数据表管理

一、相关概念

1、表（关系表）

数据表是由行和列组成的二维表格，是存储数据的载体。

2、字段（表的结构）

二维表格中的列头

3、记录（具体的数据）

表示数据列的集合（表示数据的那些行，**而列表示某类数据的抽象**）

4、表间关系

表与表之间的关联关系

5、关键字

起到关键作用的字段（列）

1）候选关键字：能唯一标识记录不重复的字段（集合）

2）主键字(主键,PK[primary key])：在候选关键字，选择一个作为主关键字

3）公共关键字：多张表中，公共存在的字段(不一定名称一样，意义一样也行)

4）外关键字(外键,FK[foreign key])：在公共关键字中，一个是主键，则另一个必定是外键

注：

- 候选关键字：0个或多个
- 主键个数：0或1
- 外键个数：0或多个
- 主键字段名和外键字段名可以不一样,但是意义要一致

二、数据类型

1、整数类型

数据类型	范围	字节
tinyint	有符号值：-128到127 无符号值：0到255	1
smallint	? smallint 或 help smallint 或 \h smallint	2
mediumint	同上，详情查看文档帮助	3
int（integer）	同上，详情查看文档帮助	4
bigint	同上，详情查看文档帮助	8
Bool,boolean	等价于tinyint(1)，0为false，其它为true	1

2、浮点数类型

float[(m,d)]

double[(m,d)]

3、定点数类型

decimal(M,D)或dec(M，D)： 内部是以字符串形式存储数值。其中，M是数值总位数，D是保留位数。DECIMAL 的存储空间并不是固定的，而由精度值 M 决定，占用 M+2 个字节。

浮点数 VS 定点数

- FLOAT 和 DOUBLE 在不指定精度时，默认会按照实际的精度（由计算机硬件和操作系统决定），DECIMAL 如果不指定精度，默认为（10，0）；
- 不论是定点还是浮点类型，如果用户指定的精度超出精度范围，则会四舍五入进行处理；
- 浮点数相对于定点数的优点是在长度一定的情况下，浮点数能够表示更大的范围；缺点是会引起精度问题；
- 在 MySQL 中，定点数以字符串形式存储，在对精度要求比较高的时候（如货币、科学数据），使用 DECIMAL 的类型比较好，另外两个浮点数进行减法和比较运算时也容易出问题，所以在使用浮点数时需要注意，并尽量避免做浮点数比较。

4、字符类型

char(m)：固定（定长）m个字符

varchar(m)：变长字符串

tinytext

text

mediumtext

longtext

enum('值1','值2','值3',...)：枚举类型，只选择其中一个值

set('值1','值2','值3',...)：可选择多个值

5、日期时间类型

1）time：时：分：秒

2）date：年-月-日

3）datetime：年-月-日 时：分：秒

4）timestamp：如果需要经常插入或者更新日期为当前系统时间，则通常使用 TIMESTAMP 来表示。

5）year：年

TIMESTAMP 与 DATETIME 除了存储字节和支持的范围不同外，还有一个最大的区别是：

- DATETIME 在存储日期数据时，按实际输入的格式存储，即输入什么就存储什么，与时区无关；
- 而 TIMESTAMP 值的存储是以 UTC（世界标准时间）格式保存的，存储时对当前时区进行转换，检索时再转换回当前时区。即查询时，根据当前时区的不同，显示的时间值是不同的。

> 提示：如果为一个 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'，因此 DATE 值未包含时间信息。如果为一个 DATE 对象分配一个 DATETIME 或 TIMESTAMP 值，结果值的时间部分被删除，因此DATE 值未包含时间信息。

6、二进制类型

1）bit(M)

```
# bit(M)，M的取值范围1-64，默认为1
DROP TABLE IF EXISTS t ;
CREATE TABLE t (
  c bit(6)
);

# 如果为 BIT(M) 列分配的值的长度小于 M 位，在值的左边用 0 填充
# 13的二进制为1101，则以下插入值为：001101
INSERT INTO t values (13);

# 错误，64的二进制为1000000，溢出
INSERT INTO t values (64);
```

2）varbinary(m)

3）binary(m)

BINARY 和 VARBINARY 类型类似于 CHAR 和 VARCHAR，不同的是它们包含二进制字节字符串。

- BINARY 类型的长度是固定的，指定长度后，不足最大长度的，将在它们右边填充 “\0” 补齐，以达到指定长度；
- VARBINARY 类型的长度是可变的，指定好长度之后，长度可以在 0 到最大值之间。

```
# BINARY 类型的长度是固定的，指定长度后，不足最大长度的，将在它们右边填充 “\0” 补齐，以达到指定长度。
DROP TABLE IF EXISTS t ;
CREATE TABLE t (
  c binary(3)
);

# 插入的值为："a\0\0"
INSERT INTO t SET c='a';

# 验证1
select c,c='a',c='a\0\0' from t ;

# 插入的值为：'ab\0'
INSERT INTO t SET c='ab';

# 验证2
select c,c='ab',c='a\0\0',c='ab\0' from t ;
```

4）tinyblog(M)

5）blob(M)

6）mediumblog(M)

7）longblog(M)

BLOB 是一个二进制的对象，用来存储可变数量的数据。BLOB 类型分为 4 种：TINYBLOB、BLOB、MEDIUMBLOB 和 LONGBLOB，它们可容纳值的最大长度不同。BLOB 列存储的是二进制字符串（字节字符串），TEXT 列存储的是非进制字符串（字符字符串）。BLOB 列是字符集，并且排序和比较基于列值字节的数值；TEXT 列有一个字符集，并且根据字符集对值进行排序和比较。

三、存储引擎

1、概念

数据库存储引擎是数据库底层软件组件，数据库管理系统使用数据引擎进行创建、查询、更新和删除数据操作。简单来说，存储引擎就是表的类型。数据库的存储引擎决定了表在计算机中的存储方式。不同的存储引擎提供不同的存储机制、索引技巧、锁定水平等功能，使用不同的存储引擎还可以获得特定的功能。用户可以根据不同的存储方式，是否进行事务处理等来选择合适的存储引擎。

存储引擎	描述
ARCHIVE	用于数据存档的引擎，数据被插入后就不能在修改了，且不支持索引。
CSV	在存储数据时，会以逗号作为数据项之间的分隔符。
BLACKHOLE	会丢弃写操作，该操作会返回空内容。
FEDERATED	将数据存储在远程数据库中，用来访问远程表的存储引擎。
InnoDB	具备外键支持功能的事务处理引擎
MEMORY	置于内存的表
MERGE	用来管理由多个 MyISAM 表构成的表集合
MyISAM	主要的非事务处理存储引擎
NDB	MySQL 集群专用存储引擎

说明：有几种存储引擎的名字还有同义词，例如，MRG_MyISAM 和 NDBCLUSTER 分别是 MERGE 和 NDB 的同义词。存储引擎 MEMORY 和 InnoDB 在早期分别称为 HEAP 和 Innobase。虽然后面两个名字仍能被识别，但是已经被废弃了。

2、常用命令

```
# 查看MySQL支持的存储引擎
# YES表示可以使用，NO表示不能使用，DEFAULT表示该引擎为当前默认的存储引擎。
show engines;
show engines \G

# 查看存储引擎信息
show variables like 'have%'

# 查看默认的存储引擎
show variables like 'storage_engine'
```

3、常用的存储引擎及特点

3.1、MyISAM

在 MySQL 5.1 版本及之前的版本，MyISAM 是默认的存储引擎。

MyISAM 存储引擎不支持事务和外键，所以访问速度比较快。如果应用主要以读取和写入为主，只有少量的更新和删除操作，并且对事务的完整性、并发性要求不是很高，那么选择 MyISAM 存储引擎是非常适合的。

MyISAM 是在 Web 数据仓储和其他应用环境下最常使用的存储引擎之一。

1）创建MyISAM表之后，默认产生三个文件

- .frm：表结构文件
- .MYD：数据文件
- .MYI：索引文件

2）在创建时指定数据文件和索引文件的位置（只有MySSAM表支持）

```
DATA DIRECTORY [=] 数据文件保存的绝对路径
INDEX DIRECTORY [=] 索引文件保存的绝对路径
```

3）MyISAM单表最大支持的数据量为2的64次方条记录

4）每张表最多可以建立64个索引

5）如果是复合索引，每个复合索引最多包含16个列，索引值最大长度是1000B

6）MyISAM引擎的存储格式

- 定长（FIXED静态）：是指字段中不包含VARCHAR/TEXT/BLOB

- 动态（DYNAMIC）：只要字段中包含了VARCHAR/TEXT/BLOB
- 压缩（COMPRESSED）：myisampack创建

7）MySQL存储引擎从5.5.5以后，InnoDB是默认引擎，之前默认是MyISAM

3.2、InnoDB

MySQL 5.5 版本之后默认的事务型引擎修改为 InnoDB。

InnoDB 存储引擎在事务上具有优势，即支持具有提交、回滚和崩溃恢复能力的事务安装，所以比 MyISAM 存储引擎占用更多的磁盘空间。

如果应用对事务的完整性有比较高的要求，在并发条件下要求数据的一致性，数据操作除了插入和查询以外，还包括很多的更新、删除操作，那么 InnoDB 存储引擎是比较合适的选择。

InnoDB 存储引擎除了可以有效地降低由于删除和更新导致的锁定，还可以确保事务的完整提交（Commit）和回滚（Rollback），对于类似计费系统或者财务系统等对数据准确性要求比较高的系统，InnoDB 都是合适的选择。

1）创建InnoDB表之后，默认产生两个文件

- .frm：表结构文件
- .ibd：存储数据和索引

2）设计遵循ACID模型，支持事务，具有从服务崩溃中恢复的能力，能够最大限度保护用户的数据

3）支持行级锁，可以提升多用户并发时的读写性能

4）支持外键，保证数据的一致性和完整性

5）InnoDB拥有自己独立的缓冲池，常用的数据和索引都在缓存中

6）对INSERT、UPDATE、DELETE操作，InnoDB会使用一种change buffering的机制来自动优化，还可以提供一致性的读，并且还能够缓存变量的数据，减少磁盘I/O，提高性能

7）所有的表都需要创建主键，最好是配合上AUTO_INCERMENT，也可以放到经常查询的列作为主键。

3.3、MEMORY

MEMORY 存储引擎将所有数据保存在 RAM 中，所以该存储引擎的数据访问速度快，但是安全上没有保障。

MEMORY 对表的大小有限制，太大的表无法缓存在内存中。由于使用 MEMORY 存储引擎没有安全保障，所以要确保数据库异常终止后表中的数据可以恢复。

如果应用中涉及数据比较少，且需要进行快速访问，则适合使用 MEMORY 存储引擎。

四、创建表

1、语法

官方文档：https://dev.mysql.com/doc/refman/5.7/en/create-table.html

```
create table [if not exists] tbl_表名 (
  字段名称1 数据类型 [unsigned|zerofill] [约束] ,
  ...
  字段名称n 数据类型  [约束]
)[engine=引擎名称 charset='编码方式'];
```

```
#创建数据库
CREATE DATABASE IF NOT EXISTS test DEFAULT CHARACTER SET 'UTF8';

#进入test数据库
USE test;

#注意：当需要输入中文时，需要临时转换客户端的编码方式，设置如下：
SET NAMES GBK;
#查看指令
status 或 \s


#数据表的相关文件
#################################################################
#例子一：创建学生信息表
CREATE TABLE student (
    id SMALLINT,
    username VARCHAR(50) ,
    age TINYINT ,
    SEX ENUM('男','女','保密'),
    height FLOAT(5,2) ,
    birth date ,
    phone INT ,
    finished TINYINT(1) COMMENT '0表示未毕业，非0表示已毕业'
)ENGINE=INNODB CHARSET=UTF8;
#存储引擎为INNODB，生成的文件有如下二个：
#1）*.frm：保存了每个表的元数据，包括表结构的定义等，该文件与数据库引擎无关。
```

```
#2) *.ibd: InnoDB引擎开启了独立表空间(my.ini中配置innodb_file_per_table = 1)产生的，存放该表的数据和索引。

#查看是否打开数据库
SELECT DATABASE();

#例子二：创建新闻分类表
CREATE TABLE cms_cate (
    id SMALLINT ,
    cate_name VARCHAR(50) ,
    cate_desc VARCHAR(200)
)ENGINE=MyISAM CHARSET=UTF8;
#存储引擎为MyISAM，生成的文件有如下三个：
#1) *.frm文件：同上
#2) *.MYD文件：存储表数据
#3) *.MYI文件：存储表索引
```

2、数据类型

```
#例子三：浮点型数据
CREATE TABLE test3 (
    num1 float(6,2) ,
    num2 double(6,2) ,
    num3 decimal(6,2)
);
#保留两位小数并支持四舍五入
insert into test3 values (3.14159,3.14159,3.14159);
insert into test3 values (3.1459,3.1459,3.1459);

#注意：decimal是以字符串的方式存储的
select * from test3 where num1=3.14;    #查询成功
select * from test3 where num1='3.14';    #查询失败

select * from test3 where num3=3.14;    #查询成功
select * from test3 where num3='3.14';    #查询成功

#例子四：枚举类型
CREATE TABLE IF NOT EXISTS test4 (
  sex ENUM('男','女','保密')
);
insert into test4 values ('男  '); # 空格会自动去除
insert into test4 values ('女');
insert into test4 values ('保密');
insert into test4 values ('妖');   # 错误
insert into test4 values (NULL);  # 可以允许为NULL
insert into test4 values (2); # 通过索引添加，索引顺序为1,2,3

#例子五：集合类型
CREATE TABLE IF NOT EXISTS test5 (
```

```
  sex SET('A','B','C','D')
);
insert into test5 values ('A,D,E');   #错误，值必须在集合范围中，且使用逗号分隔
insert into test5 values ('A,C,D');   #正确
insert into test5 values ('D,B,A');   #正确,添加时数据顺序无关，添加后数据自动排序

#注意：集合数据以二进制形式存储，即数据A,B,C,D对应的数值为：1,2,4,8。
insert into test5 values (1);     #添加A
insert into test5 values (3);     #添加A、B
insert into test5 values (6);     #添加B、C
insert into test5 values (14);      #添加B、C、D


#例子六：YEAR类型
CREATE TABLE IF NOT EXISTS test6 (
  birth YEAR
);
#year类型的取值范围是1901-2155
insert into test6 values (1901);
insert into test6 values (2155);
insert into test6 values (50);    #1-69两位数，会加2000，结果为：2050
insert into test6 values (99);    #70-99两位数，会加1900，结果为：1999

insert into test6 values (0);   #0000
insert into test6 values ('0');   #2000
insert into test6 values ('00');  #2000

#例子七：TIME类型 => 时：分：秒
CREATE TABLE IF NOT EXISTS test7 (
  birth TIME
);

insert into test7 values ('1 12:12:12');  #12时12分12秒 + 1天，结果为：36:12:12
insert into test7 values ('12:12');     #12时12分，省略了秒，结果为：12:12:00
insert into test7 values (112233);      #结果为：12:22:33:
insert into test7 values (2233);        #结果为：00:22:33
insert into test7 values (33);          #结果为：00:00:33
insert into test7 values (0);           #结果为：00:00:00
#注意：小时的取值范围为：-838到838

#例子八：DATE类型 => 年-月-日
CREATE TABLE IF NOT EXISTS test8 (
  birth DATE
);

insert into test8 values ('12-7-5');    #2012-07-05
insert into test8 values ('12/7/5');    #2012-07-05
insert into test8 values ('12#7$5');    #2012-07-05
insert into test8 values ('120705');    #2012-07-05
insert into test8 values ('19980703');    #1998-07-03
```

3、约束

- [Primary] key
- Auto_increment
- Foreign key
- null/not null,默认允许为null
- unique [key]
- Default

1）主键约束

主键默认非空

1.1）语法

```
#语法一
CREATE TABLE IF NOT EXISTS 表名 (
    字段名称 数据类型 PRIMARY KEY ,
    ...
);

#语法二：省略primary关键字
CREATE TABLE IF NOT EXISTS 表名 (
    字段名称 数据类型 KEY ,
    ...
);

#语法三
CREATE TABLE IF NOT EXISTS 表名 (
    字段名称 数据类型 ,
    ...
    字段名称 数据类型 ,
    PRIMARY KEY(字段名称)
);
```

1.2）案例

```
#1、主键约束
#方法一
CREATE TABLE IF NOT EXISTS user1 (
    id int PRIMARY KEY ,  #主键默认非空
    username varchar(50)
);

#方法二
CREATE TABLE IF NOT EXISTS user1 (
    id int ,        #主键默认非空
    username varchar(50) ,
    PRIMARY KEY(id)
);
```

```
#方法三
CREATE TABLE IF NOT EXISTS user1 (
    id int key,       #省略primary关键字
    username varchar(50)
);

#复合主键--多个字段共同实现主键
CREATE TABLE IF NOT EXISTS user2 (
    id int ,        #主键默认非空
    username varchar(50) ,
    cardNO char(18),
    PRIMARY KEY(id,cardNO)
);
```

说明：

1）注释代码可以使用#或--**（后面加空格）**

2）避免关键字可以使用反引号``

3）通过COMMENT给字段添加注释

4）数据库创建成功后，在MySQL安装目录中的data目录下，创建与数据库名称一样的目录

5）使用不同的存储引擎创建表，则在数据库目录下会产生相关的数据文件。

6）字符串类型的性能：char > varchar > text

2）自动增长

2.1）语法

注意：自动增长的字段必须是主键且数据类型为数值型数据。

```
CREATE TABLE IF NOT EXISTS 表名 (
    字段名称 数值类型 PRIMARY KEY AUTO_INCREMENT ,
    ....
) [AUTO_INCREMENT=数值]; # 设置自动增长起始值
```

2.2）案例

```
#id字段为自动增长，默认的起始值为1，增量为1
CREATE TABLE IF NOT EXISTS user3 (
    id float PRIMARY KEY AUTO_INCREMENT ,     #自动增长必须与主键一起使用
    username varchar(50)
);

#指定自动增长的起始值为100
CREATE TABLE IF NOT EXISTS user3 (
    id int PRIMARY KEY AUTO_INCREMENT ,
```

```
    username varchar(50)
)AUTO_INCREMENT=100;                              #指定自动增长的起始值

#修改自动增长的值
alter table user3 AUTO_INCREMENT=500;

#添加数据
insert into user3 values (111,'zs') ;          #可以指定自动增长字段数据
insert into user3 values (null,'zs') ;         #可以指定为null或default，该字段会自动增长
insert into user3 values (default,'zs') ;
insert into user3(username) values ('zs') ;    #忽略自动增长字段，该字段会自动增长
```

3）空，非空

3.1）语法

默认为NULL

```
#第一种方式：建表时添加约束
#语法一：定义字段的同时，添加约束
CREATE TABLE 表名(
    字段 数据类型 约束  #非空约束为：not null
);

#第二种方式：通过alter语句修改约束
#语法一：
alter table 表名
  modify 字段 数据类型 约束 ;

#语法二：
alter table 表名
  change 原字段 新字段 数据类型 约束 ;
```

3.2）案例

```
# 案例一：指定NULL、NOT NULL约束
CREATE TABLE IF NOT EXISTS user4 (
    id float PRIMARY KEY AUTO_INCREMENT ,
    username varchar(50) not null ,
    phone char(11) null,
    age int  #默认为NULL约束
);
#添加数据--可以使用default关键字表示使用默认值
insert into user4 (username) values ('zs');

# 案例二：各种语法的定义
#方法一：定义字段时添加约束
CREATE TABLE t_user(
    age INT(10) NOT NULL
```

```
);

#方法二：修改字段约束
alter table t_user
  modify age int not null ;

#语法三：修改字段名称的同时，指定约束
alter table t_user
  change age nianLing int not null ;

# 案例三：删除约束
#方法一：
alter table t_user
  modify age int ;

#语法二：
alter table t_user
  change age nianLing;
```

4）唯一

4.1）语法

```
#第一种方式：建表时添加约束
#语法一：
CREATE TABLE 表名(
    字段 数据类型 约束  #唯一约束为：unique [key]
);

#语法二：
CREATE TABLE 表名(
    字段 数据类型 ,
    CONSTRAINT 约束名称 UNIQUE(字段名称1[,字段名称2,...,字段名称n])   #支持复合约束
);

#第二种方式：通过alter语句添加约束
#语法一：
alter table 表名
  modify 字段 数据类型 约束 ;   #唯一约束为：unique [key]

#语法二：
alter table 表名
  change 原字段 新字段 数据类型 约束 ;

#语法三：
ALTER TABLE 表名 ADD UNIQUE [KEY](字段);

#语法四：
ALTER TABLE 表名 ADD CONSTRAINT 约束名称 UNIQUE [KEY](字段);
```

4.2）案例

```
# 案例一：语法的使用
#方法一：
CREATE TABLE t_user(
    user_id int primary key ,
    user_name varchar(50) unique key,
    card char(18) unique        #可以省略key
);

#方法二：
CREATE TABLE t_user(
    user_id int primary key ,
    user_name varchar(50) ,
    card char(18) ,
    CONSTRAINT UN_T_USER_NAME_CARD UNIQUE(user_name,card)     #支持复合约束
);

#方法三：
alter table t_user
  modify user_name char(18) unique ;

#方法四：
alter table t_user
  change user_name user_name char(18) unique ;

#方法五：
ALTER TABLE t_user
  ADD UNIQUE KEY(user_name);                    #key可省略

#方法六：
ALTER TABLE t_user
  ADD CONSTRAINT UN_T_USER_USER_NAME UNIQUE KEY(user_name);   #key可省略


# 案例二：唯一键与主键的区别
CREATE TABLE IF NOT EXISTS user5 (
    id int PRIMARY KEY AUTO_INCREMENT ,
    username varchar(50) unique key,
    cardNo char(18) unique
);

##注意：唯一键 区别 主键
#1）主键在一张表有0或1个，唯一键可以有0或多个
#2）主键、唯一键的字段数据允许为NULL,NULL不属于重复数据。
insert into user5(username,cardNo) values ('zs',null);
insert into user5(username,cardNo) values ('ls',null);
insert into user5(username,cardNo) values (null,'123123');

# 案例三：删除唯一约束
```

```
ALTER TABLE t_user
  DROP INDEX user_id;
```

5）默认

5.1）语法

```
CREATE TABLE IF NOT EXISTS 表名 (
    字段名称 数据类型 default 默认值
);
```

5.2）案例

```
CREATE TABLE IF NOT EXISTS user7 (
    id float PRIMARY KEY AUTO_INCREMENT ,
    username varchar(50) not null ,
    sex ENUM('男','女','保密') default '男' ,
    age int default '18'
);

#添加数据--可以使用default关键字表示使用默认值
insert into user7 (username,sex) values ('zs',default);
insert into user7 (username,sex,age) values ('ls',default,20);

#默认当前时间：now() 或 CURRENT_TIMESTAMP
```

6）检查

6.1）语法

```
# 语法一
CREATE TABLE IF NOT EXISTS 表名 (
    字段名称 数据类型 check(表达式)
);

# 语法二
CREATE TABLE IF NOT EXISTS 表名 (
    字段名称 数据类型 ,
    check(表达式)
);

# 语法三
CREATE TABLE IF NOT EXISTS 表名 (
    字段名称 数据类型
);
ALTER TABLE 表名 ADD CONSTRAINT 检查约束名 CHECK(表达式) ;
```

```
# 删除检查约束
ALTER TABLE 表名 DROP CONSTRAINT 检查约束名;
```

6.2）案例

```
# 方法一
CREATE TABLE IF NOT EXISTS user8 (
    id int PRIMARY KEY AUTO_INCREMENT ,
    username varchar(50) unique key,
    age int check(age>18)
);

# 方法二
CREATE TABLE IF NOT EXISTS user8 (
    id int PRIMARY KEY AUTO_INCREMENT ,
    username varchar(50) unique key,
    age int ,
    check(age>18)
);

-- 虽然设置了check约束，但能成功添加数据
insert into user8(username,age) values ('zs',12);
```

注意：5.7版本之前会忽略check约束，8.0实现

解决方法一：使用枚举类型

```
CREATE TABLE IF NOT EXISTS user8 (
    id int PRIMARY KEY AUTO_INCREMENT ,
    username varchar(50) unique key,
    sex enum('男','女') NOT NULL DEFAULT '男'
);
```

解决方法二：使用触发器

```
# 创建数据表
CREATE TABLE IF NOT EXISTS stu (
    id int PRIMARY KEY AUTO_INCREMENT ,
    username varchar(50) unique key,
    age int
);

# 定义触发器
DELIMITER $
create trigger tri_stu_age before insert on stu for each row
begin
  if new.age<0 or new.age>100 then set new.age=20;
  end if;
end $
```

```
DELIMITER ;
```

解决方法三：升级到8.0版本

7）外键约束

https://dev.mysql.com/doc/refman/5.7/en/create-table-foreign-keys.html

外键的使用条件

- 两个表必须是InnoDB表，MyISAM表暂时不支持外键
- 外键列必须建立了索引，MySQL 4.1.2以后的版本在建立外键时会自动创建索引，但如果在较早的版本则需要显式建立；
- 外键关系的两个表的列必须是数据类型相似，也就是可以相互转换类型的列，比如int和tinyint可以，而int和char则不可以；

外键的好处

可以使得两张表关联，保证数据的一致性和实现一些级联操作。

7.1）语法

```
# 语法一
CREATE TABLE 表名 (
    字段名称 数值型类型 ,
    ...
    字段名称 数值型类型 ,
    CONSTRAINT 外键名称 FOREIGN KEY 外键字段名称 REFERENCES 主键表名称 主键字段名称
    [ON DELETE reference_option]
    [ON UPDATE reference_option]
);

# 语法二
ALTER TABLE 表名 ADD CONSTRAINT 外键名称
  FOREIGN KEY(外键字段名称) REFERENCES 主键表名(主键字段名);

# 删除外键约束
ALTER TABLE 表名 DROP FOREIGN KEY 外键约束名;
```

reference_option：RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

1. RESTRICT：限制外表中的外键改动（默认）
2. CASCADE：跟随外键改动
3. SET NULL：设空值
4. NO ACTION：无动作（默认）
5. SET DEFAULT：设默认值

7.2）案例

```
# 部门信息表（主键表）
CREATE TABLE tbl_dept
(
    id INT(11) PRIMARY KEY,      # 部门编号
    name VARCHAR(22) NOT NULL,   # 部门名称
    location VARCHAR(50)         # 部门位置
);

# 语法一
# 员工信息表（外键表）
CREATE TABLE tbl_emp
(
    id INT(11) PRIMARY KEY,
    name VARCHAR(25),
    deptId INT(11),
    salary FLOAT,
    CONSTRAINT fk_emp_dept FOREIGN KEY(deptId) REFERENCES tbl_dept(id)
);


# 语法二
CREATE TABLE tbl_emp
(
    id INT(11) PRIMARY KEY,
    name VARCHAR(25),
    deptId INT(11),
    salary FLOAT,
);

ALTER TABLE tbl_emp ADD CONSTRAINT fk_emp_dept
  FOREIGN KEY(deptId) REFERENCES tbl_dept(id)

# 删除外键约束
ALTER TABLE tbl_emp DROP FOREIGN KEY fk_emp_dept;
```

8）UNSIGNED（无符号位）

8.1）语法

无符号数值（unsigned），针对数值型数据，默认为有符号的。

```
CREATE TABLE 表名 (
    字段名称 数值型类型 unsigned,     #无符号的数值（正数）
    字段名称 数值型类型               #默认是有符号的
);
```

8.2）案例

```
#案例：无符号数值(unsigned) => 针对数值型数据
CREATE TABLE test1 (
    num1 tinyint unsigned,    #无符号的tinyint
    num2 tinyint              #默认是有符号的
);

insert into test1 values (-12,-12);   #错误
insert into test1 values (0,-12);     #正确
```

9）ZEROFILL（零填充）

9.1）语法

零填充（zerofill）也是针对数值型数据，不够指定位数，则使用0填充。

```
CREATE TABLE 表名 (
    字段名称 数值型类型 zerofill
);
```

9.2）案例

```
CREATE TABLE test2 (
    num1 tinyint zerofill,
    num2 smallint zerofill,
    num3 mediumint zerofill,
    num4 int zerofill,
    num5 bigint zerofill
);

#查看各数值型数据默认所占位数(注意：当指定zerofill时，同时也被指定为unsigned)
desc test2;

#添加数据
insert into test2 values (1,1,1,1,1);
insert into test2 values (123,1,1,1,1);
```

五、查看表结构

```
#方法一：desc 表名;
desc student;

#方法二：describe 表名 ;
describe student;

#方法三：show columns from 表名;
show columns from cms_cate;

#Extra：表示额外的信息
```

六、删除表

```
drop table [if exists] 数据表名;
```

七、修改表结构

1、修改表名

```
#语法一：
alter table 表名
  rename [to|as] 新表名;

#语法二：
rename table 表名 to 新表名;
```

2、添加字段

```
alter table 表名
  add 字段名称 数据类型 [约束] [first|after 字段名称];
```

3、删除字段

```
alter table 表名
  drop 字段名称 ;
```

4、修改字段

```
alter table 表名
  modify 字段名称 数据类型 [约束] [first | after 字段名称]
```

5、修改字段名称

```
alter table 表名
  change 旧字段名称 新字段名称 数据类型 [约束] [first|after 字段名称]
```

6、添加默认值

```
alter table 表名
  alter 字段名称 set default 默认值;
```

7、删除默认值

```
alter table 表名
  alter 字段名称 drop default;
```

8、修改表的存储引擎

```
alter table 表名
  engine=存储引擎名称;

# 查看
SHOW CREATE TABLE 表名 \G;

# 注意：此操作只是修改某张表的存储引擎，如果希望修改默认的存储引擎，则可修改my.ini配置文件
# default-storage-engine=存储引擎名称
```

9、设置自增长的值

```
alter table 表名
  AUTO_INCREMENT=值;
```

八、数据表基本操作

1、添加

```
# 全字段添加
INSERT INTO 表名 values(值1,...,值n)

# 局部字段添加
INSERT INTO 表名(字段1,...,字段N) VALUES (值1,...,值n)

# 批量添加
INSERT INTO 表名 VALUES (值1, 值2, ...), (值1, 值2, ...), (值1, 值2, ...);
INSERT INTO 表名 (字段名1, 字段名2, ...) VALUES (值1, 值2, ...), (值1, 值2, ...), (值1, 值2, ...)
```

注意：

- 局部添加数据，并保证没添加数据字段的约束允许为空
- 字符串和日期类型数据应该包含在引号中

2、删除

```
-- 语法一
delete from 表名 [where 条件]

-- 语法二
truncate table 表名
```

注意：

- 删除一般要写条件，否则会把整张表的数据都删除了
- 局部删除数据，小慎用,一般where条件使用的字段是唯一的（主键,唯一键）
- delete VS truncate
    - truncate不能加where条件，用于清空表格数据；而delete可以加where条件，tr根据需求删除数据；
    - truncate效率高一些
    - truncate 删除带自增长的列的表后，如果再插入数据，数据从1开始；delete 删除带自增长列的表后，如果再插入数据，数据从上一次的断点处开始
    - truncate 删除数据不能回滚，delete删除可以回滚

3、修改

```
update 表名 set
    字段1='新值',
    字段2='新值',
    ...
    字段n='新值'
where 条件
```

注意：

- 修改一般要写条件，否则会把整张表都修改了
- 一般where条件使用的字段是唯一的(主键,唯一键)
- where前不允许加逗号