

本节所讲内容：

- MySQL 数据类型
- 数值类型
- 字符串类型
- 日期和时间类型
- 复合类型

39.1 MySQL 数据类型

面试常遇到的问题：

mysql 数据库中的表 金额栏位 用 int 类型。计算会不会引起偏差？ 金额一般是小数点 2 位。

如果用 **DECIMAL** 会不会好点？

DECIMAL ['desɪm(ə)l]

答：具体情况还要分析下。

金额在普通情况下用浮点数即可,但是由于 mysql 是 c 语言写的， 浮点类型也是 c 语言的。。

在精密计算的时候，也有 c 语言浮点类型精度问题。所以在数据量大的计算过后可能会出现这样的结

果

41718355.0000001.

如果要求精度高一些的计算，还是用 DECIMAL 吧，虽然效率会慢一些。

为了看懂这个问题我们就需要了解数据类型：

MySQL 数据类型

数据类型是数据的一种属性，其可以决定数据的存储格式，有效范围和相应的限制。mysql 的数据类型包括整数类型，浮点数类型，定点数类型，日期和时间类型，字符串类型和二进制类型。

为什么定义数据类型?为什么要数据分类?

答：

(1) 使系统能够根据数据类型来操作数据。

(2) 预防数据运算时出错。

例：通过强大的数据分类把每个类型与特定的行为联系在一起，执行这些行为时，数据分类可以预防错误。最长见的错误是字符与数字相加。

(3) 更有效的利用空间。数据分类，可以使用最少的存储，来存放数据，同时提高了性能。

例：如把 12345678901234567 识别为一个 (8 字节) 数字，而不是一个 17 字节的字符串，1 个字符占用一个字节。

拓展：

一个汉字占多少长度与编码有关:

UTF - 8：一个汉字 = 3 个字节

GBK：一个汉字 = 2 个字节

有 PHP 的基础测试下这个脚本

先来测试一下 php 把一个汉字认作几个字节:

测试下：

```
<?php
header('Content-Type:text/html;charset=UTF-8');

$str='我';

echo strlen($str);

?>
```

输出 3，证明在 UTF-8 编码下，一个汉字被认作 3 个字节长度.

39.2 MySQL 数据类型具体分类

39.2.1 数据类型解释

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。

单词汇总：

tiny 美 ['taɪni]

MEDIUM 美 ['miðiəm]

FLOAT 美 [floʊt]

表 39.1 数值类型

类 型	大 小	范围（有符号）	范围（无符号）	用 途
TINYINT	1 字节	(-128, 127) 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）	(0, 255) 2^8-1	小整数值

SMALLINT	2 字节	(-32 768 , 32 767)	(0 , 65 535) $2^{16}-1$	大整数值
MEDIUMINT	3 字节	(-8 388 608 , 8 388 607)	(0 , 16 777 215)	大整数值

续表

类 型	大 小	范围 (有符号)	范围 (无符号)	用 途
INT 或 INTEGER	4 字节	(-2 147 483 648 , 2 147 483 647)	(0 , 4 294 967 295) $2^{32}-1$	大整数值
BIGINT	8 字节	(-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
FLOAT	4 字节	(-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 字节	(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

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是相同的，只有他们存储的值的大小是不同的。

使用举例：

如果用来保存用户的年龄（举例来说，数据库中保存年龄是不可取的），用 TINYINT 就够了；
 九城的《纵横》里，各项技能值，用 SMALLINT 也够了；
 如果要用作一个肯定不会超过 16000000 行的表的 AUTO_INCREMENT 的 IDENTIFY 字段，当然用 MEDIUMINT 不用 INT，试想，每行节约一个字节，16000000 行可以节约 10 兆多

39.2.2 数据类型的测试

```
mysql> create database jeff;
Query OK, 1 row affected (0.00 sec)
mysql> use jeff;
Database changed
```

【例 39.1】测试取值范围

```
mysql> create table kdata ( fti tinyint,fsi smallint,fmi mediumint,fi int, fbi bigint);
mysql> insert into kdata values (123456789,123456789,123456789,123456789,123456789);
```

```
mysql> insert into kdata values (123456789,123456789,123456789,123456789,
123456789); mysql> select * from kdata; #前几个字段只存放了最大值
+-----+-----+-----+-----+
| fti | fsi | fmi | fi | fbi |
+-----+-----+-----+-----+
| 127 | 32767 | 8388607 | 123456789 | 123456789 |
| 127 | 32767 | 8388607 | 123456789 | 123456789 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

【例 39.2】插入不合规定的值 ,插入结果 为 0

```
mysql> create table kdata2 (age int);
mysql> insert into kdata2 values ('hello');
mysql> select * from kdata2;
+-----+
| age |
+-----+
| 0 |
+-----+
1 row in set (0.00 sec)
```

为什么为 0

保存的数据不存在为 null

而 null 又不是 int 类型，所以只能是 0

39.3 int 类型修饰符

单词汇总：

unsigned [ʌn'saɪnd]

zerofill 美 [zɜrə'fɪl]

unsigned 无符号整数，修饰符：规定字段只能保存正的数据。它可以增大这个字段的正数支持的范围。**zerofill** 修饰符：规定 0（不是空格）填补输出的值。使用这个值可以防止 mysql 存储负值。

【例 39.3】unsigned 和 zerofill 使用方法

```
mysql> create table kdata3 (fi int,fiu int unsigned, fiz int zerofill,fiuz int unsigned zerofill);
```

查看表结构：

```
mysql> desc kdata3;
```

```
mysql> desc kdata3;
+-----+-----+-----+-----+-----+-----+
| Field | Type                               | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| fi    | int(11)                           | YES  |     | NULL    |       |
| fiu   | int(10) unsigned                   | YES  |     | NULL    |       |
| fiz   | int(10) unsigned zerofill          | YES  |     | NULL    |       |
| fiuz  | int(10) unsigned zerofill          | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.02 sec)
```

图 39.1 unsigned 和 zerofill 使用方法

注意：发现 fiz 和 fiuz 字段值是一样的。

查看原因：

```
mysql> show create table kdata3;
| kdata31 | CREATE TABLE `kdata3` (
  `fi` int(11) DEFAULT NULL,
  `fiu` int(10) unsigned DEFAULT NULL,
  `fiz` int(10) unsigned zerofill DEFAULT NULL,
  `fiuz` int(10) unsigned zerofill DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1 |
```

测试：

```
mysql> insert into kdata3 values (10,10,10,10);
Query OK, 1 row affected (0.00 sec)
mysql> insert into kdata3 values (-10,-10,-10,-10);
Query OK, 1 row affected, 3 warnings (0.00 sec)
mysql> select * from kdata3;
+-----+-----+-----+-----+
| fi  | fiu | fiz      | fiuz      |
+-----+-----+-----+-----+
| 10 | 10 | 0000000010 | 0000000010 |
| -10 | 0 | 0000000000 | 0000000000 |
+-----+-----+-----+-----+
mysql> desc kdata3;
+-----+-----+-----+-----+-----+-----+
| Field | Type                               | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| fi    | int(11)                           | YES  |     | NULL    |       |
| fiu   | int(10) unsigned                   | YES  |     | NULL    |       |
| fiz   | int(10) unsigned zerofill          | YES  |     | NULL    |       |
| fiuz  | int(10) unsigned zerofill          | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
```

注意：int(M) 在 integer 数据类型中，M 表示最大显示宽度。

在 int(M) 中，M 的值跟 int(M) 所占多少存储空间并无任何关系。int(3)、int(4)、

int(8) 在磁盘上都是占用 4 bytes 的存储空间。其实，除了显示给用户的方式有点不同外，

int(M) 跟 int 数据类型是相同的。

如果 int 的值为 10

int (10) 显示结果为 0000000010

int (3) 显示结果为 010

就是显示的长度不一样而已 都是占用四个字节的空間，可以使用的空間也一样。

注：当我们生成固定长度的序列号时，可以使用 zerofill。 如：卡号

370 【例 39.4】默认使用空格填充，不方便显示出来。现在以 0 来填充，查看一下显示的内容。

```
mysql> create table azerofill (fi int(3), fiz int(3) zerofill, fiuz int(4) unsigned zerofill);
mysql> insert into azerofill values(11,11,11);
mysql> select * from azerofill;
```

测试，插入最于最大显示范围的值。

```
mysql> insert into azerofill values(123456,123456,123456);
mysql> select * from azerofill;
```

```
mysql> select * from azerofill;
+-----+-----+-----+
| fi   | fiz  | fiuz |
+-----+-----+-----+
| 11   | 011  | 0011 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

图 39.2 默认使用空格填充

```
mysql> select * from azerofill;
+-----+-----+-----+
| fi   | fiz  | fiuz |
+-----+-----+-----+
| 11   | 011  | 0011 |
| 123456 | 123456 | 123456 |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

图 39.3 插入最于最大显示范围的

39.4 浮点型数据类型

float(3,1) ：表示此字段有效位数为 3 位，小数点后面 1 位数字。

【例 39.5】小数点后超过 1 位，mysql 自动 给四会五入。

```
mysql> create table kdata6 (test float(3,1));
Query OK, 0 rows affected (0.03 sec)
mysql> insert into kdata6 values (123.5);
Query OK, 1 row affected, 1 warning (0.00 sec)
mysql> insert into kdata6 values (3.455);
Query OK, 1 row affected (0.00 sec)
mysql> insert into kdata6 values (123.455);
```

Query OK, 1 row affected, 1 warning (0.00 sec)

```
mysql> select * from kdata6;
```

```
+-----+
```

```
| test |
```

```
+-----+
```

```
| 99.9 | #因为 123.5 超过了 test float(3,1)能表示的范围，所以表示成最大值。
```

```
| 3.5 |
```

```
| 99.9 |
```

```
+-----+
```

3 rows in set (0.00 sec)

```
mysql> insert into kdata6 values (23.455);
```

Query OK, 1 row affected (0.00 sec)

```
mysql> select * from kdata6;
```

```
+-----+
```

```
| test |
```

```
+-----+
```

```
| 99.9 |
```

```
| 3.5 |
```

```
| 99.9 |
```

```
| 23.5 |
```

```
+-----+
```

4 rows in set (0.00 sec)

有效位数： 小数是 1 个 ， 整数就是 2 个

拓展：

double : 8 个字节来存储

decimal: 用来存储精确的小数，消耗空间，运算慢，一般想办法使用 bigint 替代它

在举例：(选)

```
mysql> create table ckdata4(tf float(5,2),td double(5,2),tdc
decimal(5,2));
```

```
mysql> insert into ckdata4 values(56.346,56.346,56.346);
```

```
mysql> select * from ckdata4;
```

```
+-----+-----+-----+
```

```
| tf      | td      | tdc     |
```

```
+-----+-----+-----+
```

```
| 56.35 | 56.35 | 56.35 |
```

```
+-----+-----+-----+
```

```
| .      | .      | .      |
```



```
mysql> create table ckdata5(tf float(5,2),td double(5,3),tdc
decimal(5,3));
```

```
mysql> insert into ckdata5 values(56.346,56.346,56.346);
```

```
mysql> create table ckdata6(tf float(10,2),td double(10,2),tdc
decimal(10,2));
```

```
mysql> insert into ckdata6
values(1234567.89,1234567.89,1234567.89);
```

```
mysql> select * from ckdata5;
```

tf	td	tdc
56.35	56.346	56.346

```
mysql> select * from ckdata6;
```

tf	td	tdc
1234567.88	1234567.89	1234567.89

这里丢失数据的原因是因为单精度浮点数的有效数字位位 8 位，而我们定义的 10 位，数据也是 10 位，所以最后两位就造成数据丢失了。

单精度浮点数在机内占 4 个字节，用 32 位二进制描述。

双精度浮点数在机内占 8 个字节，用 64 位二进制描述

39.5 字符串类型

char 和 varchar

char : 后面括号中必须用一个数值修饰,来确认字符串的范围。 大小范围 :0-255.

小于长度,空格自动补齐

大于长度,自动截短

【例 39.6】char(10); 指定了一个长度为 10 的字符值。

小于长度,空格自动补齐 大于长度,自动截短

```
mysql> create table kdata8 (aaa char(10));
Query OK, 0 rows affected (0.05 sec)
mysql> insert into kdata8 values ('1234567890111');
Query OK, 1 row affected, 1 warning (0.01 sec)
mysql> insert into kdata8 values ('abc');
Query OK, 1 row affected (0.00 sec)
mysql> select * from kdata8;
+-----+
| aaa      |
+-----+
| 1234567890 | #截断字符 111
| abc      |
+-----+
2 rows in set (0.00 sec)
字符串类类型: 区别大小写
binary 修饰符: 区分字符大小。比较 char 类型以一个二进制方式起作用。
```

【例 39.7】区分字符大小。

```
mysql> alter table kdata8 modify aaa char(10) binary;
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0
mysql> insert into kdata8 values ('ABC');
mysql> select * from kdata8 where aaa='abc';
+-----+
| aaa |
+-----+
| abc |
+-----+
1 row in set (0.00 sec)
mysql> select * from kdata8 where aaa='ABC';
+-----+
| aaa |
+-----+
create table kdata8 (aaa char(10));
```

查看不加修饰符的查询

```
mysql> alter table kdata8 modify aaa char(10);
Query OK, 3 rows affected (0.28 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql>
mysql> select * from kdata8 where aaa='ABC';
+-----+
| aaa   |
+-----+
| abc   |
| ABC   |
+-----+
2 rows in set (0.00 sec)
```

varchar : 可变长。

374

测试： varchar (4)

```
mysql> create table kdata88 (aaa varchar(4));
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> insert into kdata88 values ("");
Query OK, 1 row affected (0.00 sec)
```

```
mysql> insert into kdata88 values ("ab");
Query OK, 1 row affected (0.00 sec)
```

```
mysql> insert into kdata88 values ("abcd");
Query OK, 1 row affected (0.00 sec)
```

```
mysql> insert into kdata88 values ("abcdefg");
Query OK, 1 row affected, 1 warning (0.00 sec)
```

```
mysql> select * from kdata88;
```

```
+-----+
| aaa   |
+-----+
|      |
| ab    |
| abcd  |
| abcd  |
```

```
+-----+
```

```
4 rows in set (0.00 sec)
```

在 MySQL5.0 以上的版本中，varchar 数据类型的长度支持到了 65535，也就是说可以存放 65532 个字节的数据，起始位和结束位占去了 3 个字节。

varchar 字段是将实际内容单独存储在聚簇索引之外，内容开头用 1 到 2 个字节表示实际长度（长度超过 255 时需要 2 个字节），因此最大长度不能超过 65535。

$2^8 - 255$

39.6 char 和 varchar 区别

CHAR(M)定义的列的长度为固定的，M 取值可以为 0~255 之间，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。

VARCHAR(M)定义的列的长度为可变长字符串，M 取值可以为 0~65535 之间，(VARCHAR 的最大有效长度由最大行大小和使用的字符集确定。整体最大长度是 65,532 字节)。

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。

varchar 存储变长数据，但存储效率没有 CHAR 高。如果一个字段可能的值是不固定长度的，我们只知道它不可能超过 10 个字符，把它定义为 VARCHAR(10)是最合算的。

VARCHAR 类型的实际长度是它的值的实际长度+1。为什么"+1"呢？这一个字节用于保存

实际使用了多大的长度。

从空间上考虑，用 varchar 合适；从效率上考虑，用 char 合适。

总结：

1. 使用 VARCHAR 型字段时，你不需要为剪掉你数据中多余的空格而操心，增加用户操作的方便性
2. VARCHAR 型字段比 CHAR 型字段占用更少的内存和硬盘空间。当你的数据库很大时，这种内存和磁盘空间的节省会变得非常重要。

虽然 VARCHAR 使用起来较为灵活，但是从整个系统的性能角度来说，CHAR 数据类型的处理速度更快，有时甚至可以超出 VARCHAR 处理速度的 50%。

所以在设计数据库时应当综合考虑各方面的因素，以求达到最佳的平衡。

376

39.7 日期和时间类型

【例 39.8】date 日期

```
mysql> create table kdata10 (birthday date);
Query OK, 0 rows affected (0.04 sec)
mysql> insert into kdata10 values ('2018-01-23'),(20190304);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
mysql> insert into kdata10 value ('2018-01-23'),(20190304);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
mysql> select * from kdata10;
+-----+
| birthday |
+-----+
| 2018-01-23 |
| 2019-03-04 |
| 2018-01-23 |
| 2019-03-04 |
+-----+
4 rows in set (0.01 sec)
```

【例 39.9】time 时间

```
mysql> create table kdata11(showtime time);
Query OK, 0 rows affected (0.01 sec)
mysql> insert into kdata11 values ('11:10:23'),('11:23'),(112456);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0
mysql> select * from kdata11;
+-----+
| showtime |
+-----+
| 11:10:23 |
| 11:23:00 |
| 11:24:56 |
+-----+
3 rows in set (0.00 sec)
```

【例 39.10】 year : 00-69 : 转为 : 2000-2069 。 70-99 : 1970-1999 。

```
mysql> create table kdata13 (test year);
Query OK, 0 rows affected (0.02 sec)
mysql> insert into kdata13 values (2018),(04),(9),(69),(70);
Query OK, 5 rows affected (0.00 sec)
Records: 5 Duplicates: 0 Warnings: 0
mysql> select * from kdata13;
+-----+
| test |
+-----+
| 2018 |
| 2004 |
| 2009 |
| 2069 |
| 1970 |
+-----+
5 rows in set (0.00 sec)
```

【例 39.11】 datetime 或 timestamp

```
mysql> create table kdata14 ( f_datetime datetime,f_timestamp timestamp);
Query OK, 0 rows affected (0.06 sec)
mysql> insert into kdata14 values ('1999-11-12 23:23:45',19991112232345);
Query OK, 1 row affected (0.00 sec)
mysql> select * from kdata14;
+-----+-----+
| f_datetime          | f_timestamp          |
+-----+-----+
| 1999-11-12 23:23:45 | 1999-11-12 23:23:45 |
+-----+-----+
1 row in set (0.00 sec)
```

如果为空：

```
mysql> insert into kdata14 values (now(),null);
```

Query OK, 1 row affected (0.01 sec)

39.8 复合类型

它们字段的值，必须 从预先定义好的字符串集合中选取。

ENUM (枚举) : 只能取一个。 用于互斥。 男人，女人 。

set : 能取多个。

【例 39.12】枚举型

```
mysql> create table kdata16 (sex enum('M','F'));
Query OK, 0 rows affected (0.05 sec)
mysql> insert into kdata16 values ( 'M'),('m'),('F'),('yy'),('null');
Query OK, 5 rows affected, 2 warnings (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 2
mysql> select * from kdata16;
+-----+
| sex |
+-----+
| M   |
| M   |
| F   |
|     |
|     |
+-----+
5 rows in set (0.00 sec)
```

注意 :enum : 集合中最多包括 65536 个元素。 从 1 开始。 索引 0 ,表示错误值或 null。

如果输错，直接是空值。

【例 39.13】set 插入多个值，这样可以吗？

```
mysql> create table kdata17 ( type set('a','b','c','d','f'));
Query OK, 0 rows affected (0.05 sec)
mysql> insert into kdata17 values ('a');
Query OK, 1 row affected (0.00 sec)
mysql> insert into kdata17 values ('a','b');
ERROR 1136 (21S01): Column count doesn't match value count at row 1
```

解决：

```
mysql> insert into kdata17 values ('a,b');
```

```

Query OK, 1 row affected (0.00 sec)
mysql> insert into kdata17 values ('a,a,a,b'); #不能有一样的元素，有的话，无效。
Query OK, 1 row affected (0.00 sec)
mysql> select * from kdata17;
+-----+
| type |
+-----+
| a    |
| a,b  |
| a,b  |    #不能有一样的元素，有的话，无效。
+-----+
3 rows in set (0.00 sec)
mysql> insert into kdata17 values ('e'); #如果不存在些元素，以空白 来表示
mysql> select * from kdata17;
+-----+
| type |
+-----+
| a    |
| a    |
| a    |
| b    |
| a,b  |
| a,b  |
|      |
+-----+

```

注意：set 类型： 最包 64 类项。 在 set 中，相关的元素不能同时存在。

拓展：

Linux 显示时间戳：

```

[root@xuegod63 ~]# date +%s    #显示时间戳
1435798986

```

显示指定时间的时间戳

```

[root@365linux ~]# date -d "2010-07-20 10:25:30" +%s
1279592730

[root@xuegod63 ~]# date -d "1970-01-01 08:00:00" +%s
0

```

Linux 时间戳转日期

```

[root@xuegod63 ~]# date -d "@0"
Thu Jan  1 08:00:00 CST 1970

[root@xuegod63 ~]# date -d"@1435845465"
Thu Jul  2 21:57:45 CST 2015

```



```
[root@xuegod63 ~]# date -d"@1435845465" "+%Y/%m/%d %H:%M:%S"
2016/07/02 21:57:45
[root@xuegod63 ~]# date -d"@1435845465" "+%Y-%m-%d %H:%M:%S"
2016-07-02 21:57:45
```

面试题：

问题：

1.mysql 中什么数据类型能够储存路径

380 mysql 中，CHAR，VARCHAR，TEXT 等字符串类型都可以存储路径，但是，如果路径中使用\符号时候，这个符号会被过滤。解决办法是，路径中用/或者\\ 来代替\。这样 mysql 就不会自动过滤路径的分割字符，可以完整表示路径

2.mysql 中如何使用布尔类型？

在 sql 标准中，存在 bool 和 boolean 类型，mysql 为了支持 sql 标准也是可以定义 bool 和 boolean 类型的。但是 bool 和 boolean 类型最后转换成 TINYINT(1)。也就是说，在 mysql 中，布尔类型等价于 TINYINT(1)。因此，创建表的时候将一个字段定义成 BOOL 和 boolean 类型，数据库真实定义的是 TINYINT (1)

3.Mysql 中如何存储 JPG 图片和 MP3 音乐？

一般情况下，数据库中不直接存储图片和音频文件。而是存储图片和音频文件的路径。如果实在需要在 mysql 数据库中存储图片和音频文件，就选择 BLOB 类型，因为 blob 类型可以用来存储二进制类型的文件

mysql 报错序号对应的错误

1005：创建表失败

1006：创建数据库失败

1007：数据库已存在，创建数据库失败<=====可以忽略

1008: 数据库不存在, 删除数据库失败<=====可以忽略

1009: 不能删除数据库文件导致删除数据库失败

1010: 不能删除数据目录导致删除数据库失败

1011: 删除数据库文件失败

1012: 不能读取系统表中的记录

1020: 记录已被其他用户修改

1021: 硬盘剩余空间不足, 请加大硬盘可用空间

1022: 关键字重复, 更改记录失败

1023: 关闭时发生错误

1024: 读文件错误

1025: 更改名字时发生错误

1026: 写文件错误

1032: 记录不存在<=====可以忽略

1036: 数据表是只读的, 不能对它进行修改

1037: 系统内存不足, 请重启数据库或重启服务器

1038: 用于排序的内存不足, 请增大排序缓冲区

1040: 已到达数据库的最大连接数, 请加大数据库可用连接数

1041: 系统内存不足

1042: 无效的主机名

1043: 无效连接

1044: 当前用户没有访问数据库的权限

1045: 不能连接数据库, 用户名或密码错误

- 1048: 字段不能为空
- 1049: 数据库不存在
- 1050: 数据表已存在
- 1051: 数据表不存在
- 1054: 字段不存在
- 1062: 字段值重复, 入库失败<=====可以忽略
- 1065: 无效的 SQL 语句, SQL 语句为空
- 1081: 不能建立 Socket 连接
- 1114: 数据表已满, 不能容纳任何记录
- 1116: 打开的数据表太多
- 1129: 数据库出现异常, 请重启数据库
- 1130: 连接数据库失败, 没有连接数据库的权限
- 1133: 数据库用户不存在
- 1141: 当前用户无权访问数据库
- 1142: 当前用户无权访问数据表
- 1143: 当前用户无权访问数据表中的字段
- 1146: 数据表不存在
- 1147: 未定义用户对数据表的访问权限
- 1149: SQL 语句语法错误
- 1158: 网络错误, 出现读错误, 请检查网络连接状况
- 1159: 网络错误, 读超时, 请检查网络连接状况
- 1160: 网络错误, 出现写错误, 请检查网络连接状况

- 1161: 网络错误, 写超时, 请检查网络连接状况
- 1169: 字段值重复, 更新记录失败
- 1177: 打开数据表失败
- 1180: 提交事务失败
- 1181: 回滚事务失败
- 1203: 当前用户和数据库建立的连接已到达数据库的最大连接数, 请增大可用的数据库连接数或重启数据库
- 1205: 加锁超时
- 1211: 当前用户没有创建用户的权限
- 1216: 外键约束检查失败, 更新子表记录失败
- 1217: 外键约束检查失败, 删除或修改主表记录失败
- 1226: 当前用户使用的资源已超过所允许的资源, 请重启数据库或重启服务器
- 1227: 权限不足, 您无权进行此操作
- 1235: MySQL 版本过低, 不具有本功能