数据类型

说在前面的话： 数据类型的定义在工作中与实习生没多大关系，定义数据库要求很严格，先择的类型也很苛刻根据公司具体项目来决定的（有专门的职位，产品经理与数据库管理员。），但是还是要了解基本的用途，毕竟你永远不会只能是实习生。

整型

类型	大小	范围（有符号）	范围（无符号）	用途
TINYINT	1 Bytes	(-128，127)	(0，255)	小整数值
SMALLINT	2 Bytes	(-32 768，32 767)	(0，65 535)	大整数值
MEDIUMINT	3 Bytes	(-8 388 608，8 388 607)	(0，16 777 215)	大整数值
INT或INTEGER	4 Bytes	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	大整数值
BIGINT	8 Bytes	(-9,223,372,036,854,775,808，9 223 372 036 854 775 807)	(0，18 446 744 073 709 551 615)	极大整数值

unsigned

关键字unsigned的属性就是将数字类型无符号化（当定义的字段不可能为负数时可使用）

例如：int的类型范围事-2 147 483 648 ~ 2 147 483 647，int unsigned 的类型范围就是0 ~ 4 294 967 295。

浮点型

类型	大小	范围（有符号）	范围（无符号）	用途
FLOAT	4 Bytes	(-3.402 823 466 E+38，-1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38)	0，(1.175 494 351 E-38，3.402 823 466 E+38)	单精度浮点数值
DOUBLE	8 Bytes	(-1.797 693 134 862 315 7 E+308，-2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	双精度浮点数值
DECIMAL	对DECIMAL(M,D)，如果M>D，为M+2否则为D+2	依赖于M和D的值	依赖于M和D的值	小数值

关于浮点类型的代码

在需要编辑浮点类型字段类型后面的宽度设定里面要加逗号 (M,D) 逗号前代表整体数字的宽度，逗号后代表小数部分的宽度。如果输入小数部分超出两位将会四舍五入保留两位。**备注：设定宽度时要M>D否则会报错。**

例：

```
-------------------------------------------------------------------------------
| Table  | Create Table

+--------+-------------------------------------------------------------------------
-------------------------------------------------------------------------------
| 浮点型 | CREATE TABLE `浮点型` (
  `number_one` float(3,1) DEFAULT NULL COMMENT '单精度设置字段宽度整数为3小数为1',
  `number_two` double(5,2) DEFAULT NULL COMMENT '双精度设置字段宽度整数为5小数为2'
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+--------+-------------------------------------------------------------------------
-------------------------------------------------------------------------------
1 row in set (0.00 sec)

mysql> insert into `浮点型` values
    -> (2.1,2.23),(1.0,2.77),(1.0,2.7777777),(2.9,2.7),(2.99,2.7);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from `浮点型`;
+------------+------------+
| number_one | number_two |
+------------+------------+
|        2.1 |       2.23 |
|        1.0 |       2.77 |
|        1.0 |       2.78 |
|        2.9 |       2.70 |
|        3.0 |       2.70 |
+------------+------------+
5 rows in set (0.00 sec)
```

总结：

1、浮点数存在误差问题。
2、对货币等对精度敏感的数据，应该用定点数表示或存储。
3、编程中，如果用到浮点数，要特别注意误差问题，并尽量避免做浮点数比较。
4、要注意浮点数中一些特殊值的处理。

定点数(DECIMAL)

不会丢失精度,支持无符号，整数与小数分开储存保证精度的同时也加大储存量。

例：

```
mysql> show create table `浮点型`;
+--------+-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
| Table  | Create Table
                                                                                |
+--------+-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
| 浮点型 | CREATE TABLE `浮点型` (
  `number_three` decimal(20,19) DEFAULT NULL,
  `number_four` float(20,19) DEFAULT NULL,
  `number_five` double(20,19) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+--------+-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> insert into `浮点型` values
    -> (8.1111111111111111111, 8.1111111111111111111, 8.1111111111111111111);
Query OK, 1 row affected (0.00 sec)

mysql> select * from `浮点型`;
+----------------------+----------------------+----------------------+
| number_three         | number_four          | number_five          |
+----------------------+----------------------+----------------------+
| 8.1111111111111111111 | 8.1111106872558600000 | 8.1111111111111100000 |
+----------------------+----------------------+----------------------+
1 row in set (0.00 sec)
```

字符串与文本类型

类型	大小	用途
CHAR	0-255 bytes	定长字符串
VARCHAR	0-65535 bytes	变长字符串
TINYBLOB	0-255 bytes	不超过 255 个字符的二进制字符串
TINYTEXT	0-255 bytes	短文本字符串
BLOB	0-65 535 bytes	二进制形式的长文本数据
TEXT	0-65 535 bytes	长文本数据
MEDIUMBLOB	0-16 777 215 bytes	二进制形式的中等长度文本数据
MEDIUMTEXT	0-16 777 215 bytes	中等长度文本数据
LONGBLOB	0-4 294 967 295 bytes	二进制形式的极大文本数据
LONGTEXT	0-4 294 967 295 bytes	极大文本数据

特点

关于各类的字节占用空间：

根据字符集编码的种类不同，无论是英文还是汉字的字符占用的空间在不同字符编码下，占用的空间是不同的。一个字符占用的空间，在那时不一定是1byte。所以说各类型的最大字节有多大就能存下与其相同数量的字符是不成立的。

比如说 varchar 的理论字符储存空间大小为 0-65535 个字节，但是在中文字符集编码 GBK 下一个字符要占用2个字节，因此 varchar 实际储存能储存的字符只有理论的一半。

总结：在设定字符串类型时，要注意自己的字符集编码一个字符要用多少个字节，以防数据溢出被截取。

另外在实际企业中我们绝大多使用 UTF8

对字符集编码感兴趣可跳转下面链接

UTF-8 不是固定字长编码的, 而是一种变长的编码方式. 它可以使用 1~4 个字节表示一个符号, 根据不同的符号而变化字节长度. 这是种比较巧妙的设计, 如果一个字节的第一位是 0, 则这个字节单独就是一个字符; 如果第一位是 1, 则连续有多少个 1, 就表示当前字符占用多少个字节.

————————————————————

原文链接：https://blog.csdn.net/weixin_42558093/article/details/113555181

定长字符串和变长字符串的区别：

定长字符串在储存时，设置多大空间就储存多大空间，不管你的数据占用多大空间，如果空间超出则会被截取。

变长字符串在储存时，所占资源空间是你存储内容的长度。多余的空间会自动回收。

总结：varchar相较于char，更能节省资源空间。

但是，还在有些情况下，数据库对定长字段操作有优化，数据量量大时做特定操作时，会比变长字符串快很多，效率高。

布尔类型

在存储数据时，经常会用到“是”、“否”；“有”、“无”这种数据。

注意：MySQL 不存在 Boolean，Boolean 在 MySQL 里的类型为 tinyint(1)

但这不代表你不能这么写（唉，其他语言写些习惯了），MySQL会自动帮你把他转化。

例：

```
mysql> create table `boolean`(
    -> gril boolean comment'是女孩么'
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql> select * from `boolean`;
Empty set (0.00 sec)

mysql> insert into `boolean` values(true);
Query OK, 1 row affected (0.00 sec)

mysql> select * from `boolean`;
+------+
| gril |
+------+
|    1 |
+------+
1 row in set (0.00 sec)
```

枚举类型（ENUM）

介绍

首先，要说的是「 ENUM 类型其实是一个字符串类型 」，虽然作用很像 C 或 C++ 中的枚举类型。

其次，要说明的是，ENUM 类型的值都是从 **允许值列表中选择**(在ENUM后面所跟的值详见，例)。

再次，**允许值** 列表在创建表结构时就定义好的，表创建完成后可以使用 alter 语句来修改允许值列表。

例：

```
mysql> use `数据类型`;
Database changed
mysql> create table `enum`(
    -> skin_color enum ('yellow', 'white', 'black')
    -> );
Query OK, 0 rows affected (0.11 sec)
```

```
mysql> desc `enum`;
+------------+-------------------------------+------+-----+---------+-------+
| Field      | Type                          | Null | Key | Default | Extra |
+------------+-------------------------------+------+-----+---------+-------+
| skin_color | enum('yellow','white','black') | YES  |     | NULL    |       |
+------------+-------------------------------+------+-----+---------+-------+
1 row in set (0.02 sec)
```

```
mysql> insert into enum values ('white');
Query OK, 1 row affected (0.01 sec)

mysql> insert into enum values (1);
Query OK, 1 row affected (0.00 sec)

mysql> select * from `enum`;
+------------+
| skin_color |
+------------+
| white      |
| yellow     |
+------------+
2 rows in set (0.00 sec)

mysql> insert into enum values ('green');
ERROR 1265 (01000): Data truncated for column 'skin_color' at row 1
```

每个枚举值均有一个索引值,什么是索引值？在列说明中列表**允许值** 被从 1 开始编号。看不懂？结合上面例图看如下表:

值	索引值
NULL	NULL
''	0
'yellow'	1
'white'	2
'black'	3

在下列某些情况下，值也可以是空串('') 或 NULL

总结

优点

1. 数据更紧凑。因为 ENUM 列一般都是有限的值，一般不多余 5 个这样，这就比保存 `true` 或 `false` 节省空间多了。因为 MySQL 会在创建或者修改表结构时将 ENUM 允许的值自动编码为数字，而这个数字一般的分配空间为 1 ~ 2 字节 (byte) ，具体取决于实现。

   例如，将值为 `medium` 的100万行插入表将需要 100 万字节的存储空间，而如果将实际字符串 `medium` 存储在 VARCHAR 列中则需要 600 万字节。

2. 更好的可读性，虽然在存储的是数字，但在输入和输出时使用的都是对应的允许值。

缺点

1.千万不要使用数字作为枚举值，因为这样容易混淆它们的字面值和内部索引值。

2.在 ORDER BY 语句中使用 ENUM 更要注意

3.创建和使用 ENUM 数据类型的一些问题

4.枚举值字面量和内部索引的问题

5.处理枚举值字面量的一些问题

6.ENUM 类型中的 NULL 或空值问题

7.ENUM 类型的排序问题

8.ENUM 类型的一些限制

享受这些好处的同时，也要承担 ENUM 所带来的各种负面影响，感兴趣详见下面链接

https://www.twle.cn/c/yufei/mysqlfav/mysqlfav-basic-enum2.html

---

SET集合类型

介绍

形式与ENUM类型一样。SET类型的值可以取列表中的一个元素或者多个元素的组合。取多个元素时，不同元素之间用逗号隔开。SET类型的值最多只能是有64个元素构成的组合，根据成员的不同，存储上也有所不同：1～8成员的集合，占1个字节。9～16成员的集合，占2个字节。17～24成员的集合，占3个字节。25～32成员的集合，占4个字节。33～64成员的集合，占8个字节。

例：

```
mysql> create table `set`(
    -> subject set('math','chinese','english','chemistry','physics') comment '学科'
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql> desc `set`;
+---------+-----------------------------------------------------+------+-----+---------+-------+
| Field   | Type                                                | Null | Key | Default | Extra |
+---------+-----------------------------------------------------+------+-----+---------+-------+
| subject | set('math','chinese','english','chemistry','physics') | YES  |     | NULL    |       |
+---------+-----------------------------------------------------+------+-----+---------+-------+
1 row in set (0.00 sec)
```

```
mysql> insert into `set` values ('math,chinese,english,chemistry');
Query OK, 1 row affected (0.00 sec)

mysql> insert into `set` values ('1');
Query OK, 1 row affected (0.00 sec)

mysql> insert into `set` values ('2');
Query OK, 1 row affected (0.00 sec)

mysql> insert into `set` values ('4');
Query OK, 1 row affected (0.00 sec)

mysql> insert into `set` values ('8');
Query OK, 1 row affected (0.00 sec)

mysql> select * from `set`;
+--------------------------------+
| subject                        |
+--------------------------------+
| math,chinese,english,chemistry |
| math                           |
| chinese                        |
| english                        |
| chemistry                      |
+--------------------------------+
5 rows in set (0.00 sec)
```

每个set值均有一个索引值,但是与枚举不同，在列说明中列表**允许值** 被从 1 开始编号以2的幂递增。看不懂？结合上面例图看如下表:

值	索引值
NULL	NULL
''	0
'math'	1
'chinese'	2
'english'	4
‘chemistry’	8
'physics'	16

时间日期类型

行业中规定每一张表都应当有时间类型

类型	大小 (bytes)	范围	格式	用途
DATE	3	1000-01-01/9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'/'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901/2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00/9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	4	1970-01-01 00:00:00/2038 结束时间是第 **2147483647** 秒，北京时间 **2038-1-19 11:14:07**，格林尼治时间 2038年1月19日 凌晨 03:14:07	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

在实际使用中多数使用 date time 类型。

例：

```
mysql> insert into enum values ('green');
ERROR 1265 (01000): Data truncated for column 'skin_color' at row 1
mysql> create table `时间类型`(
    -> time datetime comment'时间'
    -> );
Query OK, 0 rows affected (0.06 sec)

mysql> desc `时间类型`;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| time  | datetime | YES  |     | NULL    |       |
+-------+----------+------+-----+---------+-------+
1 row in set (0.01 sec)

mysql> insert into `时间类型` values ('1974-04-05 12:00:00');
Query OK, 1 row affected (0.00 sec)

mysql> select * from `时间类型`;
+---------------------+
| time                |
+---------------------+
| 1974-04-05 12:00:00 |
+---------------------+
1 row in set (0.00 sec)
```