

Mysql Data

[零 参考资料 2](#_Toc46494870)

[一 Character Sets 2](#_Toc46494871)

[1 Character Sets & Collations 2](#_Toc46494872)

[2 Unicode 2](#_Toc46494873)

[二 Data Types 3](#_Toc46494874)

[1 数字 3](#_Toc46494875)

[1.1 概述 3](#_Toc46494876)

[1.2 整数 3](#_Toc46494877)

[1.3 Fixed-Point Types 4](#_Toc46494878)

[1.4 浮点 4](#_Toc46494879)

[1.5 其他 5](#_Toc46494880)

[2 时间 6](#_Toc46494881)

[2.1 概述 6](#_Toc46494882)

[2.2 DATE/DATATIME/TIMESTAMP 6](#_Toc46494883)

[2.3 TIME/YEAR 8](#_Toc46494884)

[3 字符 8](#_Toc46494885)

[3.1 概述 9](#_Toc46494886)

[3.2 CHAR & VARCHAR 9](#_Toc46494887)

[3.3 BINARY & VARBINARY 10](#_Toc46494888)

[3.4 BLOB & TEXT 11](#_Toc46494889)

[3.5 ENUM & SET 12](#_Toc46494890)

[4 JSON 13](#_Toc46494891)

[5 Spatial Data Type 13](#_Toc46494892)

# 零 参考资料

《高性能Mysql 第三版》

<https://dev.mysql.com/doc/refman/5.7/en/charset.html>

<https://dev.mysql.com/doc/refman/5.7/en/data-types.html>

<https://dev.mysql.com/doc/refman/5.7/en/literals.html>

<https://dev.mysql.com/doc/refman/5.7/en/precision-math.html>

<https://dev.mysql.com/doc/internals/en/algorithms.html>

# Character Sets

<https://dev.mysql.com/doc/refman/5.7/en/charset.html>

## Character Sets & Collations

* + - 1. Mysql server 默认Character Set是latin1，collation是latin1\_swedish\_ci
      2. 每个Character Set至少支持一种collation，可以通过SHOW CHARACTER SET查看
      3. collation有以下特征
         1. 不同Character Set之间没有相同的collation
         2. 每个charater set都有默认的collation
         3. collation名称组成：characterSet\_
      4. ASCII是Unicode的子集，支持Unicode的U+0000 to U+007F的字符，因此ASCII字符可以完整的转为Unicode字符以及其他

## Unicode

<https://dev.mysql.com/doc/refman/5.7/en/charset-unicode-conversion.html>

* + - 1. 支持Basic Multilingual Plane(BMP)
         1. code值，0 to 65535（U+0000 to U+FFFF）
         2. 可用8，16，24bits（1 to 3byte）可变长度编码
         3. 也可以用16bits的固定长度编码
         4. 对所有主流语言的字符足够使用
      2. 支持BMP之外的U+10000 to U+10FFFF的字符：
         1. code值：U+10000 to U+10FFFF
         2. 每字符最多有4byte
      3. UTF-8(Unicode Transformation Format with 8-bit units):
         1. 根据RFC 3629标准实现
         2. 主要思路是：用不同长度字节序列编码不同Unicode字符
         3. 每个字符可以有 1 to 4byte

基本的latin 字母、数字、标点符号用1 byte

欧洲、中东语言中的非latin字母用2 byte

中日韩等象形文字用3 to 4 byte

* + - 1. surrogate机制：当一个字符编码值大于0xffff时
         1. 取10bit与0xd800相加，结果放在头16bit
         2. 剩余的bit与0xdc00相加，结果放在后16bit
         3. 结果的32bit，头16bit值为0xd800 to 0xdbff,尾 16bit值为 0xdc00 to 0xdbff
      2. Mysql支持的Unicode
         1. 采用不带BOM(byte order mark) 的big-endian方式实现UCS-2/UTF-16/UTF-32
         2. UTF-8不带BOM

| **Character Set** | **说明** |
| --- | --- |
| utf8mb3 | 1. 只支持BMP 2. 每字符大小1 to 3 bytes 3. mysql会将语句中utf8mb3转为utf8 |
| utf8 | utf8mb3的别名 |
| ucs2 | 1. 即UCS-2 2. 只支持BMP 3. 每字符固定大小2 bytes 4. 与utf8mb3有完全相同的字符集编码和指令 |
| utf8mb4 | 1. 支持BMP and supplementary 2. 每字符大小：1 to 4 bytes 3. 是utf8mb3超集，支持BMP部分，编码值与utf8mb3相同 4. supplementary部分字符，采用4 byte存储 5. SELECT CONCAT(utf8mb3\_col, utf8mb4\_col)，结果值采用utf8mb4 6. WHERE utf8mb3\_tbl.utf8mb3\_col = utf8mb4\_tbl.utf8mb4\_col; 采用utf8mb4字段的collation 7. 完整的UTF-8 |
| utf16 | 1. 即UTF-16 2. 支持BMP and supplementary 3. 每字符大小2 or 4 bytes 4. 是ucs2的补充字符扩展 5. 采用big-endian编码方式 6. supplementary 部分字符采用32bit序列的surrogate机制展示 |
| utf16le | 1. 支持BMP and supplementary 2. 每字符大小2 or 4 bytes 3. 与utf16差异是，用的是little-endian编码方式 |
| utf32 | 1. 即UTF-32 2. 支持BMP and supplementary 3. 每字符大小固定4 bytes |

# Data Types

## 数字

<https://dev.mysql.com/doc/refman/5.7/en/numeric-types.html>

<https://dev.mysql.com/doc/refman/5.7/en/number-literals.html>

### 概述

### 整数

|  |  |
| --- | --- |
| 类型 | 说明 |
| TINYINT | TINYINT[(M)] [UNSIGNED] [ZEROFILL]  存储大小：1byte  有符号取值：-128～127  无符号取值：0-25 |
| SMALLINT | SMALLINT[(M)] [UNSIGNED] [ZEROFILL]  存储大小：2byte  有符号取值：-32768～32767  无符号取：0-65535 |
| MEDUIMINT | MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]  存储大小：3byte  有符号取值：-8388608 ～ 8388607  无符号取：0 to 16777215 |
| INT | INT[(M)] [UNSIGNED] [ZEROFILL]  存储大小：4byte  有符号取值：-2147483648 ～ 2147483647  无符号取：0- 4294967295 |
| INTEGER | 同INT |
| BIGINT | BIGINT[(M)] [UNSIGNED] [ZEROFILL]  存储大小：8byte  有符号取值：-9223372036854775808～9223372036854775807  无符号取：0- 18446744073709551615 |

### Fixed-Point Types

<https://dev.mysql.com/doc/refman/5.7/en/precision-math.html>

* + - 1. DECIAL/NUMERIC字段值由二进制表示，并每九个数字打包成4 bytes，剩余部分按下表计算大小

| **Leftover Digits** | **Number of Bytes** |
| --- | --- |
| 0 | 0 |
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 2 |
| 5 | 3 |
| 6 | 3 |
| 7 | 4 |
| 8 | 4 |

|  |  |
| --- | --- |
| 类型 | 说明 |
| DECIMAL | DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]  1 精确的有小数的数值(fixed-point number)，可以存储指定M，D的所有值，比如DECIMAL(5,2)，可以存储-999.99 ～ 999.99  2 M：整数部分位数，不包含小数点和负号，最大值为65，默认值是10  3 D：小数点后面部分位数，取值0-30，默认值为0，当值为0时，值为整数  4 如果指定UNSIGNED，值不能为负数  5 Mysql采用二进制格式存储数值  6 如果小数点后面的数值长度大于D，会被转为D位，一般情况下是直接截断 |
| NUMERIC | NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL]  同DECIMAL，实际也是以DECIMAL实现 |
| DEC | DEC[(M[,D])] [UNSIGNED] [ZEROFILL]  同DECIMAL |
| FIXED | FIXED[(M[,D])] [UNSIGNED] [ZEROFILL]  同DECIMAIL，为了兼容其他数据库 |

### 浮点

<https://dev.mysql.com/doc/refman/5.7/en/problems-with-float.html>

|  |  |
| --- | --- |
| 类型 | 说明 |
| FLOAT | 所有FLOAT计算，Mysql都使用DOUBLE处理  大小：4byte  FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]  1 理论上取值范围，IEEE标准：  -3.402823466E+38 ～ -1.175494351E-38  0  1.175494351E-38 ～3.402823466E+38  2 实际取值范围依赖OS/硬件  3 M：不包含符号和小数点的总长度  4 D：小数点后面位数  5 如果M和D省略，默认为硬件支持的值范围  6 如果指定UNSIGNED，则不能存储负数  7 非标准Mysql扩展  8 精确到小数点后7位  FLOAT(p) [UNSIGNED] [ZEROFILL]  1 p为精度位数，只用于确定具体值处理方式，不影响实际取值范围   * 1. p值为0-23时，实际处理为无D/M的FLOAT   2. p值为24-53时，实际为无D/M的DOUBLE   2 实际取值范围，根据实际值类型决定  3 此类型是为了兼容ODBC |
| DOUBLE | DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]  大小：8byte  IEEE标准取值范围：  -1.7976931348623157E+308 ～ -2.2250738585072014E-308  0  2.2250738585072014E-308 to 1.7976931348623157E+308  实际取值范围依赖OS/硬件  如果M和D省略，默认为硬件支持的值范围  精确到小数点后15位  非标准Mysql扩展  如果指定UNSIGNED，则不能存储负数 |
| DOUBLE PRECISION | DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL]  等同于DOUBLE |
| REAL | REAL[(M,D)] [UNSIGNED] [ZEROFILL]  当REAL\_AS\_FLOAT配置启用时，等同于FLOAT，否则等同于DOUBLE |

### 其他

<https://dev.mysql.com/doc/refman/5.7/en/bit-value-literals.html>

|  |  |
| --- | --- |
| 类型 | 说明 |
| BIT | BIT[(M)]  M取值1-64，默认是1，大小为(M+7)/8 bytes  如果实际值长度小于M，会左边0补齐  在NDB cluster 中，表中的所有bit字段长度总和不能超过4096 |
| BOOLEAN | 1实际存储为TINYINT(1)；0:false，非0:true  2在Mysql中，FALSE=0，TRUE=1  2.1 SELECT IF(0 = FALSE, 'true', 'false') 》 true;  2.3 SELECT IF(1 = TRUE, 'true', 'false') 》true ;  2.2 SELECT IF(2 = TRUE, 'true', 'false') 》 false; |
| BOOL | 同BOOLEAN |

## 时间

<https://dev.mysql.com/doc/refman/5.7/en/date-and-time-types.html>

<https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html>

### 概述

* + - 1. 日期部分值默认是year-month-day的顺序，其他顺序需要通过STR\_TO\_DATE函数转换的
      2. 年用两位数字表达会自动转成四位：
         1. 年在70-99的会转为1970-1999
         2. 年在00-69的会转为2000-2069
      3. 日期时间跟数字可以根据上下文自动互相转换
      4. 如果赋给时间或日期字段一个越界值或不符合规范的值：
         1. 默认情况下会自动转为对应数据类型的ZERO Value
         2. 如果是TIME类型，则会被裁减到合适的TIME范围内的值
         3. 可以通过ALLOW\_INVALID\_DATES配置适当放宽限制

如果ALLOW\_INVALID\_DATES启用，则不执行日期完整检查，日期部分只检查month值在1-12，day值在1-31

* + - 1. Mysql允许year/month/day为0值：比如2009-00-00、2009-01-00，0000-00-00
      2. 如果不允许zero value，则启用NO\_ZERO\_DATE
      3. 日期/时间的Zero value，通过ODBC传输，会自动被转为NULL
      4. TIME/DATETIME/TIMESTAMP支持6位fractional part，可以通过${type\_name}(fsp)语法指定：
         1. ${type\_name}取值TIME/DATETIME/TIMESTAMP
         2. fsp取值0-6：如果取值0，则无fractional part，默认是0
         3. 例子：CREATE TABLE t1 (t TIME(3), dt DATETIME(6), ts TIMESTAMP(0))
      5. Mysql允许任何标点符号作为时间/日期分隔符，但毫秒部分只能用小数点
         1. 比如'10:11:12'，在DATE字段被识别为2010-11-12
         2. 比如10:45:15，在DATE字段被识别为'0000-00-00'，因为月份45是错误的

### DATE/DATATIME/TIMESTAMP

<https://dev.mysql.com/doc/refman/5.7/en/timestamp-initialization.html>

* + - 1. DATE和DATETIME的取值范围不保证今后版本中也适用(For the DATE and DATETIME range descriptions, “supported” means that although earlier values might work, there is no guarantee)
      2. DATETIME和TIMESTAMP的值的Fractional part都会被保存下来
      3. Fractional part大小

| **Fractional Seconds Precision** | **Storage Required** |
| --- | --- |
| 0 | 0 bytes |
| 1, 2 | 1 byte |
| 3, 4 | 2 bytes |
| 5, 6 | 3 bytes |

|  |  |
| --- | --- |
| 类型 | 说明 |
| DATA | Zero Value: '0000-00-00'   1. 大小：3byte 2. 取值范围'1000-01-01' to '9999-12-31' 3. 默认显示格式：YYYY-MM-DD 4. 可以赋值数字或指定格式的字符串 |
| DATETIME[(fsp)] | Zero Value: '0000-00-00 00:00:00'   1. 大小：5byte + Fractional part大小 2. 取值范围：'1000-01-01 00:00:00.000000' to '9999-12-31 23:59:59.999999' 3. fsp取值0-6：如果取值0，则无fractional part，默认是0 4. 默认显示格式YYYY-MM-DD hh:mm:ss[.fraction] 5. 可以赋值数字或指定格式的字符串 |
| TIMESTAMP[(fsp)] | Zero Value: '0000-00-00 00:00:00'   1. 大小：4byte + Fractional part大小 2. 取值范围'1970-01-01 00:00:01.000000' UTC to '2038-01-19 03:14:07.999999' UTC 3. 默认情况下存值时以服务器所在时区存，取值时以服务器所在时区取 4. 实际时区可以在每次请求时设定，这可能导致取的值跟存的值不同 5. 保存的是从1970-01-01 00:00:00开始算的秒数 6. 本身无法显示1970-01-01 00:00:00，因为此时间是0秒，会被识别为Zero value 7. fsp取值0-6：如果取值0，则无fractional part，默认是0 8. 如果sql mode为MAXDB，TIMESTAMP字段   7.1将以DATETIME实现，取值范围与DATETIME相同  7.2 显示格式同DATETIME  7.3 不能自动初始化/更新为当前日期时间  7.4 MAXDB 被标记为deprecated   1. 当配置explicit\_defaults\_for\_timestamp=enabled时   8.1 语法DEFAULT CURRENT\_TIMESTAMP或 ON UPDATE CURRENT\_TIMESTAMP无法自动赋值，必须显示赋值TIMESTAMP字段  8.2 如果没显示指定NOT NULL的TIMESTAMP字段，允许NULL值   1. 当配置explicit\_defaults\_for\_timestamp=disabled时   9.1 表中第一个TIMESTAMP字段默认附带(DEFAULT CURRENT\_TIMESTAMP and ON UPDATE CURRENT\_TIMESTAMP),即在无显示赋值时，被自动赋值当前日期时间  9.2 表中任何一个TIMESTAMP字段如果没指定NULL，传值NULL时会自动赋值为当前日期时间 |

### TIME/YEAR

|  |  |
| --- | --- |
| 类型 | 说明 |
| TIME[(fsp)] | Zero Value: '00:00:00'   1. 大小：3byte + Fractional part大小 2. 取值范围：'-838:59:59.000000' to '838:59:59.000000' 3. 显示格式：hh:mm:ss[.fraction],既可以用于表示一个时间点，也可以用于表示运行时间或两个事件的时间间隔 4. 可以赋值数字或指定格式的字符串 5. fsp取值0-6：如果取值0，则无fractional part，默认是0 6. 赋值时注意：    1. 11:12 表示 11:12:00    2. 1122 表示 00:11:12 7. 如果赋了异常的值    1. 如果值仅仅超过了上下限，则会转为上下限值，比如'-850:00:00'转为 '-838:59:59'，'850:00:00' 转为'838:59:59'    2. 如果值非法，则转成Zero value 8. 如果TIME字段存储的值为'00:00:00'，无法判断是存的Zero value，还是因为异常转换成'00:00:00' |
| YEAR | Zero Value:0000   1. 大小：1byte 2. 取值范围 0000，1901 to 2155 3. 显示格式为四位，YYYY 4. 赋值时：   3.1如果值是'0' - '69'的数字字符串，mysql会转为2000-2069  3.2如果值是'70' - '99'的数字字符串，mysql会转为1970-1999  3.3如果值是1-69的数字，转为2001-2069  3.4如果值是70-99的数字，转为1970-1999  3.5如果值是0的数字，则转为Zero value   1. strict SQL mode：   4.1如果开启，非法值会报错  4.2如果关闭，非法值会转为Zero value   1. YEAR(2)在5.7已移除 |

## 字符

<https://dev.mysql.com/doc/refman/5.7/en/string-types.html>

<https://dev.mysql.com/doc/refman/5.7/en/string-functions.html>

<https://dev.mysql.com/doc/refman/5.7/en/charset-column.html>

<https://dev.mysql.com/doc/refman/5.7/en/silent-column-changes.html>

<https://dev.mysql.com/doc/refman/5.7/en/charset-national.html>

<https://dev.mysql.com/doc/refman/5.7/en/charset.html>

### 概述

* + - 1. 对于CHAR/VARCHAR/TEXT等字符串字段，长度单位是字符
      2. 对于BINARY/VARBINARY/BLOB等二进制字符串字段，长度单位是bytes
      3. CHAR/VARCHAR/TEXT/ENUM/SET等字段，可以通过CHARACTER SET/CHARACTER单独指定字符集，通过COLLATE指定排序规则，比如

1. **CREATE** **TABLE** t
2. (
3. c1 **VARCHAR**(20) **CHARACTER** **SET** utf8,
4. c2 TEXT **CHARACTER** **SET** latin1 **COLLATE** latin1\_general\_cs
5. );
   * + - 1. 如果通过[CHARACTER SET|CHARACTER] binary设置字段，则：

CHAR被转为BINARY

VARCHAR 被转为 VARBINARY

TEXT转为BLOB

ENUM/SET则无转化

* + - 1. 对于CHAR,VARCHAR,TEXT,ENUM,SET数据类型，可以声明binary(\_bin)collation或 BINARY attribute，从而使用底层字符码值进行排序和比较
      2. CHAR/VARCHAR/TEXT 字段搜索都忽略尾部空格(除了LIKE查询)

1. **CREATE** **TABLE** names (myname **CHAR**(10));
3. **INSERT** **INTO** names **VALUES** ('Jones');
5. **SELECT** myname = 'Jones', myname = 'Jones  ' **FROM** names;
6. +------------------+--------------------+
7. | myname = 'Jones' | myname = 'Jones  ' |
8. +------------------+--------------------+
9. |                1 |                  1 |
10. +------------------+--------------------+
12. mysql> **SELECT** myname LIKE 'Jones', myname LIKE 'Jones  ' **FROM** names;
13. +---------------------+-----------------------+
14. | myname LIKE 'Jones' | myname LIKE 'Jones  ' |
15. +---------------------+-----------------------+
16. |                   1 |                     0 |
17. +---------------------+-----------------------+
    * + 1. 由e的特性影响，有唯一约束的字段，如果插入两个仅末尾空格差异的值，也会造成主键冲突
           1. 比如插入’a’和’a ’,会报主键冲突

### CHAR & VARCHAR

<https://dev.mysql.com/doc/refman/5.7/en/charset-national.html>

<https://dev.mysql.com/doc/refman/5.7/en/column-count-limit.html>

|  |  |
| --- | --- |
| 类型 | 说明 |
| CHAR | [NATIONAL] CHAR[(M)] [CHARACTER SET charset\_name] [COLLATE collation\_name]   1. M取值范围，0-255个字符，默认 1，完整名称：CHARACTER 2. 固定长度字符串，存储时不足部分右边空格补齐，默认情况下    1. 读取时会移除尾部空哥    2. 如果PAD\_CHAR\_TO\_FULL\_LENGTH参数启用了，读取时则不移除尾部空格 3. 默认字符集是utf8 4. NATIONAL CHAR是SQL标准定义CHAR字段方式，缩写是NCHAR 5. CHAR(0)占1bit，可以赋值NULL或’’(空字符串) 6. 如果strict SQL mode禁用，则插入的值长度超过M，则会被截断 7. 任何SQL mode下，超过M的空格都会被静默截断 |
| VARCHAR | [NATIONAL] VARCHAR(M) [CHARACTER SET charset\_name] [COLLATE collation\_name]   1. M取值0-65535，单位byte，完整名称：CHARACTER VARYING 2. 字段实际可存储字符数，依赖M值和采用的字符集 3. 每个字段前缀有：    1. 如果数据大小小于等于255，前缀有1 byte    2. 如果大于255，则前缀有2 byte 4. NATIONAL VARCHAR是SQL标准定义VARCHAR字段的方式，缩写为NVARCHAR 5. 默认字符集是utf8 6. 如果strict SQL mode禁用，则插入的值长度超过M，则会被截断 7. 任何SQL mode下，超过M的空格都会被截断，并产生一个warning |

* + - 1. CHAR&VARCHAR差异，字符集均为latin1，非strict SQL mode

| **Value** | CHAR(4) | **Storage Required** | VARCHAR(4) | **Storage Required** |
| --- | --- | --- | --- | --- |
| '' | '    ' | 4 bytes | '' | 1 byte |
| 'ab' | 'ab  ' | 4 bytes | 'ab' | 3 bytes |
| 'abcd' | 'abcd' | 4 bytes | 'abcd' | 5 bytes |
| 'abcdefgh' | 'abcd' | 4 bytes | 'abcd' | 5 bytes |

1. **CREATE** **TABLE** vc (v **VARCHAR**(4), c **CHAR**(4))
3. **INSERT** **INTO** vc **VALUES** ('ab  ', 'ab  ');
5. **SELECT** CONCAT('(', v, ')'), CONCAT('(', c, ')') **FROM** vc;
6. +---------------------+---------------------+
7. | CONCAT('(', v, ')') | CONCAT('(', c, ')') |
8. +---------------------+---------------------+
9. | (ab  )              | (ab)                |
10. +---------------------+---------------------+

### BINARY & VARBINARY

|  |  |
| --- | --- |
| 类型 | 说明 |
| BINARY | BINARY[(M)]   1. 跟CHAR类似，不过存储的是二进制字节字符串数据 2. 如果实际长度小于M，则用zero byte(0x00)填充右边 3. 读取时，不会被移除 |
| VARBINARY(N) | VARBINARY(M)   1. 跟VARCHAR相似，不过存储的是二进制字节字符串数据 |

* + - 1. 排序对比基于字节对应的数字值
      2. 长度单位都是byte
      3. 如果strict SQL mode禁用，则插入的值长度超过M，则会被截断，并产生warnging
      4. 当排序或DISTINCT操作时，0x00和空格时不同值，且0x00排在空格前
      5. 当用BINARY字段时需要注意下面的情况

1. **CREATE** **TABLE** t (c **BINARY**(3));
3. **INSERT** **INTO** t **SET** c = 'a';
5. **SELECT** HEX(c), c = 'a', c = 'a\0\0' **from** t;
6. +--------+---------+-------------+
7. | HEX(c) | c = 'a' | c = 'a\0\0' |
8. +--------+---------+-------------+
9. | 610000 |       0 |           1 |
10. +--------+---------+-------------+

### BLOB & TEXT

<https://dev.mysql.com/doc/refman/5.7/en/storage-requirements.html>

<https://dev.mysql.com/doc/refman/5.7/en/blob.html>

<https://dev.mysql.com/doc/refman/5.7/en/column-indexes.html>

<https://dev.mysql.com/doc/refman/5.7/en/internal-temporary-tables.html>

* + - 1. BLOB存储二进制字符串，基于binary字符集和排序规则，排序和比较基于byte对应的数值
      2. TEXT存储字符字符串，排序和比较基于实际采用字符集和排序规则
      3. 如果strict SQL mode禁用，则插入的值长度超过M，则会被截断，并产生warnging
      4. 任何SQL mode下，插入TEXT字段超过M的空格都会被截断，并产生一个warning
      5. 查询结果，字段值都不会做填充或截取
      6. 对于TEXT字段，如果有唯一索引，插入两个区别仅为尾部空格的值会导致逐渐冲突，具体参考3.1
      7. BLOB和TEXT不能有DEFAULT值，建立索引时必须指定索引长度如下

CREATE TABLE test (blob\_col BLOB, INDEX(blob\_col(10)));

* + - 1. 为了兼容，LONG和LONG VARCHAR 映射为MEDIUMTEXT
      2. ODBC将BLOB定义为LONGVARBINARY，将TEXT定义为LONGVARCHAR
      3. 注意：
         1. 字段值默认只有首1024byte参与排序，可通过max\_sort\_length配置
         2. 如果用临时表处理查询结果中有BLOB/TEXT字段，会因为MEMORY存储引擎的数据结构不支持此类字段，导致实际是在disk上执行
         3. BLOB/TEXT最大支持大小依赖于其具体类型，但C/S交互依赖于实际可用memory和buffer大小，以及通信协议
      4. BLOB/TEXT内部实现是单独分配的对象，每次打开表，都会单独分配一个对象

|  |  |
| --- | --- |
| 类型 | 说明 |
| TINYBLOB | 1. 最大为255byte的BLOB 2. 每个字段都有前缀1byte表示长度 |
| BLOB | BLOB[(M)]   1. 最大为65535(216 − 1)byte 2. 每个字段都有前缀2byte表示长度 3. 如果指定M（单位byte），则会创建最小的能容纳M大小的BLOB字段 |
| MEDIUMBLOB | 1. 最大为16777215 (224 − 1) 2. 每个字段都有前缀3byte表示长度 |
| LONGBLOB | 1. 最大为4294967295(232 − 1)byte 或者4GB,实际大小依赖C/S通信协议限制和可用存储空间 2. 每个字段都有前缀4byte表示长度 |
| TINYTEXT | TINYTEXT [CHARACTER SET charset\_name] [COLLATE collation\_name]   1. 最大支持255(28 − 1)byte的字符长度，实际可存储字符数量依赖字符集 2. 每个字段都有前缀1byte表示长度 |
| TEXT | TEXT[(M)] [CHARACTER SET charset\_name] [COLLATE collation\_name]   1. 最大为65535(216 − 1)byte，实际可存储字符数量依赖字符集 2. 每个字段都有前缀2byte表示长度 3. 如果指定M（单位byte），则会创建最小的能容纳M大小的TEXT字段 |
| MEDIUMTEXT | MEDIUMTEXT [CHARACTER SET charset\_name] [COLLATE collation\_name]   1. 最大为16777215 (224 − 1) ，实际可存储字符数量依赖字符集 2. 每个字段都有前缀3byte表示长度 |
| LONGTEXT | LONGTEXT [CHARACTER SET charset\_name] [COLLATE collation\_name]   1. 最大为4294967295(232 − 1)byte 或者4GB,实际大小依赖C/S通信协议限制和可用存储空间，实际可存储字符数量依赖字符集 2. 每个字段值都有前缀4 byte表示长度 |

### ENUM & SET

<https://dev.mysql.com/doc/refman/5.7/en/create-table-files.html#limits-frm-file>

https://dev.mysql.com/doc/refman/5.7/en/string-type-syntax.html

* + - 1. 一张表最多255个ENUM/SET字段
      2. 定义ENUM/SET字段时，元素尾部空格会被自动删除

### 

| **ENUM 第5，10点样例，ENUM('Mercury', 'Venus', 'Earth')** | |
| --- | --- |
| **Value** | **Index** |
| NULL | NULL |
| '' | 0 |
| 'Mercury' | 1 |
| 'Venus' | 2 |
| 'Earth' | 3 |

1. --ENUM 第8点样例
2. numbers ENUM('0','1','2');
3. **INSERT** **INTO** t (numbers) **VALUES**(2),('2'),('3');
4. **SELECT** \* **FROM** t;
5. +---------+
6. | numbers |
7. +---------+
8. | 1       |
9. | 2       |
10. | 2       |
11. +---------+

| **SET第4点样例，SET('a','b','c','d')** | | |
| --- | --- | --- |
| SET**Member** | **Decimal Value** | **Binary Value** |
| 'a' | 1 | 0001 |
| 'b' | 2 | 0010 |
| 'c' | 4 | 0100 |
| 'd' | 8 | 1000 |
| 'a,d' | 9 | 1001 |

|  |  |
| --- | --- |
| 类型 | 说明 |
| ENUM | ENUM('value1','value2',...) [CHARACTER SET charset\_name] [COLLATE collation\_name]   1. 值只能是定义的集合里的元素、NULL、空字符串 2. 底层存储为integer，定义时指定的字符串元素会被自动编码为数字，读取时数字会转为对应的字符串 3. 最大65535个distinct elements，实际小于3000个， 4. 元素必须是带引号的字符串，元素索引下标从1开始 5. 在strict SQL mode禁用时，插入非法值会用空字符串代替，下标为0，因此可以用SELECT \* FROM tbl\_name WHERE enum\_col=0;找出所有非法记录 6. 可以通过SELECT enum\_col+0 FROM tbl\_name;方式获取ENUM元素下标 7. 数值计算的Function处理ENUM值时，使用元素下标进行计算 8. 如果存储一个数字到ENUM，样例参考    1. 如果是number，则以元素下标处理，实际存储的是ENUM对应下标的元素    2. 如果是数字字符串，如果有匹配ENUM元素，则存储为对应元素    3. 如果是数字字符串，且没有匹配ENUM的元素，则以元素下标处理 9. ENUM默认值：    1. 如果没声明NOT NULL约束，则默认值为NULL    2. 如果声明了NOT NULL约束，则默认值为第一个元素 10. 排序：     1. 默认情况下，按照元素下标排序，NULL值在所有元素之前     2. 可以指定按字母排序 |
| SET | SET('value1','value2',...) [CHARACTER SET charset\_name] [COLLATE collation\_name]   1. 值只能是定义的集合里的元素 2. 底层存储为integer 3. 最多64个distinct members 4. 如果将一个数字存入set字段，则结果值按数字的二进制表示的1值位置决定具体存储值 5. 排序按照建表时元素顺序决定，NULL值在所有元素之前 6. 如果插入值中有非法字符：    1. 如果strict SQL mode启用，则直接报错    2. 如果strict SQL mode禁用，则忽略非法字符，将有效字符插入，并生成警告 |

## JSON

## Spatial Data Type