decimal。
* D取值范围为0~30，同时必须<=M。double只保证16位有效数字的准确性，所以DOUBLE(M,D)中，M<=16时，数字通常是准确的。如果M和D都有明确定义，其超出范围后的处理同decimal。

    FLOAT和DOUBLE中，若M的定义分别超出7和17，则多出的有效数字部分，取值是不定的，通常数值上会发生错误。因为浮点数是不准确的，所以我们要避免使用“=”来判断两个数是否相等。

    MySql中的浮点数遵循IEEE 754标准。

    内存中，FLOAT占4-byte（1位符号位 8位表示指数 23位表示尾数），DOUBLE占8-byte（1位符号位 11位表示指数 52位表示尾数）。IEEE754标准还对尾数的格式做了规范：d.dddddd...，小数点左面只有1位且不能为零，计算机内部是二进制，因此，尾数小数点左面部分总是1。显然，这个1可以省去，以提高尾数的精度。由上可知，单精度浮点数的尾数是用24bit表示的，双精度浮点数的尾数是用53bit表示的。所以就能算出取值范围和准确的有效位数了，但MySql中其实略有不同。

### 2.4、BIT（1）

    BIT数据类型可用来保存位字段值。BIT(M)类型允许存储M位值。M范围为1~64，默认为1。

    BIT其实就是存入二进制的值，类似010110。

    如果存入一个BIT类型的值，位数少于M值，则左补0.

    如果存入一个BIT类型的值，位数多于M值，MySQL的操作取决于此时有效的SQL模式：

* 如果模式未设置，MySQL将值裁剪到范围的相应端点，并保存裁减好的值。
* 如果模式设置为traditional(“严格模式”)，超出范围的值将被拒绝并提示错误，并且根据SQL标准插入会失败。

    下面是官方示例：

1. mysql> CREATE TABLE t (b BIT(8));
2. mysql> INSERT INTO t SET b = b'11111111';
3. mysql> INSERT INTO t SET b = b'1010';
4. mysql> INSERT INTO t SET b = b'0101';
5. mysql> SELECT b+0, BIN(b+0), OCT(b+0), HEX(b+0) FROM t;
6. +------+----------+----------+----------+
7. | b+0 | BIN(b+0) | OCT(b+0) | HEX(b+0) |
8. +------+----------+----------+----------+
9. | 255 | 11111111 | 377 | FF |
10. | 10 | 1010 | 12 | A |
11. | 5 | 101 | 5 | 5 |
12. +------+----------+----------+----------+

## 3、字符串类型(14)

    字符串类型指CHAR、VARCHAR、BINARY、VARBINARY、BLOB、TEXT、ENUM和SET。

### 3.1、CHAR和VARCHAR类型（2）

    CHAR和VARCHAR类型声明的长度表示你想要保存的最大字符数。例如，CHAR(30)可以占用30个字符。默认长度都为255。

    CHAR列的长度固定为创建表时声明的长度。长度可以为从0到255的任何值。当保存CHAR值时，在它们的右边填充空格以达到指定的长度。当检索到CHAR值时，尾部的空格被删除掉，所以，我们在存储时字符串右边不能有空格，即使有，查询出来后也会被删除。在存储或检索过程中不进行大小写转换。

    所以当char类型的字段为唯一值时，添加的值是否已经存在以不包含末尾空格（可能有多个空格）的值确定，比较时会在末尾补满空格后与现已存在的值比较。

    VARCHAR列中的值为可变长字符串。长度可以指定为0到65,535之间的值（实际可指定的最大长度与编码和其他字段有关，比如，本人MySql使用utf-8编码格式，大小为标准格式大小的2倍，仅有一个varchar字段时实测最大值仅21844，如果添加一个char(3)，则最大取值减少3。整体最大长度是65,532字节）。

    同CHAR对比，VARCHAR值保存时只保存需要的字符数，另加一个字节来记录长度(如果列声明的长度超过255，则使用两个字节)。

    VARCHAR值保存时不进行填充。当值保存和检索时尾部的空格仍保留，符合标准SQL。

    如果分配给CHAR或VARCHAR列的值超过列的最大长度，则对值进行裁剪以使其适合。如果被裁掉的字符不是空格，则会产生一条警告。如果裁剪非空格字符，则会造成错误(而不是警告)并通过使用严格SQL模式禁用值的插入。

    下面显示了将各种字符串值保存到CHAR(4)和VARCHAR(4)列后的结果：



    表中最后一行的值只适用在不使用严格模式时；如果MySQL运行使用严格模式，超过列长度的值不保存，并且会出现错误。

    因为空格的原因，相同的值存入到长度都足够的varvhar和char中，取出可能会不同，比如"a"和"a  "。

### 3.1、BINARY和VARBINARY类型（2）

    BINARY和VARBINARY类型类似于CHAR和VARCHAR类型，但是不同的是，它们存储的不是字符字符串，而是二进制串。所以它们没有字符集，并且排序和比较基于列值字节的数值值。

    当保存BINARY值时，在它们右边填充0x00(零字节)值以达到指定长度。取值时不删除尾部的字节。比较时所有字节很重要（因为空格和0x00是不同的，0x00<空格），包括ORDER BY和DISTINCT操作。比如插入'a '会变成'a \0'。

    对于VARBINARY，插入时不填充字符，选择时不裁剪字节。比较时所有字节很重要。

    当类型为BINARY的字段为主键时，应考虑上面介绍的存储方式。

### 3.2、BLOB和TEXT类型（8）

    BLOB是一个二进制大对象，可以容纳可变数量的数据。有4种BLOB类型：TINYBLOB、BLOB、MEDIUMBLOB和LONGBLOB。它们只是可容纳值的最大长度不同。

    有4种TEXT类型：TINYTEXT、TEXT、MEDIUMTEXT和LONGTEXT。这些对应4种BLOB类型，有相同的最大长度和存储需求。

    BLOB列被视为二进制字符串。TEXT列被视为字符字符串，类似CHAR和BINARY。

    在TEXT或BLOB列的存储或检索过程中，不存在大小写转换。

    未运行在严格模式时，如果你为BLOB或TEXT列分配一个超过该列类型的最大长度的值，值被截取以保证适合。如果截掉的字符不是空格，将会产生一条警告。使用严格SQL模式，会产生错误，并且值将被拒绝而不是截取并给出警告。

    在大多数方面，可以将BLOB列视为能够足够大的VARBINARY列。同样，可以将TEXT列视为VARCHAR列。

    BLOB和TEXT在以下几个方面不同于VARBINARY和VARCHAR：

* 当保存或检索BLOB和TEXT列的值时不删除尾部空格。(这与VARBINARY和VARCHAR列相同）。
* 比较时将用空格对TEXT进行扩充以适合比较的对象，正如CHAR和VARCHAR。
* 对于BLOB和TEXT列的索引，必须指定索引前缀的长度。对于CHAR和VARCHAR，前缀长度是可选的。
* BLOB和TEXT列不能有默认值。

    MySQL Connector/ODBC将BLOB值定义为LONGVARBINARY，将TEXT值定义为LONGVARCHAR。

    BLOB或TEXT对象的最大大小由其类型确定，但在客户端和服务器之间实际可以传递的最大值由可用内存数量和通信缓存区大小确定。你可以通过更改max\_allowed\_packet变量的值更改消息缓存区的大小，但必须同时修改服务器和客户端程序。

    每个BLOB或TEXT值分别由内部分配的对象表示。

    它们（TEXT和BLOB同）的长度：

* Tiny：最大长度255个字符(2^8-1)
* BLOB或TEXT：最大长度65535个字符(2^16-1)
* Medium：最大长度16777215个字符(2^24-1)
* LongText 最大长度4294967295个字符(2^32-1)

    实际长度与编码有关，比如utf-8的会减半。

### 3.3、ENUM（1）

    MySql中的ENUM是一个字符串对象，其值来自表创建时在列规定中显式枚举的一列值。

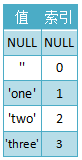
    可以插入空字符串""和NULL：

* 如果你将一个非法值插入ENUM(也就是说，允许的值列之外的字符串)，将插入空字符串以作为特殊错误值。该字符串与“普通”空字符串不同，该字符串有数值值0。
* 如果将ENUM列声明为允许NULL，NULL值则为该列的一个有效值，并且默认值为NULL。如果ENUM列被声明为NOT NULL，其默认值为允许的值列的第1个元素。

    值的索引规则如下：

* 来自列规定的允许的值列中的值从1开始编号。
* 空字符串错误值的索引值是0。所以，可以使用下面的SELECT语句来找出分配了非法ENUM值的行：mysql> SELECT \* FROM tbl\_name WHERE enum\_col=0;
* NULL值的索引是NULL。

    如下例：



    ENUM最多可以有65,535个元素。当创建表时，ENUM成员值的尾部空格将自动被删除。

    使用方式：

1. CREATE TABLE shirts (
2. name VARCHAR(40),
3. size ENUM('x-small', 'small', 'medium', 'large', 'x-large')
4. );
5. INSERT INTO shirts (name, size) VALUES ('dress shirt','large'),('t-shirt','medium'),('polo shirt','small');
6. SELECT name, size FROM shirts WHERE size = 'medium';
7. UPDATE shirts SET size = 'small' WHERE size = 'large';

    如果将返回值设为数值，将返回索引值，比如讲上面的查询语句改为：

1. SELECT name, size+0 FROM shirts WHERE size = 'medium';

    如果将一个数字保存到ENUM列，数字被视为索引，并且保存的值是该索引对应的枚举成员(这不适合LOAD DATA，它将所有输入视为字符串）。不建议使用类似数字的枚举值来定义一个ENUM列，因为这很容易引起混淆。

    ENUM值根据索引编号进行排序）。例如，对于ENUM('a'，'b')，'a'排在'b'前面，但对于ENUM('b'，'a')，'b'排在'a'前面。空字符串排在非空字符串前面，并且NULL值排在所有其它枚举值前面。要想防止意想不到的结果，按字母顺序规定ENUM列。还可以使用GROUP BY CAST(col AS CHAR)或GROUP BY CONCAT(col)来确保按照词汇对列进行排序而不是用索引数字。

### 3.4、SET类型(1)

    SET是一个字符串对象，可以有零或多个值，其值来自表创建时规定的允许的一列值。指定包括多个SET成员的SET列值时各成员之间用逗号(‘,’)间隔开。例如，指定为SET('one', 'two') NOT NULL的列可以有下面的任何值：

* ''
* 'one'
* 'two'
* 'one,two'

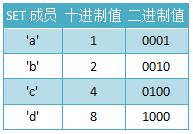
    SET最多可以设置64个值。创建表时，SET成员值的尾部空格将自动被删除。检索时，保存在SET列的值使用列定义中所使用的大小写来显示。

    MySQL用数字保存SET值，所保存值的低阶位对应第1个SET成员。如果在数值上下文中检索一个SET值，检索的值的位设置对应组成列值的SET成员。

    例如，可以这样从一个SET列检索数值值：

1. mysql> SELECT set\_col+0 FROM tbl\_name;

    如果将一个数字保存到SET列中，数字的二进制的1的位置确定了列值中的SET成员。对于指定为SET('a','b','c','d')的列，成员有下面的十进制和二进制值：



    如果你为该列分配一个值9，其二进制形式为1001，因此第1个和第4个SET值成员'a'和'd'被选择，结果值为 'a,d'。

    对于包含多个SET元素的值，当插入值时元素所列的顺序并不重要。在值中一个给定的元素列了多少次也不重要。当以后检索该值时，值中的每个元素出现一次，根据表创建时指定的顺序列出元素。例如，假定某个列指定为SET('a','b','c','d')：

1. CREATE TABLE myset (col SET('a', 'b', 'c', 'd'));
2. INSERT INTO myset (col) VALUES ('a,d'), ('d,a'), ('a,d,a'), ('a,d,d'), ('d,a,d');
3. SELECT \*,col+0 FROM myset;
4. SELECT \*,col+0 FROM myset where col='a,b';

    结果：

1. a,d 9
2. a,d 9
3. a,d 9
4. a,d 9
5. a,d 9

    SET值按数字顺序排序。NULL值排在非NULL SET值的前面。

    通常情况，可以使用FIND\_IN\_SET()函数或LIKE操作符搜索SET值：

    mysql> SELECT \* FROM tbl\_name WHERE FIND\_IN\_SET('value',set\_col)>0;

    mysql> SELECT \* FROM tbl\_name WHERE set\_col LIKE '%value%';

    第1个语句找出SET\_col包含value set成员的行。第2个类似，但有所不同：它在其它地方找出set\_col包含value的行，甚至是在另一个SET成员的子字符串中。

    下面的语句也是合法的：

    mysql> SELECT \* FROM tbl\_name WHERE set\_col & 1;

    mysql> SELECT \* FROM tbl\_name WHERE set\_col = 'val1,val2';

    第1个语句寻找包含第1个set成员的值。第2个语句寻找一个确切匹配的值。应注意第2类的比较。将set值与'val1,val2'比较返回的结果与同'val2,val1'比较返回的结果不同。指定值时的顺序应与在列定义中所列的顺序相同。

    如果想要为SET列确定所有可能的值，使用SHOW COLUMNS FROM tbl\_name LIKE set\_col并解析输出中第2列的SET定义。

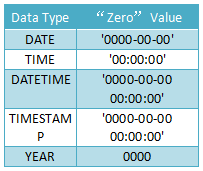
**有什么实际应用呢？**

    比如我们设定用户的权限控制，一个用户可能会有多种权限，我们使用所有权限创建一个SET类型的字段，我们不需要用一系列int来定义各种权限了，直接使用一个SET字段即可：

1. /\*
2. 用户权限permission表
3. \*/
4. create table user\_permission(
5. id int UNSIGNED not null auto\_increment,
6. user\_id int not null ,
7. permission set('阅读','评论','发帖') not null,
8. primary key(id),
9. unique (user\_id)
10. );
11. desc user\_permission;
12. insert into user\_permission values (0,1,'阅读'),(0,2,'阅读'),(0,3,'阅读,评论');
13. insert into user\_permission values (0,4,'阅读,评论,发帖');
14. select \*,permission+0 from user\_permission;
15. select permission from user\_permission where user\_id=1;
16. select \* from user\_permission where permission & 10;
17. SELECT \* FROM user\_permission WHERE FIND\_IN\_SET('评论',permission)>0;

## 4、时间日期类型（5）

    他们的“0”值如下：



### 4.1、DATE, DATETIME, 和TIMESTAMP类型（3）

    这三者其实是关联的，都用来表示日期或时间。

    当你需要同时包含日期和时间信息的值时则使用DATETIME类型。MySQL以'YYYY-MM-DD HH:MM:SS'格式检索和显示DATETIME值。支持的范围为'1000-01-01 00:00:00'到'9999-12-31 23:59:59'。

    当你只需要日期值而不需要时间部分时应使用DATE类型。MySQL用'YYYY-MM-DD'格式检索和显示DATE值。支持的范围是'1000-01-01'到 '9999-12-31'。

    TIMESTAMP类型同样包含日期和时间，范围从'1970-01-01 00:00:01' UTC 到'2038-01-19 03:14:07' UTC。

    可以使用任何常见格式指定DATETIME、DATE和TIMESTAMP值：

* 'YYYY-MM-DD HH:MM:SS'或'YY-MM-DD HH:MM:SS'格式的字符串。允许“不严格”语法：任何标点符都可以用做日期部分或时间部分之间的间割符。例如，'98-12-31 11:30:45'、'98.12.31 11+30+45'、'98/12/31 11\*30\*45'和'98@12@31 11^30^45'是等价的。
* 'YYYY-MM-DD'或'YY-MM-DD'格式的字符串。这里也允许使用“不严格的”语法。例如，'98-12-31'、'98.12.31'、'98/12/31'和'98@12@31'是等价的。
* YYYYMMDDHHMMSS'或'YYMMDDHHMMSS'格式的没有间割符的字符串，假定字符串对于日期类型是有意义的。例如，'19970523091528'和'970523091528'被解释为'1997-05-23 09:15:28'，但'971122129015'是不合法的(它有一个没有意义的分钟部分)，将变为'0000-00-00 00:00:00'。
* 'YYYYMMDD'或'YYMMDD'格式的没有间割符的字符串，假定字符串对于日期类型是有意义的。例如，'19970523'和'970523'被解释为 '1997-05-23'，但'971332'是不合法的(它有一个没有意义的月和日部分)，将变为'0000-00-00'。
* YYYYMMDDHHMMSS或YYMMDDHHMMSS格式的数字，假定数字对于日期类型是有意义的。例如，19830905132800和830905132800被解释为 '1983-09-05 13:28:00'。
* YYYYMMDD或YYMMDD格式的数字，假定数字对于日期类型是有意义的。例如，19830905和830905被解释为'1983-09-05'。
* 函数返回的结果，其值适合DATETIME、DATE或者TIMESTAMP上下文，例如NOW()或CURRENT\_DATE。

    对于包括日期部分间割符的字符串值，如果日和月的值小于10，不需要指定两位数。'1979-6-9'与'1979-06-09'是相同的。同样，对于包括时间部分间割符的字符串值，如果时、分和秒的值小于10，不需要指定两位数。'1979-10-30 1:2:3'与'1979-10-30 01:02:03'相同。

    数字值应为6、8、12或者14位长。如果一个数值是8或14位长，则假定为YYYYMMDD或YYYYMMDDHHMMSS格式，前4位数表示年。如果数字 是6或12位长，则假定为YYMMDD或YYMMDDHHMMSS格式，前2位数表示年。其它数字被解释为仿佛用零填充到了最近的长度。

    指定为非限定符字符串的值使用给定的长度进行解释。如果字符串为8或14字符长，前4位数表示年。否则，前2位数表示年。从左向右解释字符串内出现的各部分，以发现年、月、日、小时、分和秒值。这说明不应使用少于6字符的字符串。例如，如果你指定'9903'，认为它表示1999年3月，MySQL将在你的表内插入一个“零”日期值。这是因为年和月值是99和03，但日部分完全丢失，因此该值不是一个合法的日期。但是，可以明显指定一个零值来代表缺少的月或日部分。例如，可以使用'990300'来插入值'1999-03-00'。

    可以将一个日期类型的值分配给一个不同的日期类型。但是，值可能会更改或丢失一些信息：

* 如果你为一个DATETIME或TIMESTAMP对象分配一个DATE值，结果值的时间部分被设置为'00:00:00'，因为DATE值未包含时间信息。
* 如果你为一个DATE对象分配一个DATETIME或TIMESTAMP值，结果值的时间部分被删除，因为DATE值未包含时间信息。
* 记住尽管可以使用相同的格式指定DATETIME、DATE和TIMESTAMP值，不同类型的值的范围却不同。例如，TIMESTAMP值不能早于1970或晚于2037。这说明一个日期，例如'1968-01-01'，虽然对于DATETIME或DATE值是有效的，但对于TIMESTAMP值却无效，如果分配给这样一个对象将被转换为0。

    当指定日期值时请注意某些缺陷：

* 指定为字符串的值允许的非严格格式可能会欺骗。例如，值'10:11:12'由于‘:’间割符看上去可能象时间值，但如果用于日期上下文值则被解释为年'2010-11-12'。值'10:45:15'被转换为'0000-00-00'因为'45'不是合法月。
* 在非严格模式，MySQL服务器只对日期的合法性进行基本检查：年、月和日的范围分别是1000到9999、00到12和00到31。任何包含超出这些范围的部分的日期被转换成'0000-00-00'。请注意仍然允许你保存非法日期，例如'2002-04-31'。要想确保不使用严格模式时日期有效，应检查应用程序。 在严格模式，非法日期不被接受，并且不转换。
* 包含两位年值的日期会令人模糊，因为世纪不知道。MySQL使用以下规则解释两位年值： o 00-69范围的年值转换为2000-2069。 o 70-99范围的年值转换为1970-1999。

    各种相关操作：

### 4.2、TIME类型（1）

    MySQL以'HH:MM:SS'格式检索和显示TIME值(或对于大的小时值采用'HHH:MM:SS'格式)。

    TIME值的范围可以从'-838:59:59'到'838:59:59'。小时部分会因此大的原因是TIME类型不仅可以用于表示一天的时间(必须小于24小时)，还可能为某个事件过去的时间或两个事件之间的时间间隔(可以大于24小时，或者甚至为负)。

    你可以用各种格式指定TIME值：

* 'D HH:MM:SS.fraction'格式的字符串。还可以使用下面任何一种“非严格”语法：'HH:MM:SS.fraction'、'HH:MM:SS'、'HH:MM'、'D HH:MM:SS'、'D HH:MM'、'D HH'或'SS'。这里D表示日，可以取0到34之间的值。请注意MySQL还不保存分数。
* 'HHMMSS'格式的没有间割符的字符串，假定是有意义的时间。例如，'101112'被理解为'10:11:12'，但'109712'是不合法的(它有一个没有意义的分钟部分)，将变为'00:00:00'。
* HHMMSS格式的数值，假定是有意义的时间。例如，101112被理解为'10:11:12'。下面格式也可以理解：SS、MMSS、HHMMSS、HHMMSS.fraction。请注意MySQL还不保存分数。
* 函数返回的结果，其值适合TIME上下文，例如CURRENT\_TIME。

    对于指定为包括时间部分间割符的字符串的TIME值，如果时、分或者秒值小于10，则不需要指定两位数。'8:3:2'与'08:03:02'相同。

    为TIME列分配简写值时应注意。没有冒号，MySQL解释值时假定最右边的两位表示秒。(MySQL解释TIME值为过去的时间而不是当天的时间）。例如，你可能认为'1112'和1112表示'11:12:00'(11点过12分)，但MySQL将它们解释为'00:11:12'(11分，12 秒)。同样，'12'和12 被解释为 '00:00:12'。相反，TIME值中使用冒号则肯定被看作当天的时间。也就是说，'11:12'表示'11:12:00'，而不是'00:11:12'。

    超出TIME范围但合法的值被裁为范围最接近的端点。例如，'-850:00:00'和'850:00:00'被转换为'-838:59:59'和'838:59:59'。

    无效TIME值被转换为'00:00:00'。请注意由于'00:00:00'本身是一个合法TIME值，只从表内保存的一个'00:00:00'值还不能说出原来的值是 '00:00:00'还是不合法的值。

### 4.3、YEAR类型（1）

    YEAR类型是一个单字节类型用于表示年。

    MySQL以YYYY格式检索和显示YEAR值。范围是1901到2155。

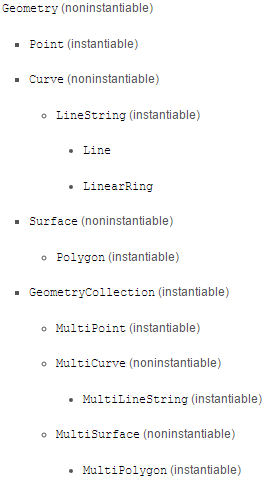
    可以指定各种格式的YEAR值：

* 四位字符串，范围为'1901'到'2155'。
* 四位数字，范围为1901到2155。
* 两位字符串，范围为'00'到'99'。'00'到'69'和'70'到'99'范围的值被转换为2000到2069和1970到1999范围的YEAR值。
* 两位整数，范围为1到99。1到69和70到99范围的值被转换为2001到2069和1970到1999范围的YEAR值。请注意两位整数范围与两位字符串范围稍有不同，因为你不能直接将零指定为数字并将它解释为2000。你必须将它指定为一个字符串'0'或'00'或它被解释为0000。
* 函数返回的结果，其值适合YEAR上下文，例如NOW()。

    非法YEAR值被转换为0000。

### 5、几何类型（8）

    几何类型层次结构如下：

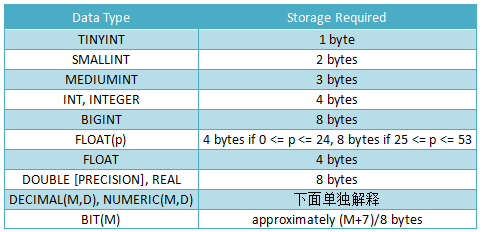


    到用的时候再说吧。

    官方文档：<http://dev.mysql.com/doc/refman/5.7/en/opengis-geometry-model.html>

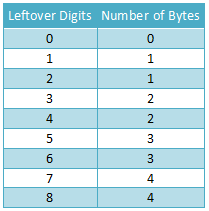
## 6、各种类型占用的存储

### 6.1、数值类型

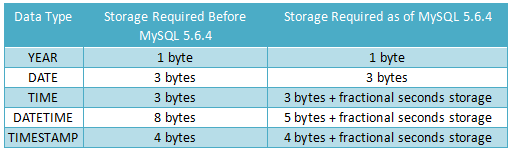


    定点数的比较特殊，而且与具体版本也有关系，此处单独解释：

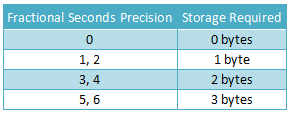
    使用二进制格式将9个十进制(基于10)数压缩为4个字节来表示DECIMAL列值。每个值的整数和分数部分的存储分别确定。每个9位数的倍数需要4个字节，并且“剩余的”位需要4个字节的一部分。下表给出了超出位数的存储需求：



### 6.2、时间日期

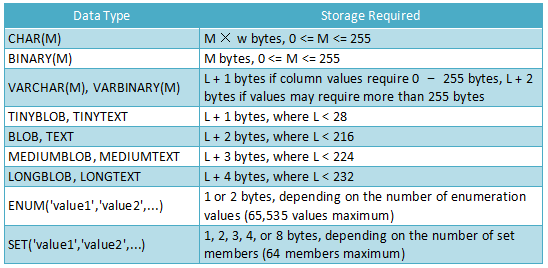


    从版本5.6.4开始，存储需求就有所改变，根据精度而定。不确定部分需要的存储如下：



    比如，TIME(0), TIME(2), TIME(4), 和TIME(6) 分别使用3, 4, 5, 6 bytes。

### 6.3、字符串



## 7、类型的选择

    为了优化存储，在任何情况下均应使用最精确的类型。

    例如，如果列的值的范围为从1到99999，若使用整数，则MEDIUMINT UNSIGNED是好的类型。在所有可以表示该列值的类型中，该类型使用的存储最少。

    用精度为65位十进制数(基于10)对DECIMAL 列进行所有基本计算(+、-、\*、/)。

    使用双精度操作对DECIMAL值进行计算。如果准确度不是太重要或如果速度为最高优先级，DOUBLE类型即足够了。为了达到高精度，可以转换到保存在BIGINT中的定点类型。这样可以用64位整数进行所有计算，根据需要将结果转换回浮点值。

## 8、使用其他数据库的SQL语句

    为了使用为其它数据库编写的SQL执行代码，MySQL按照下表所示对列类型进行映射。通过这些映射，可以很容易地从其它数据库引擎将表定义导入到MySQL中：

