MySQL数据类型和表约束

一、MySQL 数据类型

- 整数类型：BIT、BOOL、TINYINT、SMALLINT、MEDIUMINT、 INT、 BIGINT
- 浮点数类型：FLOAT、DOUBLE、DECIMAL
- 字符串类型：CHAR、VARCHAR、TINYTEXT、TEXT、MEDIUMTEXT、LONGTEXT、TINYBLOB、BLOB、MEDIUMBLOB、LONGBLOB
- 日期类型：Date、DateTime、TimeStamp、Time、Year
- 其他数据类型：BINARY、VARBINARY、ENUM、SET、Geometry、Point、MultiPoint、LineString、MultiLineString、Polygon、GeometryCollection等

二、MySQL 定义列数据类型

```
mysql> create table 表名(
    字段名   列类型 unsigned [可选的参数],  # 记住加逗号
    字段名   列类型 [可选的参数],  # 记住加逗号
    字段名   列类型 [可选的参数]   # 最后一行不加逗号
)charset=utf8;                  # 后面加分号
```

1、数字

1、整型

MySQL数据类型	字节	比特位	最小值	最大值
tinyint(n)	1	8	-128	127
smallint(n)	2	16	-32,768	32,767
mediumint(n)	3	24	-8388608	8388607
int(n)	4	32	-2,147,483,648	2,147,483,647
bigint(n)	8	64	-9,223,372,036,854,775,808	9,223,372,036,854,775,807

- 当整数值超过 int 数据类型支持的范围时，就可以采用 bigint。
- 在 MySQL 中，int 数据类型是主要的整数数据类型。
- 只有当参数表达式是 bigint 数据类型时，函数才返回 bigint。MySQL 不会自动将其它整数数据类型（tinyint、smallint 和 int）提升为 bigint。
- int(n)里的n是表示SELECT查询结果集中的显示宽度，并不影响实际的取值范围，没有影响到显示的宽度

2、 整数实例

```
mysql> drop table if exists test_tinyint;
mysql> create table test_tinyint (
    num tinyint
) engine=innodb charset=utf8;

insert into test_tinyint values(-100);
insert into test_tinyint values(255);
```

- 执行代码时候报错 "Out of range value for column 'num' at row 1"，插入的数字范围越界了，说明MySQL中整型默认是带符号的。
- 把 num 字段定义改为 "num tinyint unsigned" 插入就不会报错了，但是的插入-100又报错了，因为无符号整型是无法表示负数的。

3、int(n) 需要注意

- 无论 N 等于多少，int 永远占 4 个字节
- N 表示的是显示宽度，不足的用 0 补足，超过的无视长度而直接显示整个数字，但要整型设置了unsigned zerofill才有效

4、int（n）实例

```
mysql> drop table if exists test_int_width;
mysql> create table test_int_width (
    a int(5),
    b int(5) unsigned,
    c int(5) unsigned zerofill,
    d int(8) unsigned zerofill
) engine=innodb charset=utf8;

mysql> insert into test_int_width values(1, 1, 1, 1111111111);

mysql> select * from test_int_width;
+------+------+-------+------------+
| a    | b    | c     | d          |
+------+------+-------+------------+
|    1 |    1 | 00001 | 1111111111 |
+------+------+-------+------------+
1 row in set (0.00 sec)
```

2、浮点型

- 在MySQL中浮点型有两种，分别为 float、double。

MySQL 浮点型数据类型	字节数	含义
float(m,d)	4	单精度浮点型 8位精度(4字节) m总个数，d小数位
double(m,d)	8	双精度浮点型 16位精度(8字节) m总个数，d小数位

- 设一个字段定义为 float(6,3)
- 如果插入一个数123.45678,实际数据库里存的是123.457，但总个数还以实际为准，即6位。整数部分最大是3位，
- 如果插入数12.123456，存储的是12.1234，如果插入12.12，存储的是12.1200.
- 下面还是用SQL来简单看一下float和double型数据，以float为例，double同理：

```
mysql> drop table if exists test_float;
mysql> create table test_float (
    num float(5, 2)
) engine=innodb charset=utf8;

mysql> insert into test_float values(1.233);
mysql> insert into test_float values(1.237);
mysql> insert into test_float values(10.233);
mysql> insert into test_float values(100.233);
mysql> insert into test_float values(1000.233);
mysql> insert into test_float values(10000.233);
mysql> insert into test_float values(100000.233);
```

- 查询结果：

```
mysql> select * from test_float;
+---------+
| num     |
+---------+
|    1.23 |
|    1.24 |
|   10.23 |
|   10.23 |
|  100.23 |
+---------+
```

- 从结果总结一下float(M,D)、double(M、D)的用法规则：
- D 表示浮点型数据小数点之后的精度，假如超过D位则四舍五入，即1.233四舍五入为1.23，1.237四舍五入为1.24
- M 表示浮点型数据总共的位数，D=2则表示总共支持五位，即小数点前只支持三位数，所以我们并没有看到1000.23、10000.233、100000.233这三条数据的插入，因为插入都报错了
- 注意： 当不指定M、D的时候，会按照实际的精度来处理。
- 实例

```
mysql> create table t5(
    id int auto_increment primary key,
    salary decimal(16,10),
    num float
)charset=utf8;

正好 10 位：
mysql> insert into t5 (salary, num) values (500023.2312345678,
5000.23748372847832748 32);
Query OK, 1 row affected (0.04 sec)

mysql> select * from t5;
+----+-------------------+---------+
| id | salary            | num     |
+----+-------------------+---------+
|  1 | 500023.2312345678 | 5000.24 |
+----+-------------------+---------+
1 row in set (0.00 sec)
少于10位：
```

```
mysql> insert into t5 (salary, num) values (500023.231234567,
5000.2374837284783274832);
Query OK, 1 row affected (0.04 sec)

mysql> select * from t5;
+----+------------------+---------+
| id | salary           | num     |
+----+------------------+---------+
|  1 | 500023.2312345678 | 5000.24 |
|  2 | 500023.2312345670 | 5000.24 |
+----+------------------+---------+
```

多于10位：

```
mysql> insert into t5 (salary, num) values (500023.23123456789,
5000.2374837284783274832);
Query OK, 1 row affected, 1 warning (0.03 sec)
mysql> select * from t5;
+----+------------------+---------+
| id | salary           | num     |
+----+------------------+---------+
|  1 | 500023.2312345678 | 5000.24 |
|  2 | 500023.2312345670 | 5000.24 |
|  3 | 500023.2312345679 | 5000.24 |
+----+------------------+---------+
```

3、定点数

- 浮点型在数据库中存放的是近似值，而定点类型在数据库中存放的是精确值。
- decimal(m,d) 参数m<65 是总个数，d<30且 d<m 是小数位。
- 定点型的数据类型decimal类型

```
mysql> drop table if exists test_decimal;
mysql> create table test_decimal (
    float_num float(10, 2),
    double_num double(20, 2),
    decimal_num decimal(20, 2)
) engine=innodb charset=utf8;

mysql> insert into test_decimal values(1234567.66, 1234567899000000.66,
1234567899000000.66);
mysql> insert into test_decimal values(1234567.66, 12345678990000000.66,
12345678990000000.66);
```

- 运行结果为：

```
mysql> select * from test_decimal;
+------------+---------------------+---------------------+
| float_num  | double_num          | decimal_num         |
+------------+---------------------+---------------------+
| 1234567.62 |  1234567899000000.80 |  1234567899000000.66 |
| 1234567.62 | 12345678990000000.00 | 12345678990000000.66 |
+------------+---------------------+---------------------+
2 rows in set (0.00 sec)
```

- decimal 无论写入数据中的数据是多少，都不会存在精度丢失问题，
- decimal 类型常见于银行系统、互联网金融系统等对小数点后的数字比较敏感的系统中。

- float/double 在db中存储的是近似值，而decimal则是以字符串形式进行保存的
- decimal(M,D)的规则和float/double相同，但区别在float/double在不指定M、D时默认按照实际精度来处理而decimal在不指定M、D时默认为decimal(10, 0)

4、字符串

MySQL数据类型	含义
char(m)	固定长度，最多255个字符
varchar(m)	可变长度，最多65535个字符
tinytext	可变长度，最多255个字符
text	可变长度，最多65535个字符
mediumtext	可变长度，最多2的24次方-1个字符
longtext	可变长度，最多2的32次方-1个字符

- char表示定长字符串，长度是固定的；如果插入数据的长度小于char的固定长度时，则用空格填充；因为长度固定，所以存取速度要比varchar快很多，甚至能快50%，但正因为其长度固定，所以会占据多余的空间，是空间换时间的做法；对于char来说，最多能存放的字符个数为255，和编码无关
- varchar表示可变长字符串，长度是可变的；插入的数据是多长，就按照多长来存储；varchar在存取方面与char相反，它存取慢，因为长度不固定，但正因如此，不占据多余的空间，是时间换空间的做法；对于varchar来说，最多能存放的字符个数为65532
- 结合性能角度（char更快），节省磁盘空间角度（varchar更小），具体情况还需具体来设计数据库才是妥当的做法。
- MySQL处理char类型数据时会将结尾的所有空格处理掉而varchar类型数据则不会

1、char 和 varchar

- char(n) 若存入字符数小于n，则以空格补于其后，查询之时再将空格去掉。所以char类型存储的字符串末尾不能有空格，varchar不限于此。
- char(n) 固定长度，char(4)不管是存入几个字符，都将占用4个字节，varchar是存入的实际字符数+1个字节（n<=255）或2个字节(n>255)，所以varchar(4),存入3个字符将占用4个字节。
- char 类型的字符串检索速度要比 varchar 类型的快。

```
mysql> drop table if exists test_string;
mysql> create table test_string (
    char_value char(5),
    varchar_value varchar(5)
) engine=innodb charset=utf8;

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

- 使用length函数来看一下结果

```
mysql> select length(char_value),length(varchar_value) from test_string;
+--------------------+----------------------+
| length(char_value) | length(varchar_value) |
+--------------------+----------------------+
|                  1 |                    1 |
|                  2 |                    2 |
|                  1 |                    2 |
|                  2 |                    3 |
+--------------------+----------------------+
4 rows in set (0.00 sec)
```

- 验证结论，char类型数据并不会取最后的空格。
- varchar 型数据占用空间大小及可容纳最大字符串限制

  1、创建表，utf8的编码格式

  ```
  mysql> drop table if exists test_varchar;
  mysql> create table test_varchar (
      varchar_value varchar(100000)
  ) engine=innodb charset=utf8;
  ```

  执行报错：

  ```
  ERROR 1074 (42000): Column length too big for column 'varchar_value' (max =
  21845); use BLOB or TEXT instead
  ```

  - 改为 21844就不会有问题，因此在utf8编码下varchar(n)，n最大=21845

2、创建表，gbk 的编码格式

```
mysql> drop table if exists test_varchar;
mysql> create table test_varchar (
    varchar_value varchar(100000)
) engine=innodb charset=gbk;
```

- 同样的报错：

```
ERROR 1074 (42000): Column length too big for column 'varchar_value' (max =
32767); use BLOB or TEXT instead
```

- 改为 32766就不会有问题，因此在 gbk 编码下varchar(n)，n最大=32767

3、创建表，utf8mb4 的编码格式

```
mysql> drop table if exists test_varchar;
mysql> create table test_varchar (
    varchar_value varchar(100000)
) engine=innodb charset=utf8mb4;
```

- 执行报错：

```
ERROR 1074 (42000): Column length too big for column 'varchar_value' (max =
16383); use BLOB or TEXT instead
```

- 改为 16382就不会有问题，因此在 utf8mb4 编码下varchar(n)，n最大=16383
- 不同编码格式下，varchar(n)最大的n不同，那么为什么会有这样的区别呢，分点详细解释一下：
- MySQL要求一个行的定义长度不能超过 65535 即 64K
- 对于未指定varchar字段not null的表，会有1个字节专门表示该字段是否为null
- varchar(n)，当M范围为0<=n<=255时会专门有一个字节记录varchar型字符串长度，当M>255时会专门有两个字节记录varchar型字符串的长度，把这一点和上一点结合，那么65535个字节实际可用的为65535-3=65532个字节
- 所有英文无论其编码方式，都占用1个字节，但对于gbk编码，一个汉字占两个字节，因此最大n=65532/2=32766；对于utf8编码，一个汉字占3个字节，因此最大M=65532/3=21844，
- utf8mb4 编码方式，1个字符最大可能占4个字节，那么varchar(n)，n最大为65532/4=16383
- varchar(n) n>255时转为tinytext
- varchar(n) n>500时转为text
- varchar(n) n>20000时转为mediumtext

2、varchar 和 text

- varchar可指定n，text不能指定，内部存储varchar是存入的实际字符数+1个字节（n<=255）或2个字节(n>255)，text是实际字符数+2个字节。
- varchar可直接创建索引，text创建索引要指定前多少个字符。varchar查询速度快于text,在都创建索引的情况下，text的索引似乎不起作用。
- MySQL单行最大数据量为64K,当varchar(M)的M大于某些数值时，varchar会自动转为text：
- 过大的内容 varchar 和 text没有区别，
- 单行64K即65535字节的空间，varchar只能用63352/65533个字节，但是 text可以65535个字节全部用起来
- text可以指定text(n)，但是n无论等于多少都没有影响
- text不允许有默认值，varchar允许有默认值
- varchar和text两种数据类型，使用建议是能用varchar就用varchar而不用text（存储效率高），varchar(M)的M有长度限制，如果大于限制可以使用mediumtext（16M）或者longtext（4G）。

5、二进制数据 (Blob)

- BLOB 和 text 存储方式不同，TEXT 以文本方式存储，英文存储区分大小写，而 Blob 是以二进制方式存储，不分大小写。
- BLOB 存储的数据只能整体读出。
- TEXT可以指定字符集，Blob 不用指定字符集。
- blob是用于存储例如图片、音视频这种文件的二进制数据的

6、日期时间类型

1、MySQL支持五种形式的日期类型

数据类型	字节数	格式	备注
date	3	yyyy-MM-dd	存储日期值
time	3	HH:mm:ss	存储时分秒
year	1	yyyy	存储年
datetime	8	yyyy-MM-dd HH:mm:ss	存储日期+时间
timestamp	4	yyyy-MM-dd HH:mm:ss	存储日期+时间，可作时间戳

2、实例

```
mysql> drop table if exists test_time;
mysql> create table test_time (
    date_value date,
    time_value time,
    year_value year,
    datetime_value datetime,
    timestamp_value timestamp
) engine=innodb charset=utf8;

mysql> insert into test_time values(now(), now(), now(), now(), now());
```

- 查询结果

```
mysql> select * from test_time;
+------------+------------+------------+---------------------+-----------------
---+
| date_value | time_value | year_value | datetime_value      | timestamp_value
  |
+------------+------------+------------+---------------------+-----------------
---+
| 2020-02-14 | 10:33:49   |       2020 | 2020-02-14 10:33:49 | 2020-02-14
10:33:49 |
+------------+------------+------------+---------------------+-----------------
---+
1 row in set (0.00 sec)
```

- datetime占8个字节，timestamp占4个字节
- 由于大小的区别，datetime 与 timestamp 能存储的时间范围也不同，datetime 的存储范围为1000-01-01 00:00:00——9999-12-31 23:59:59，timestamp存储的时间范围为19700101080001——20380119111407
- datetime 默认值为空，当插入的值为 null 时，该列的值就是null；timestamp默认值不为空，当插入的值为 null 的时候，mysql会取当前时间
- datetime 存储的时间与时区无关，timestamp 存储的时间及显示的时间都依赖于当前时区
- 在实际工作中，一张表往往会有两个默认字段，一个记录创建时间而另一个记录最新一次的更新时间，这种时候可以使用 timestamp 类型来实现.

6、枚举

- 枚举用于列出所有的选项

```
mysql> create table t9 (
    id int auto_increment primary key,
    gender enum('male','female')
)charset utf8;

mysql> insert into t9 (gender) values ('male');
Query OK, 1 row affected (0.04 sec)

mysql> insert into t9 (gender) values ('female');
Query OK, 1 row affected (0.03 sec)

mysql> insert into t9 (gender) values ('dshajjdsja');
```

7、MYSQL 数据类型的长度和范围

- MySQL各数据类型及字节长度：

数据类型	字节长度	范围或用法

数据类型	字节长度	范围或用法
Bit	1	无符号[0,255]，有符号[-128,127]，天缘博客备注：BIT和BOOL布尔型都占用1字节
TinyInt	1	整数[0,255]
SmallInt	2	无符号[0,65535]，有符号[-32768,32767]
MediumInt	3	无符号[0,2^24-1]，有符号[-2^23,2^23-1]]
Int	4	无符号[0,2^32-1]，有符号[-2^31,2^31-1]
BigInt	8	无符号[0,2^64-1]，有符号[-2^63 ,2^63 -1]
Float(M,D)	4	单精度浮点数。天缘博客提醒这里的D是精度，如果D<=24则为默认的FLOAT，如果D>24则会自动被转换为DOUBLE型。
Double(M,D)	8	双精度浮点。
Decimal(M,D)	M+1或M+2	未打包的浮点数，用法类似于FLOAT和DOUBLE，天缘博客提醒您如果在ASP中使用到Decimal数据类型，直接从数据库读出来的Decimal可能需要先转换成Float或Double类型后再进行运算。
Date	3	以YYYY-MM-DD的格式显示，比如：2009-07-19
Date Time	8	以YYYY-MM-DD HH:MM:SS的格式显示，比如：2009-07-19 11：22：30
TimeStamp	4	以YYYY-MM-DD的格式显示，比如：2009-07-19
Time	3	以HH:MM:SS的格式显示。比如：11：22：30
Year	1	以YYYY的格式显示。比如：2009
Char(M)	M	定长字符串。
VarChar(M)	M	变长字符串，要求M<=255
Binary(M)	M	类似Char的二进制存储，特点是插入定长不足补0
VarBinary(M)	M	类似VarChar的变长二进制存储，特点是定长不补0
Tiny Text	Max:255	大小写不敏感
Text	Max:64K	大小写不敏感
Medium Text	Max:16M	大小写不敏感
Long Text	Max:4G	大小写不敏感
TinyBlob	Max:255	大小写敏感
Blob	Max:64K	大小写敏感
MediumBlob	Max:16M	大小写敏感
LongBlob	Max:4G	大小写敏感
Enum	1或2	最大可达65535个不同的枚举值

数据类型	字节长度	范围或用法
Set	可达8	最大可达64个不同的值

8、MySQL 数据类型使用建议

- 在指定数据类型的时候一般是采用从小原则，比如能用TINYINT的最好就不用INT，能用FLOAT类型的就不用DOUBLE类型，这样会对MYSQL在运行效率上提高很大，尤其是大数据量测试条件下。
- 不需要把数据表设计的太过复杂，功能模块上区分或许对于后期的维护更为方便，慎重出现大杂烩数据表
- 数据库的最后设计结果一定是效率和可扩展性的折中，偏向任何一方都是欠妥的

9、MySQL 选择数据类型的基本原则

- 根据选定的存储引擎，确定如何选择合适的数据类型。
- MyISAM 数据存储引擎和数据列：MyISAM数据表，最好使用固定长度(CHAR)的数据列代替可变长度(VARCHAR)的数据列。
- MEMORY存储引擎和数据列：MEMORY数据表目前都使用固定长度的数据行存储，因此无论使用CHAR或VARCHAR列都没有关系。两者都是作为CHAR类型处理的。
- InnoDB 存储引擎和数据列：建议使用 VARCHAR类型。对于InnoDB数据表，内部的行存储格式没有区分固定长度和可变长度列（所有数据行都使用指向数据列值的头指针），因此在本质上，使用固定长度的CHAR列不一定比使用可变长度VARCHAR列简单。因而，主要的性能因素是数据行使用的存储总量。由于CHAR平均占用的空间多于VARCHAR，因 此使用VARCHAR来最小化需要处理的数据行的存储总量和磁盘I/O是比较好的。

三、MySQL 数据类型约束

- 约束是一种限制，它通过对表的行或者列的数据做出限制，来确保表数据的完整性和唯一性. 在mysql当中一般有一下这几种约束：

MySQL关键字	含义
NULL	数据列可包含NULL值
NOT NULL	数据列不允许包含NULL值
DEFAULT	默认值
PRIMARY KEY	主键
AUTO_INCREMENT	自动递增，适用于整数类型
UNSIGNED	无符号
CHARACTER SET name	指定一个字符集

1、非空约束

- 就是限制数据库中某个值是否可以为空，null字段值可以为空，not null字段值不能为空

```
mysql> create table testnull(id int, username varchar(20) not null);    # 创建 testnull 设置 username 字段为非空约束 notnull
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> insert into testnull(id,username) values(1,'qfedu');              # 插入2个
数据2个字段
Query OK, 1 row affected (0.01 sec)

mysql> insert into testnull (id) values(2);                              # 没有插
入 username 字段报错，username 必须有一个默认值
ERROR 1364 (HY000): Field 'username' doesn't have a default value
mysql> desc testnull;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| id       | int(11)     | YES  |     | NULL    |       |
| username | varchar(20) | NO   |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

- 注意： 如果约束不生效可以先设置一下sql_mode

```
mysql> set session sql_mode='STRICT_TRANS_TABLES';
```

2、唯一约束

- 字段添加唯一约束之后，该字段的值不能重复，也就是说在一列当中不能出现一样的值。

```
# 添加唯一约束
ALTER TABLE tbl_name ADD [CONSTRAINT[symbol]]
UNIQUE [INDEX|KEY] [index_name] [index_type] (index_col_name)

# 删除唯一约束
ALTERT TABLE tbl_name DROP {INDEX|KEY} index_name
```

- 已经添加的值不能再重复的插入

```
mysql> select * from testnull;  # 查询表里面的数据
+------+----------+
| id   | username |
+------+----------+
|    1 | qfedu    |
+------+----------+
1 row in set (0.00 sec)

mysql> alter table testnull add unique(id);   # 设置 id 字段添加一个唯一约束 unique
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> insert into testnull(id,username) values(1,'qfedu1');    # 插入一个值（1）给
id 报错，提示不能重复值
ERROR 1062 (23000): Duplicate entry '1' for key 'id'
```

3、主键约束

- 主键保证记录的唯一性，主键自动为NOT NULL
- 每张数据表只能存在一个主键 NOT NULL + UNIQUE KEY
- 一个UNIQUE KEY 又是一个NOT NULL的时候，那么它被当做PRIMARY KEY主键

- 当一张表里没有一个主键的时候，第一个出现的非空且为唯一的列被视为有主键。

```
# 添加主键约束
ALTER TABLE tbl_name ADD [CONSTRAINT[sysbol]]
PRIMARY KEY [index_type] (index_col_name)

# 删除主键约束
ALTER TABLE tbl_name DROP PRIMARY KEY
```

实例

```
mysql> create table user(id int primary key, name varchar(20),number int);
Query OK, 0 rows affected (0.01 sec)

mysql> desc user;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int(11)     | NO   | PRI | NULL    |       |
| name   | varchar(20) | YES  |     | NULL    |       |
| number | int(11)     | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> insert into user(id,name,number) values(1,'1',1);
Query OK, 1 row affected (0.00 sec)

mysql> select * from user;
+----+------+--------+
| id | name | number |
+----+------+--------+
|  1 | 1    |      1 |
+----+------+--------+
1 row in set (0.00 sec)

mysql> insert into user(id,name,number) values(1,'1',1);
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
mysql> insert into user(id,name,number) values(1,'2',2);
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
mysql> insert into user(id,name,number) values(2,'2',2);
Query OK, 1 row affected (0.00 sec)

mysql> select * from user;
+----+------+--------+
| id | name | number |
+----+------+--------+
|  1 | 1    |      1 |
|  2 | 2    |      2 |
+----+------+--------+
2 rows in set (0.00 sec)

mysql> insert into user(id,name,number) values(0,'3',3);
Query OK, 1 row affected (0.01 sec)

mysql> select * from user;
+----+------+--------+
| id | name | number |
```

```
+----+------+--------+
|  0 | 3    |      3 |
|  1 | 1    |      1 |
|  2 | 2    |      2 |
+----+------+--------+
3 rows in set (0.00 sec)
```

4、自增长

- 自增长 AUTO_INCREMENT 自动编号，且必须与主键组合使用 默认情况下，起始值为1，每次的增量为1。 当插入记录时，如果为AUTO_INCREMENT数据列明确指定了一个数值，则会出现两种情况：
- 如果插入的值与已有的编号重复，则会出现出错信息，因为AUTO_INCREMENT数据列的值必须是唯一的；
- 如果插入的值大于已编号的值，则会把该插入到数据列中，并使在下一个编号将从这个新值开始递增。也就是说，可以跳过一些编号。如果自增序列的最大值被删除了，则在插入新记录时，该值被重用。

1、添加自增长

```
mysql> ALTER TABLE user CHANGE id id INT NOT NULL AUTO_INCREMENT;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE user;
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| id     | int(11)     | NO   | PRI | NULL    | auto_increment |
| name   | varchar(20) | YES  |     | NULL    |                |
| number | int(11)     | YES  |     | NULL    |                |
+--------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM user;
Empty set (0.00 sec)
```

2、插入值

```
mysql> INSERT INTO user(name, number) VALUES('take',2), ('which',4);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM `user`;
+----+-------+--------+
| id | name  | number |
+----+-------+--------+
|  1 | take  |      2 |
|  2 | which |      4 |
+----+-------+--------+
2 rows in set (0.00 sec)
```

3、删除自增长

```
mysql> ALTER TABLE user CHANGE id  id INT NOT NULL;
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> DESCRIBE user;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int(11)     | NO   | PRI | NULL    |       |
| name   | varchar(20) | YES  |     | NULL    |       |
| number | int(11)     | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

5、默认约束

1、添加/删除默认约束

```
ALTER TABLE tbl_name ALTER [COLUMN] col_name {SET DEFAULT literal | DROP
DEFAULT}
```

2、设置默认值

```
mysql> ALTER TABLE user ALTER number SET DEFAULT 0;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE user;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int(11)     | NO   | PRI | NULL    |       |
| name   | varchar(20) | YES  |     | NULL    |       |
| number | int(11)     | YES  |     | 0       |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

3、插入值

```
mysql> ALTER TABLE user CHANGE id id INT NOT NULL AUTO_INCREMENT;
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> INSERT INTO user(name) VALUES('rock');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO user(name) VALUES('rock');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM user;
+----+-------+--------+
| id | name  | number |
+----+-------+--------+
|  1 | take  |      2 |
|  2 | which |      4 |
```

```
|  3 | rock  |      0 |
|  4 | rock  |      0 |
+----+-------+--------+
4 rows in set (0.00 sec)
```

6、外键约束

- 外键约束要求数据表的存储引擎只能为 InnoDB

查看当前mysql服务器支持的存储引擎

```
mysql> SHOW ENGINES;
```

1、编辑数据表的默认存储引擎

```
[root@qfedu.com ~]# vim /etc/my.cnf
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
character-set-server=utf8mb4
default-storage-engine=INNODB
[root@qfedu.com ~]# systemctl restart mysqld
```

2、实例

```
mysql> create table teacher(id int primary key auto_increment, name varchar(20)
not null);   # 创建老师表
Query OK, 0 rows affected (0.00 sec)

mysql> create table students(id int primary key auto_increment, name varchar(20)
not null,teacher_id int,foreign key(teacher_id) references teacher(id));
     # 创建学生表
Query OK, 0 rows affected (0.01 sec)

mysql> insert into teacher(id,name) values(1,'qfedu');
Query OK, 1 row affected (0.00 sec)

mysql> insert into students(id,name,teacher_id) values(1,'qfedu',1);
            # 插入老师表中id来关联
Query OK, 1 row affected (0.00 sec)

mysql> select * from students;
+----+-------+------------+
| id | name  | teacher_id |
+----+-------+------------+
|  1 | qfedu |          1 |
+----+-------+------------+
1 row in set (0.00 sec)

mysql> select * from teacher;
+----+-------+
| id | name  |
+----+-------+
|  1 | qfedu |
+----+-------+
```

```
1 row in set (0.00 sec)
```

四、MySQL 索引

- 索引作为一种数据结构，其用途是用于提升检索数据的效率。

1、MySQL 索引的分类

- 普通索引（INDEX）：索引列值可重复
- 唯一索引（UNIQUE）：索引列值必须唯一，可以为NULL
- 主键索引（PRIMARY KEY）：索引列值必须唯一，不能为NULL，一个表只能有一个主键索引
- 全文索引（FULL TEXT）：给每个字段创建索引

2、MySQL 不同类型索引用途和区别

- 普通索引常用于过滤数据。例如，以商品种类作为索引，检索种类为“手机”的商品。
- 唯一索引主要用于标识一列数据不允许重复的特性，相比主键索引不常用于检索的场景。
- 主键索引是行的唯一标识，因而其主要用途是检索特定数据。
- 全文索引效率低，常用于文本中内容的检索。

3、MySQL 使用索引

1、创建索引

1、普通索引（INDEX）

```
# 在创建表时指定
mysql> CREATE TABLE student (
    id INT NOT NULL,
    name VARCHAR(100) NOT NULL,
    birthday DATE,
    sex CHAR(1) NOT NULL,
    INDEX nameIndex (name(50))
);

# 基于表结构创建
mysql> CREATE INDEX nameIndex ON student(name(50));

# 修改表结构创建
mysql> ALTER TABLE student ADD INDEX nameIndex(name(50));
```

2、唯一索引（UNIQUE）

```
# 在创建表时指定
mysql> CREATE TABLE student (
    id INT NOT NULL,
    name VARCHAR(100) NOT NULL,
    birthday DATE,
    sex CHAR(1) NOT NULL,
    UNIQUE idIndex (id)
);
# 基于表结构创建
mysql> CREATE UNIQUE INDEX idIndex ON student(id)
```

3、主键索引（PRIMARY KEY）

```
# 创建表时时指定
mysql> CREATE TABLE student (
    id INT,
    name VARCHAR(100) NOT NULL,
    birthday DATE,
    sex CHAR(1) NOT NULL,
    PRIMARY KEY (id)
);

# 修改表结构创建
mysql> ALTER TABLE student ADD PRIMARY KEY (id);
```

主键索引不能使用基于表结构创建的方式创建。

2、删除索引

1、普通索引（INDEX）

```
# 直接删除
mysql> DROP INDEX nameIndex ON student;

# 修改表结构删除
mysql> ALTER TABLE student DROP INDEX nameIndex;
```

2、唯一索引（UNIQUE）

```
# 直接删除
mysql> DROP INDEX idIndex ON student;

# 修改表结构删除
mysql> ALTER TABLE student DROP INDEX idIndex;
```

3、主键索引（PRIMARY KEY）

```
mysql> ALTER TABLE student DROP PRIMARY KEY;
```

主键不能采用直接删除的方式删除。

3、查看索引

```
mysql> SHOW INDEX FROM student;
```

4、选择索引的原则

- 常用于查询条件的字段较适合作为索引，例如WHERE语句和JOIN语句中出现的列
- 唯一性太差的字段不适合作为索引，例如性别
- 更新过于频繁（更新频率远高于检索频率）的字段不适合作为索引
- 使用索引的好处是索引通过一定的算法建立了索引值与列值直接的联系，可以通过索引直接获取对应的行数据，而无需进行全表搜索，因而加快了检索速度
- 但由于索引也是一种数据结构，它需要占据额外的内存空间，并且读取索引也加会大IO资源的消耗，因而索引并非越多越好，且对过小的表也没有添加索引的必要