day01

Mysql 选择5.6.48版本

解压完成后，在bin目录下

```
服务端 mysqld.exe

客户端 mysql.exe
```

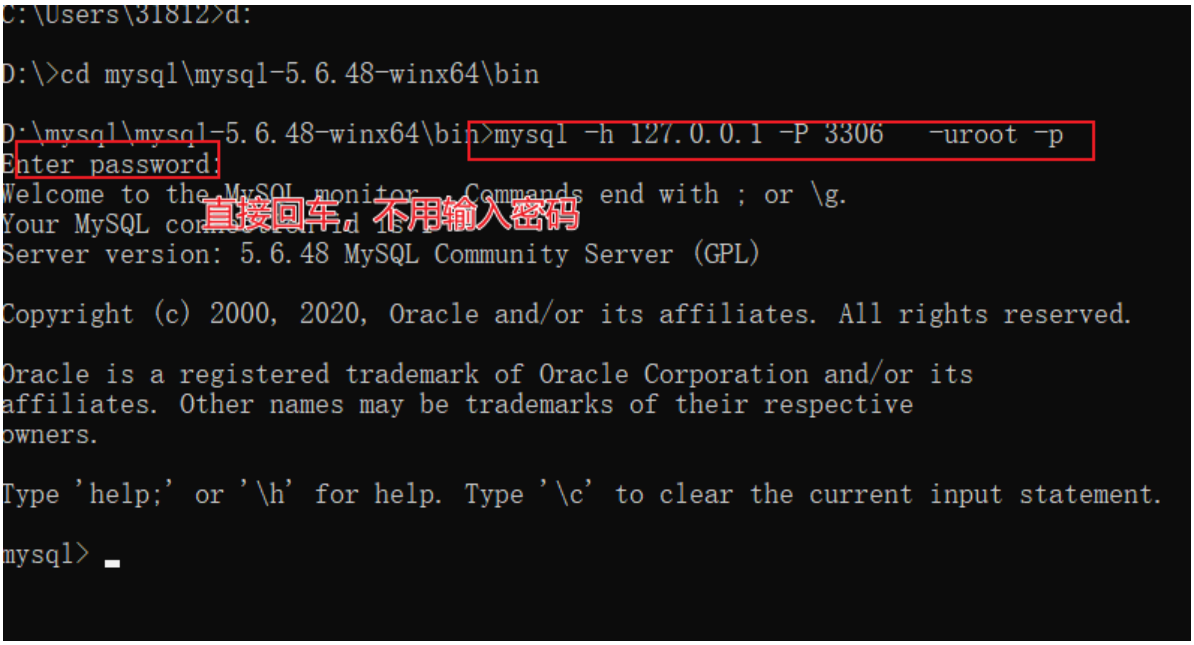
注意:

```
配置MySQL，终端以管理员身份运行
```

启动服务端:

- 切换到mysqld所在的bin目录下，然后终端输入mysqld，回车
- 保留cmd窗口重新打开一个cmd作为客户端,然后切换到bin目录下，输入mysql -h 127.0.0.1 -uroot -p
- 刚开始是没有密码的，直接回车就可

```
常用软件默认端口号
Mysql 3306
redis 6379
mongodb 27017
django 8000
flask 5000
```



关系型数据库

mongodb是最像关系型的非关系型数据库

- 数据彼此之间有约束或者联系
- 存储数据一般都是以表的形式

非关系型数据库

- 通常以key value键值对的形式存储数据

SQL语句初始

1. Mysql第一次以管理员身份进入是没有密码的，直接回车即可
2. 基本命令：

```
1 .show databases;  查看所有的数据库库名
2 .客户端连接服务端命令：
mysql -h 127.0.0.1 -P 3306   -uroot -p
可以简写成  mysql    -uroot -p
3.当输入的命令不对，又不想让服务器执行并返回报错信息，可以用 \c 取消
4.客户端退出：  退出命令加不加分号都可以
        quit
         exit
5.当在连接服务端的时候，发现只输入mysql的时候也能连接 ，但是不是管理员身份，只是一个游客模式
```

环境变量配置及系统服务制作

知识补充

```
1.查看当前具体进程
tasklist
tasklist |findstr mysqld
2.杀死具体进程 （在管理员cmd窗口才可以）
taskkill /F /PID PID号
```

环境变量配置

- 每次启动mysqld需要先切到对应的文件路径下才可以，这样就太繁琐，
- 可以把bin 目录添加到系统变量Path下
- 每次都需要启用两个cmd窗口，我们可以将mysql服务端制作成系统服务（开机自动启用）

```
查看当前计算机的运行进程数 services.msc
"""将musqld制作为系统服务，mysqld --install"""
将mysqld移除系统服务mysqld --remove
```

设置密码

```
mysqladmin -uroot -p原密码 password 新密码
直接在终端输入就可以

```

忘记密码重置密码

```
可以将mysql获取用户名和密码校验功能看成是一个装饰器，
装饰在了客户端请求访问的功能上

#1.关闭当前mysql服务端
命令行方式启动（让mysql跳过用户名密码验证功能）
mysqld --skip-grant-tables
#2.直接以无密码的方式连接
mysql -uroot -p 直接回车
#3.修改当前用户的密码
update mysql.user set password=password(123456) where user='root' and
host='localhost';
"""
真正存储用户表的密码字段  存储的是密文
只有用户知道明文是什么，其他人不知道，这样更加安全
密码比对也只能比对密文
"""
#4. 立刻将修改数据刷到硬盘
flush privileges;
#5.关闭当前服务端，然后以正常校验授权表的形式启动
```

统一编码

```
mysql> \s
--------------
mysql  Ver 14.14 Distrib 5.6.48, for Win64 (x86_64)

Connection id:          4
Current database:
Current user:           ODBC@localhost
SSL:                    Not in use
Using delimiter:        ;
Server version:         5.6.48 MySQL Community Server (GPL)
Protocol version:       10
Connection:             localhost via TCP/IP
Server characterset:    latin1
Db     characterset:    latin1
Client characterset:    gbk
Conn.  characterset:    gbk
TCP port:               3306
Uptime:                 23 min 5 sec

Threads: 1  Questions: 15  Slow queries: 0  Opens: 67  Flush tables: 1  Open tables: 60
--------------

mysql> \
```

mysql默认配置文件

```
my-default.ini
ini结尾的一般都是配置文件
程序启动会先加载配置文件中的配置后才真正的启动

[mysqld] #一旦服务器启动立刻加载下面的配置
sql_mode=NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES
[client]#其他客户端
```

需要自己创建一个my.ini的配置文件
验证配置是否自动加载

```
[mysql]
print('hello world')
#修改配置文件后一定要重启服务才生效
```

统一编码

在新创建的my.ini配置文件中重新写入

```
[mysqld]
character-set-server=utf8
collation-server=utf8_general_ci
[client]
default-character-set=utf8
[mysql]
default-character-set=utf8
```

将管理员的用户名和密码添加到配置文件中

```
#这样不用每次都输入账号密码了
[mysqld]
character-set-server=utf8
collation-server=utf8_general_ci
[client]
default-character-set=utf8
[mysql]
user="root"
password="123.com"
default-character-set=utf8

```

```
mysql> \s
--------------
mysql  Ver 14.14 Distrib 5.6.48, for Win64 (x86_64)

Connection id:          1
Current database:
Current user:           root@localhost
SSL:                    Not in use
Using delimiter:        ;
Server version:         5.6.48 MySQL Community Server (GPL)
Protocol version:       10
Connection:             localhost via TCP/IP
Server characterset:    utf8
Db     characterset:    utf8
Client characterset:    utf8
Conn.  characterset:    utf8
TCP port:               3306
Uptime:                 22 sec

Threads: 1  Questions: 5  Slow queries: 0  Opens: 67  Flush tables: 1  Open tables: 60  Queries per second
--------------
```

增删改查

库的增删改查

```
#增
create database 数据库名;
create database db2 charset='gbk';
#查
show databases; #查所有
show create database db1; #查单个
#改
alter database db1 charset='utf8';
#删
drop database db1;
```

表的增删改查

```
"""
操作表（文件）的时候要指定 所在的库（文件夹） 使用use 数据库名    切换到用操作表的数据库，
"""
#查看当前所在的库
select database();
#切换库
use db1;

#增
create table t1(id int,name char(4));
#查
show tables; #查看当前库下面所有的表
show  create table t1; #查看单个表
describe t1; #支持简写 desc t1;
#改
alter table t1 modify name char(16);

#删
drop table t1;

"""
就是在不同的数据库也可以对另一个数据库操作
createa table db2.t1(id int); 也可以使用绝对路径的形式操作不同的库
"""
```

数据的增删改查

```
"""
一定要先有库，再有表，才能有数据
"""

#增
insert into t1 values(1,"json");
insert into t1 values(1,"json"),(2,"egon"),(3,"tank");
#查
select * from  t1; #查t1里所有的，但数据量特别大的时候不建议使用
select id,name from t1; #查id name 这两个字段
select name from t1;# 查name字段
select * from t1\G;  #当表字段特多，展示错乱的时候，可以使用\G分行展示
```

```
#改
update t1 set name="DSB" where id >1;

#删
delete from t1 where id>1;
delete from t1 where id=2;
delete from t1 where name="json";
#清空表的所有数据
delete from
```

存储引擎

针对不同的数据应该有不同的处理机制来存储

存储引擎就是不同的处理机制

MySQL主要的存储引擎

- **innodb** :MySQL5.5八版本及之后的默认的存储引擎
  存储数据更加的安全
- **Myisam**:MySQL5.5之前默认的存储引擎
  速度比innodb更快
- **memory**：内存引擎（数据全部存放在内存中）断电数据丢失
- **blackhole**：无论存什么都立刻消失

```
"""
#查看所有的存储引擎
show engines;

#不同的存储引擎在存储表的时候，异同点：
create table t1(id int) engine=innodb;
create table t2(id int) engine=myisam;
create table t3(id int) engine=blackhole;
create table t4(id int) engine=memory;

#存数据
insert into t1 values(1);
insert into t2 values(1);
insert into t3 values(1);
insert into t4 values(1);
"""
```

```
mysql> show engines;
+--------------------+---------+----------------------------------------------------------------+--------------+------+------+
| Engine             | Support | Comment                                                        | Transactions | XA   | Savep
s |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------+
| FEDERATED          | NO      | Federated MySQL storage engine                                 | NULL         | NULL | NULL
|
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables                          | NO           | NO   | NO
|
| MyISAM             | YES     | MyISAM storage engine                                          | NO           | NO   | NO
|
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears) | NO           | NO   | NO
|
| CSV                | YES     | CSV storage engine                                             | NO           | NO   | NO
|
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables      | NO           | NO   | NO
|
| ARCHIVE            | YES     | Archive storage engine                                         | NO           | NO   | NO
|
| InnoDB             | DEFAULT | Supports transactions, row-level locking, and foreign keys     | YES          | YES  | YES
|
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                             | NO           | NO   | NO
|
+--------------------+---------+----------------------------------------------------------------+--------------+------+------+
9 rows in set (0.00 sec)
```

支持事务 行锁 外键

不同存储引擎异同点:

创建表的完整语法

```
"""
#语法
create table 表名(
    字段名1 类型(宽度) 约束条件,
    字段名2 类型(宽度) 约束条件,
    字段名3 类型(宽度) 约束条件
)

"""

#注意:
1 同一张表中，字段名不能重复
2 宽度和约束条件是可选的,而字段名和字段类型是必须要写的
    约束条件写的话，也支持写多个
    字段名1 类型(宽度) 约束条件1 约束条件2 约束条件3,
    create table t5(id);报错
3 最后一个字段后面不能有逗号
    create table t6(
    id int,
    name char,
);  报错

"""补充"""
```

```
#宽度：
    一般情况下指定是对存储数据的限制
    create table t7(name char); 默认宽度是1
    insert into t7 values('Json'); 只会存储J,其他字符不会存储
    针对不同版本会出现不同的效果
    5.6版本默认没有开启严格模式，规定只能存一个字符，你给了多个字符，那么会自动帮你截取
    5.7版本及以上或者开启了严格模式，那么规定了几个，就不能超，一旦超出范围就立刻报错

"""严格模式到底开不开"""
MySQL5.7之后的版本默认都是开启严格模式的
"""使用数据库的准则"""
能尽量少的让数据库干活就尽量少，不要给数据库增加额外的压力
"""约束条件  not null不能插入null"""
create table t8(id int ,name char not null);
insert into t8 value(1,"json");不报错，正常
insert into t8 value(2,null);报错
"""
宽度和约束条件的关系：
    宽度是用来限制数据的存储
    约束条件是在宽度的基础上增加额外的约束

"""
```

MySQL基本数据类型

整形

1. 分类：TINYINT SMALLINT MEDUIMINT INT BINGINT
2. 作用：存储年龄，等级，id 号码 。。。等等

```
以TINYINT
    默认情况下是带符号的
    超出只存最大值
create table t9(id tinyint);
insert into values(-129),(256);
#约束条件之无符号
create table t10 (id  tinyint unsigned);
insert into t10 values(-1),(256);
```

```
create table t11(id int);
insert into  t11 values(-1,256);
#int默认情况下是带符号的
#整形默认情况下都是带符号的
```

```
"""针对整形 括号内的宽度到底是干嘛的"""

create table t12(id int(8));
insert into t12 values(123467489);
"""特例：只有整数括号内的数字不是表示限制位数
id int(8)
    如果数字没有查出8位，那么默认用空格填充至8位
    如果超出8位，那么有几位就存几位（但是要遵循最大范围）
"""

#用零填充至8位
create table t13（id int(8) unsigned zerofill);

```

整型总结:

针对整形字段，括号内**无需指定宽度**，因为默认的宽度已经足够显示所有的数据了

严格模式

```
ysql> show variables like '%mode%';
+--------------------------------+------------------------+
| Variable_name                  | Value                  |
+--------------------------------+------------------------+
| binlogging_impossible_mode     | IGNORE_ERROR           |
| block_encryption_mode          | aes-128-ecb            |
| gtid_mode                      | OFF                    |
| innodb_autoinc_lock_mode       | 1                      |
| innodb_strict_mode             | OFF                    |
| pseudo_slave_mode              | OFF                    |
| slave_exec_mode                | STRICT                 |
| sql_mode                       | NO_ENGINE_SUBSTITUTION |
+--------------------------------+------------------------+
8 rows in set (0.00 sec)
```

```
"""如何查看严格模式"""
show variables like "%mode";

#模糊匹配/查询
    关键字  like
        %:匹配任意多个字符
        _: 匹配任意单个字符

#修改严格模式：
    set session  只在当前窗口有效
    set global  全局有效

    set global sql_mode="STRICT_TRANS_TABLES";
    修改完之后，重新进入服务端即可

```

重启sql服务后，在查看严格模式，就成了修改后的样子

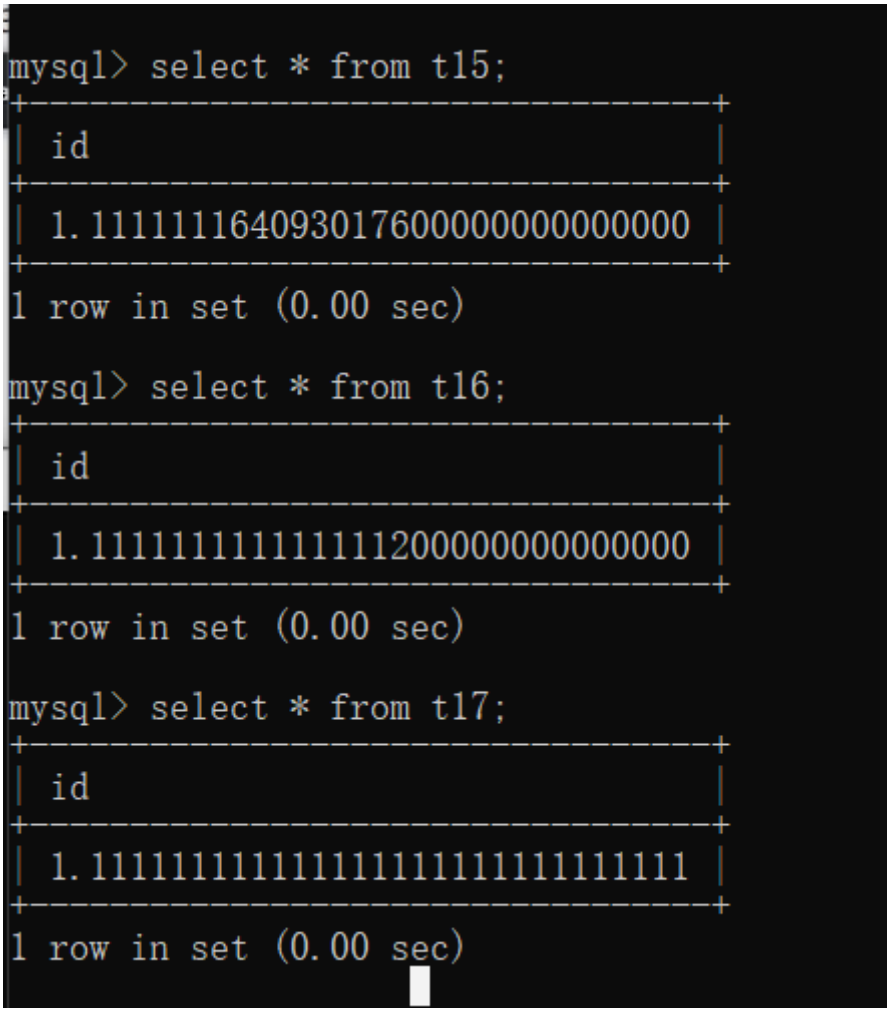
```
mysql> show variables like '%mode%';
+--------------------------------+----------------------+
| Variable_name                  | Value                |
+--------------------------------+----------------------+
| binlogging_impossible_mode     | IGNORE_ERROR         |
| block_encryption_mode          | aes-128-ecb          |
| gtid_mode                      | OFF                  |
| innodb_autoinc_lock_mode       | 1                    |
| innodb_strict_mode             | OFF                  |
| pseudo_slave_mode              | OFF                  |
| slave_exec_mode                | STRICT               |
| sql_mode                       | STRICT_TRANS_TABLES  |
+--------------------------------+----------------------+
8 rows in set (0.00 sec)
```

浮点型

1. 分类：FLOAT DOUBLE DECIMAL
2. 作用：身高，体重，薪资 ..... 等等

```
"""存储限制"""
float(255,30);#总共255位，小数部分占30位
double(255,30);#总共255位，小数占30位
decimal(65,30);#总共65位，小数占30位
```

```
"""精确度验证"""
create table t15(id float(255,30));
create table t16(id double(255,30));
create table t17(id decimal(65,30));

insert into t15 values(1.111111111111111111111111111114111);
insert into t16 values(1.111111111111111111111111111114111);
insert into t17 values(1.111111111111111111111111111114111);

"""float < double < decimal"""
```



字符类型

1. 分类：char (定长)　　varchar（变长)

```
"""
char(4)超过4个字符，直接报错，不够4个空格补全
varchar(4)超过4个字符，直接报错，不够4个字符有几个存几个
"""
```

```
create table t18(name char(4));
create table t19(name varchar(4));

insert into t18 values("a");
insert into t19 values("a");

"""小方法： char_length 统计字段长度"""

select char_length(name) from t18;
select char_length(name) from t19;
"""首先可以肯定char在硬盘上存的绝对是真正的数据，带有空格的
但是，在显示的时候MySQL会自动将多余的空格剔除"""
#在次修改sql_mode 让MySQL不要自动剔除操作
set global sql_mode="STRICT_TRANS_TABLES,PAD_CHAR_TO_FULL_LENGTH";
```

```
"""char 和varchar对比"""
char
```

```
        缺点：浪费空间
        优点：  直接按照固定的字符存取数据即可
        jason segon alex wusir tank
        存直接按照5个字符存，取也直接按照五个字符存取
varchar:
        优点：节省空间
        缺点：存取较为麻烦
        1bytes+jason 1bytes+segon 1bytes+alex 1bytes+wusir 1bytes+tank
        存的时候需要制作报头
        取的时候也需要先读取报头，然后才读取真正的数据

以前基本用的char 现在用的varchar也挺多

```

时间类型

1. 分类：date(年月日)datetime（年月日时分秒） time(时分秒) Year （年）

```
create table student(
    id int,
     name varchar(16),
    born_year year,
    birth date,
    study_time time,
    reg_time datetime
);
insert into student values(1,"zSong","2010","2000-11-11","11:11:11","2022-08-08 11:11:11");
```

枚举与集合类型

1. 分类

```
"""
枚举 (enum) 多选一
集合（set） 多选多
"""
```

2. 使用

```
create table user(
    id int,
    name char(16),
    gender enum('male','female','others')

);
insert into user values(1,"json","male");
insert into user values(2,"agon","xxxxxxooo");报错
#枚举字段，在存数据的时候只能从枚举里面选择一个存储
```

day02

约束条件

```
"""
约束条件
zerofill
unsigned
not null
"""
```

default默认值

```
"""知识点补充"""
create table t1(
    id int ,
    name char(16)
);
insert into t1(name,id) values("json",1);

create table t2(
    id int ,
    name char(16),
    gender enum('male','female','others') default 'male'
);
insert into t2(id,name) vaules(1,'json');
insert into t2 values(2,'egon','female');
```

unique 唯一

```
"""
#单列唯一:
create  table t3(
    id int unique,
    name char(16)
);
insert into t3 values(1,'zhao'),(1,'kang');报错，id唯一，不能重复
insert into t3 values(1,'zhao'),(2,'kang');
#联合唯一
create table t4(
    id int,
    ip char(16),
    port int,
    unique(ip,port)
);
isnert into t4 values(1,'127.0.0.1',8080);
insert into t4 values(2,'127.0.0.1',8081);
insert into t4 values(3,'127.0.0.2',8080);
insert into t4 values(4,'127.0.0.1',8080);报错 与第一个ip和port完全重复


"""
```

primary key主键

```
从约束效果上，"""primary key 等价于not null + unique"""
"""非空且唯一！！！"""
create table t5(id int primary key);
insert into t5 values(null);报错，不能为空
insert into t5 values(1),(1);报错，不能重复
insert into t5 vaules(1)(2);

"""
除了有约束效果之外，它还是innodb存储引擎组织数据的依据
innodb存储引擎在创建表的时候必须要有primary key
因为它类似于书的目录，能够帮助提示查询效率并且也是建立表的依据

"""
# 1. 一张表中有且只有一个主键，如果没有设置主键，那么会从上往下搜索直到遇到一个非空且唯一的字段将它自动提升为主键
create table t6(
    id int,
    name char(16),
    age int not null unique,
    addr char(32) not null unique
);
desc t6;


# 2. 如果表中没有主键也没有其他任何的非空且唯一的字段，
#那么innodb会采用自己内部提供一个隐藏字段作为主键，
#隐藏意味着你无法使用到它，就无法显示查询速度

#3. 一张表中通常都应该有一个主键字段，并且通常将id /uid/sid字段作为主键
"""#单个字段主键"""
create table t6(
    id int primary key,
    name char(16)
);
"""联合主键(多个字段联合起来作为表的主键，本质上还是一个主键)"""
create table t7(
    id int,
    ip char(16),
    port int,
    primary key(ip,port)
);
```

```
mysql> create table t6(id int, name char(16),age int not null unique,addr char(32) not nul
nique);
Query OK, 0 rows affected (0.39 sec)

mysql> desc t6;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| id    | int(11)  | YES  |     | NULL    |       |
| name  | char(16) | YES  |     | NULL    |       |
| age   | int(11)  | NO   | PRI | NULL    |       |
| addr  | char(32) | NO   | UNI | NULL    |       |
+-------+----------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

也就意味着在创建表的时候id字段一定要加primary key

auto_increment自增

```
当编号特别多个时候，人为的去维护太麻烦，
create table t7(
    id int primary key auto_increment,
    name char(16)
);
insert into t8(name) values('jsson'),('lisi'),('zhaosng');
#auto_increment只能加在主键上，不能给普通字段加
```

结论

```
以后在创建表的id(数据的唯一标识id/uid/sid)字段的时候，要加上主键，自增
id int primary key auto_increment
```

补充

```
delete from  t7在删除表中数据的时候，主键的自增不会停止
truncate t7 清空表中数据并重置主键
```

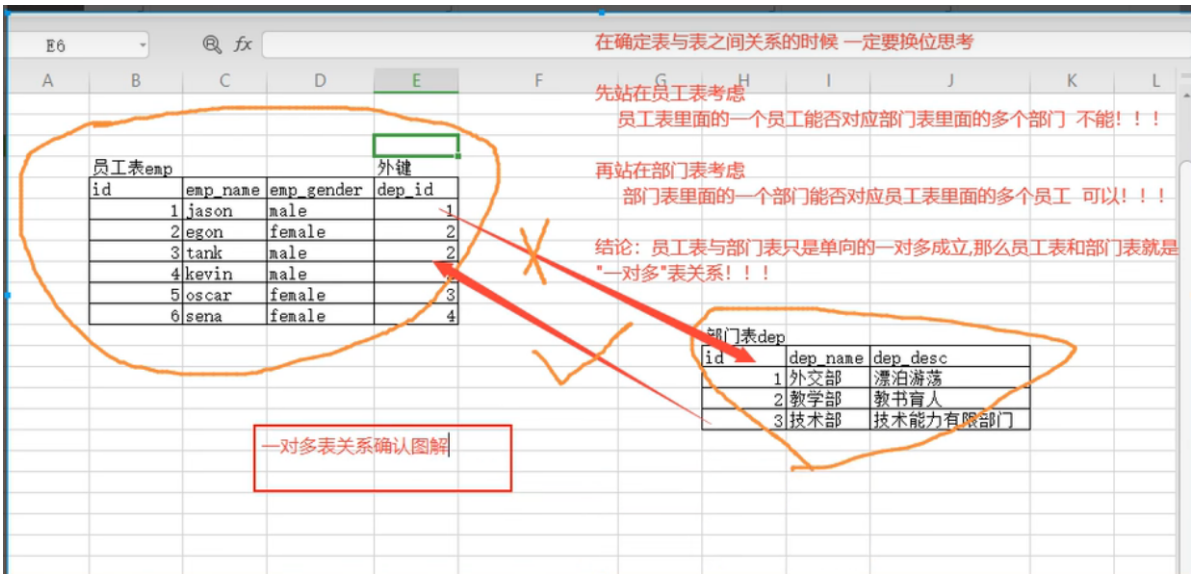
外键

```
"""
外键就是用来帮助我么建立表与表之间的关系
foregin key
"""
```

表关系

```
"""
表与表之间最多只有四种关系
    一对多关系
        在MySQL中没有多对一这个概念
        一对多，多对一，都叫一对多！！
    多对多关系
    一对一关系
    没有关系
"""
```

一对多表关系




```
foreign key
    1. 一对多表关系，外键字段建在多的一方
    2. 创建表的时候一定要先建立被关联表
    3. 录入数据的时候必须先录入被关联的表

```

```sql
#SQL语句建立表关系
create table dep(
    id int primary key auto_increment,
    dep_name char(16),
    dep_desc char(32)
);
create table emp(
    id int primary key auto_increment,
    name char(16),
    gender enum('male','femla','others') default 'male',
    dep_id int,
    foreign key(dep_id) references dep(id)
);
insert into dep(dep_name,dep_desc) values('教学部','教书育人'),('IT部','技术过硬'),('外交部','多人外交');
insert into emp(name,dep_id) values("jsal",2),('kangyiming',1);
```

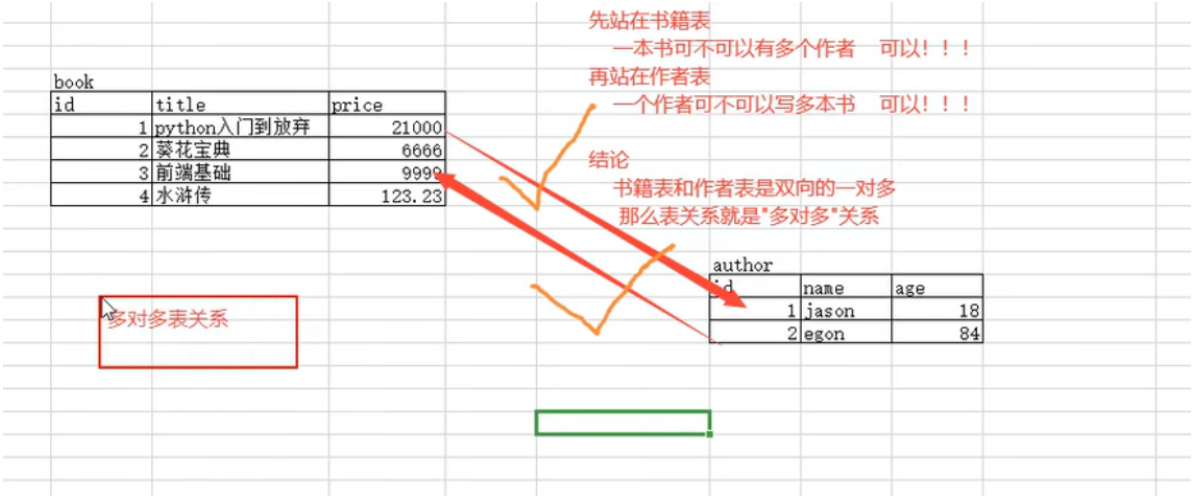
```
"""
#修改emp里面的dep_id字段或者dep表里面id字段
update dep set id=200 whereid=2; 报错
#删除dep里面的数据
delete from dep;  报错
"""
```

怎么修改怎么删除呢？请往下看

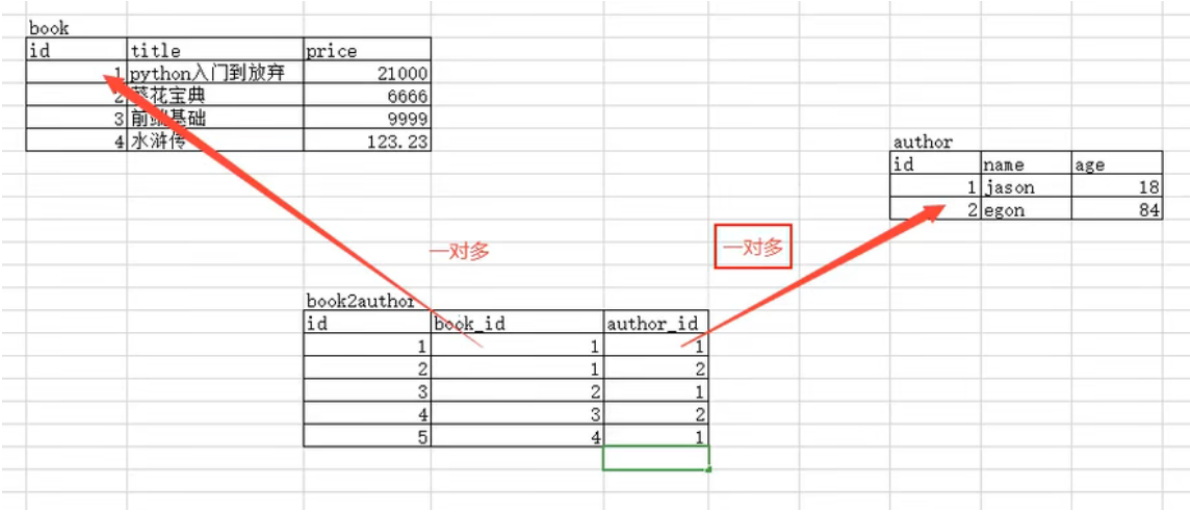
```
更新就同步更新，删除就同步删除
"""
级联更新   级联删除
```

```
"""
create table dep(
    id int primary key auto_increment,
    dep_name char(16),
    dep_desc char(32)
);
create table emp(
    id int primary key auto_increment,
    name char(16),
    gender enum('male','femla','others') default 'male',
    dep_id int,
    foreign key(dep_id) references dep(id)
    on update cascade #同步更新
    on delete cascade #同步删除
);

```

多对多表关系




```
"""图书表 和作者表"""
create table book(
    id int primary key auto_increment,
    title varchar(32),
    price int,
    author_id int,
    foreign key(author_id) references author(id)
    on update cascade
    on delete cascade
);报错
create table author(
    id int primary key auto_increment,
    name varchar(32),
    age int,
    book_id int,
    foreign key(book_id) references author(id)
    on update cascade
    on delete cascade
);报错
"""针对多对多的表关系，不能在两张原有的表中创建外键
需要单独开一个新表，专门用来存储两张表数据之间的关系
"""
```




```sql
create table book(
    id int primary key auto_increment,
    title varchar(32),
    price int
);
create table author(
    id int primary key auto_increment,
    name varchar(32),
    age int
);
insert into book(title,price) values('三国演义',666),('西游记',124),('红楼梦',47);
insert into author(name,age) values('kangyiming',18),('zsong',22);

create table book_author(
    id int primary key auto_increment,
    author_id int,
    book_id int,
    foreign key(author_id) references author(id)
    on update cascade
    on delete cascade,
    foreign key(book_id) references book(id)
    on update cascade
    on delete cascade

);
insert into book_author(author_id,book_id) values(1,1),(1,2),(2,3);
```

一对一表关系

```sql
"""
一对一外键字段建在任意一方都可以，但是推荐建在查询频率较高的表中

"""
create table author_detail(
    id int primary key auto_increment,
    phone int,
    addr varchar(64)
);
```

```sql
create table author(
    id int primary key auto_increment,
    name varchar(32),
    age int ,
    author_detail_id int unique,
    foreign key(author_detail_id) references author_detail(id)
    on update cascade
    on delete cascade
);
```

总结

```
表关系的建立需要用到foreign key
一对多：外键字段建在多的一方
多对多：自己开设第三张表来存储
一对一：建在任意一方都可以，推荐建在查村频率多的表中
```

补充

```
"""
表与表之间如果有关系的话，可以有两种建立联系的方式
1. 就是通过外键强制性的建立关系
2. 就是自己通过sql语句逻辑层面上建立关系
    delete from emp where id=1;
    delete from dep where id=1;

创建外键会消耗一定的资源，并且增加了表与表之间的耦合度
在实际项目中，如果表特别多，其实可以不做任何外键处理，直接通过sql语句来建立逻辑层面上的关系

"""
```

修改表

```sql
#MySQL对大小写不敏感的。不区分大小写

"""
1. 修改表名
alter table 表名 rename 新表明;
2.增加字段
alter table 表名 add 字段名 字段类型(宽度) 约束条件;
alter table 表名 add 字段名 字段类型(宽度) 约束条件 first;
alter table 表名 add 字段名 字段类型(宽度) 约束条件 after 字段名;

3.删除字段
    alter table 表名 drop 字段名;
4. 修改字段
alter table 表名 modify 字段名 字段类型(宽度) 约束条件;
alter table 表名 change 旧字段名 新字段名 字段类型(宽度) 约束条件;

"""
```

复制表

```
"""
sql语句的结果其实也是一张虚拟表
言外之意就是针对这个查询结果还可以继续用查询表的语法继续操作该虚拟表

"""
create table 新表名 select * from 旧表名;  #不能复制主键，外键...
create table 新表名 select * from 旧表名 where id>3;
#只能复制数据
```

如何查询表

```
"""
select
where
group by
having
distinct
order by
limit
regexp
like

"""
```

提前准备表

```
create table emp(
    id int primary key auto_increment,
    name varchar(32) not null,
    sex enum('male','female') not null default 'male',
    age  int(3) unsigned not null default 23,
    hire_date date not null,
    post varchar(50),
    post_comment varchar(100),
    salary double(15,2),
    office int,    #一个部门一个屋子
    depart_id int
);
"""插入记录"""
#三个部门：教学，销售，运营
insert into emp(name,sex,age,hire_date,post,salary,office,depart_id) values
('song','male',32,'19990911','teacher',10000000,401,1),
('zhang','male',27,'20100708','teacher',80000,401,1),
('san','female',42,'20070401','teacher',740000,401,1),
('lisi','female',29,'201503016','teacher',14000,401,1),
('wangwu','male',22,'20000311','teacher',67000,401,1),
('哈哈','male',29,'20170311','sale',68000.94,402,2),
('喜喜','male',20,'20081111','sale',780000.74,402,2),
('蛇姐','female',23,'20140111','sale',17000.09,402,2),
('二二','female',21,'20011212','sale',670000.44,402,2),
('喜之郎','female',24,'20061012','sale',190000.38,402,2),
```

```
('傻逼','male',21,'20170311','operation',48000,403,3),
('喜哈','male',20,'20051111','operation',78000,403,3),
('裸李','female',19,'20040112','operation',79000,403,3),
('三八','male',20,'20011201','operation',60000,403,3),
('程咬金','male',21,'20060112','operation',195000,403,3),
('罗丽莉','female',18,'20001201','operation',650000,403,3),
('青霞','female',17,'20040112','operation',1950500,403,3);
```

```
#当表字段特别多的时候，展示会错乱，可以是用\G分行展示
select * from emp\G
#如果在插入中文的时候出现乱码，或者空白的现象，可以将字符编码统一设置为GBK
```

几个重要关键字的执行顺序

```
"""书写顺序"""
select id,name from emp where id>3;
"""执行顺序"""
from
where
select

"""
虽然执行顺序和书写顺序不一致，你在写sql语句的时候可能不知道怎么写
你就按照书写顺序的方式写sql语句
    select * 先用*号占位
    之后去补全后面的sql语句
    最后将*号替换成你想要的具体字段

"""
```

where筛选（过滤）条件

```
#作用：对整体数据的一个筛选操作
"""#1. 查询id大于等于3小于等于6的数据"""
select id,name from emp where id>=3 and id<=6;
select id,name from emp where id between 3 and 6;
"""#2.查询薪资是80000或者60000或者79000的数据"""
select * from emp where salary=80000 or salary=60000 or salary=79000;
select * from emp where salary in(80000,60000,79000);

"""3.查询员工姓名中包含o的员工的姓名和薪资"""
"""
#模糊查询 like
        % 匹配任意多个字符
        _匹配任意单个字符
"""
select id,name from emp where name like '%o%';

"""4.查询员工姓名是由四个字符组成的，姓名和薪资"""
select name,salary from emp where name like '____';#四个下划线
select name,salary from emp where char_length(name)=4;

"""5. 查询id小于3或者大于6的数据
```

```
select * from emp where id<3 or id>6;
select * from emp where id  not between 3 and 6;

"""6. 查询薪资不在20000，18000，17000的数据"""
select * from emp where salary not in (2000,17000,18000);
"""7.查询岗位描述为空的员工的姓名和岗位名"""#针对NULL不能用=号，要用is
select name,post from emp where post_comment is NULL;
```

group by分组

```
#分组实际应用场景
    男女比例
    部门平均薪资
    部门秃头率
    国家之间的数据统计
```

什么时候需要分组？？

```
关键字：每个，平均，最高，最低

"""
聚合函数
    max
    min
    avg
    count
    sum
"""
```

```
"""1.按照部门分组"""
select * from emp group by post;
"""
分组之后，最小可操作单位应该是组，不在是组内的单个数据
上述命令在没有设置严格模式的时候是可以正常执行的，返回的是分组之后，每个组的第一个数据，但是这不符合分组的规范；分组之后不应该考虑单个数据，而应该以组为操作单位（分组后，没法直接获取组内的单个数据）
    如果设置了严格模式，上述命令会直接报错
"""
set blobal sql_mode ="strict_trans_tables,only_full_group_by";

"""设置了严格模式后，默认只能拿到分组的数据
select post from emp group by post;  #按照什么分组就只能拿什么分组
按照什么分组就只能拿到分组，其他字段不能直接获取，需要借助一些方法
"""
"""1.获取每个部门的最高薪资"""
select post, max(salary) from emp group by post;
select post as '部门' ,max(salary) as '薪资' from emp group by post;
#as 可以给字段起别名，也可以直接忽略不写，但是不推荐，因为省略的话语义不明确，容易混乱
"""2. 获取每个部门的最低薪资"""
select post ,min(salary) from emp group by  post;
"""3.获取每个部门的平均薪资"""
select post ,avg(salary) from emp group by post;
"""4.获取每个部门的工资总和"""
```

```
select post,sum(salary) from emp group by post;
"""5.获取每个部门的人数"""
select post,couont(salry) from emp group by post;
select post,count(id) from emp group by post;
select post,count(age) from emp group by post;
select post,count(post_comment) from emp group by post;#null不可以
"""6.查询分组之后的部门名称和每个部门下的所有员工姓名
group_concat"""
#group_concat不单单可以支持获取分组之后的其他字段，还支持拼接操作
select post ,group_concat(name) from emp group by post;
select post ,group_concat(name,'_DSB') from emp group by post;
select post ,group_concat(name,':',salary) from emp group by post;
#concat，不分组的时候用
select concat('NAME:',name) ,concat('SALARY:',salary) from emp;
```

补充 as语法

```
#as语法不单单可以给字段起别名，也可以给表临时起别名
select emp.id,emp.name from emp;相当于select id,name from emp;
select emp.id,emp.name from emp as t1;报错，已经临时改成t1了
需要这样，select t1.id,t1.name from emp as  t1;
```

```
#查询每个人的年薪
select name,salary *12 from emp;
"""如果多个字段之间的连接符号是相同的情况下，你可以直接使用contact_ws来完成
select concat_ws(':',name,age,sex) from emp;#用冒号隔开
"""类似Python里面的join 方法
"""
'?'.join([101,125145,1545])#报错
```

分组注意事项

```
"""关键字where和group by 同时出现的时候，group by 必须在where的后面
where先对整体数据进行过滤，然后在分组操作
聚合函数只能在分组之后使用，where后面不能使用聚合函数，除非使用了分组，where后面才可以跟聚合函数
select id,name,age from emp where max(salary)>3000;报错
select max(salary) from emp;  #不分组默认整张表就是一组，正常执行
"""

"""统计各部门年龄在20岁以上的员工的平均薪资"""
    1.先求所有年龄大于20岁的员工
    select * from emp where age>20;
    2. 在对结果进行分组
    select * from emp where age>20 group by post;
    或者，两步并一步
    select psot,avg(salary) from emp where age>20 group by post;
```

having分组之后的筛选条件

```
having语法跟where是一致的
只不过having 是在分组之后的过滤条件
#而having是可以直接使用聚合函数的
"""1.统计各部门年龄在20岁以上的员工的平均工资，并且保留平均工资大于20000的部门"""
select post,avg(salary) from emp
    where age>20
    group by post
    having avg(salary) >20000;
```

distinct去重

```
"""
必须是完全一样的数据才可以去重
有主键存在的情况下，是不可能去重的
[
{'id':1,'name':'jason','age':18},
{'id':2,'name':'zhao','age':18},
{'id':3,'name':'song','age':18},
]

ORM 对象关系映射，让不同SQL语句的人也能够操作数据库
表            类
一条条数据    对象
字段对应的值    对象的属性
再写类，就意味着在创建表
用类生成对象，就意为者再创建数据
对象点属性，就是在获取数据字段对应的值
目的就是减轻python程序员压力，只需要面向对象知识点就可以操作MySQL
"""

select distinct id,name from emp;#带上主键起不到效果，不能去重
select distinct name from emp;
```

order by排序

```
select * from emp order by salary;
select * from emp order by salary asc;
select * from emp order by salary desc;
"""
order by 默认是升序，   asc 该asc可以省略不写
也可以修改为降序   desc
"""

#如果按照age降序排，遇到age相同，再按salary升序排
select * from emp order by age desc,salary asc;

"""统计各部门在10以上的员工的平均工资，并且保留平均工资大于1000的部门，然后对工资降序
排"""
select post,avg(salary) from emp
    where age>20
    group by post
```

```
    having avg(salary) >1000
    order by avg(salary) desc;
```

limit限制展示条数

```

"""针对数据过多，通常都是做分页处理"""
select * from emp limit 3;#从emp拿数据，只拿3条
select * from emp limit 0,5; #从0开始，展示5条
select * from emp limit 5,5;#从第5个开始，展示5条数据
第一个参数是起始位置，第二个参数是展示条数
```

正则

```

MySQL同样支持正则
select * from emp where name regexp '^j.*(n|y)$';
"""
Python里面使用正则需要先 导入re模块
    1. re常用方法
    findall :分组优先展示，^j.*(n|y)$，不会展示所有正则表达式匹配到的内容，而是展示括号内匹配到的内容
    match：从头匹配
    serch：整体匹配
    2.贪婪与非贪婪匹配
    正则表达式默认 是贪婪匹配
    将贪婪变成非贪婪只需要再正则表达式后面加?
    .* 贪婪模式
    .*? 非贪婪模式
"""
```

多表查询

前期准备

```
create table dep(
    id int,
    name varchar(20)
);
create table emp(
    id int primary key auto_increment,
    name varchar(20),
    sex enum('male','female') not null default 'male',
    age int,
    dep_id int
);
```

```
"""插入数据"""
insert into dep values(200,'技术'),(201,'人力资源'),(202,'销售'),(203,'运营');

insert  into emp(name,sex,age,dep_id) values
('jason','male',18,200),
('egon','female',23,201),
('kevin','male',22,202),
('lisi','male',44,203),
('herry','female',19,204);
```

拼表，联表操作

```
select * from dep,emp;#终端回车后显结果叫 笛卡尔积

select * from emp.dep where emp.dep_id=dep.id;

"""4种方法
inner join 内连接
left join 左连接
right join 右连接
union  全连接
"""
#inner join只拼接两张表中公共的部分
select *  from emp inner join dep on emp.dep_id =dep.id;
#left join 左表所有的数据展示出来，没有对应的数据用NULL
select * from emp left join dep on emp.dep_id=dep.id;
#right join 右表所有的数据展示出来，没有对应的数据用NULL
select *  from emp right join dep on emp.dep_id =dep.id;
#union 全连接，左右两表所有的数据都显示出来
select * from emp left join dep on emp.dep_id=dep.id;
union
select *  from emp right join dep on emp.dep_id =dep.id;
```

子查询

```
"""
子查询就是我们平时解决问题的思路
    分布解决问题
    第一步
    第二步
    .....

将一个查询语句的结果当作另外一个查询语句的条件去用

"""
#查询部门是技术或者人力资源的员工信息
1. 先获取部门的id号
2. 再去员工表里筛选对应的
select id from dep where name="技术" or name="人力资源";

select name from emp where dep_id  in (200,201);

```

```
select * from emp where dep_id in (select id from dep where name="技术" or name="人力资源");
```

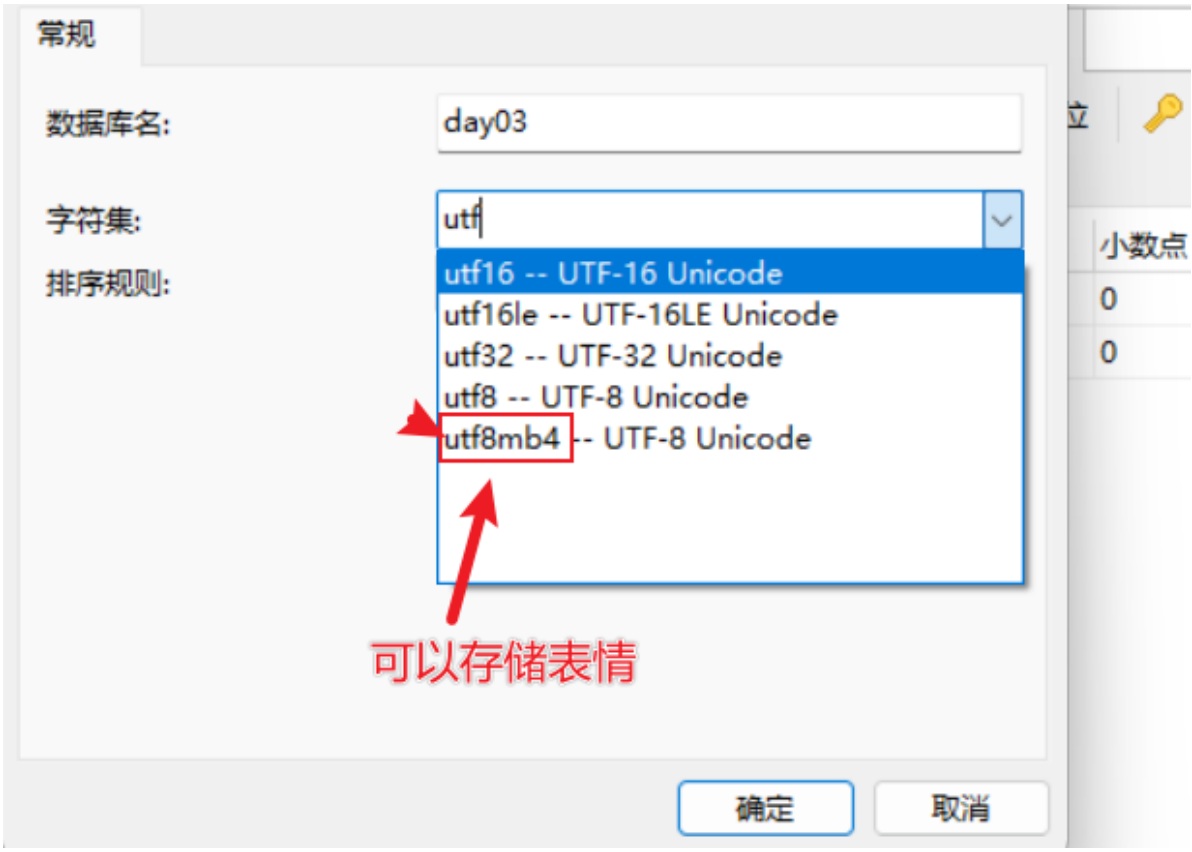
总结

```
表的查询结果可以作为其他表的查询结构
也可以通过起别名的方式把它当作一张虚拟表跟其他表关联
"""
多表查询就两种方式
    1.先拼接表再查询
    子查询一步一步来

"""
```

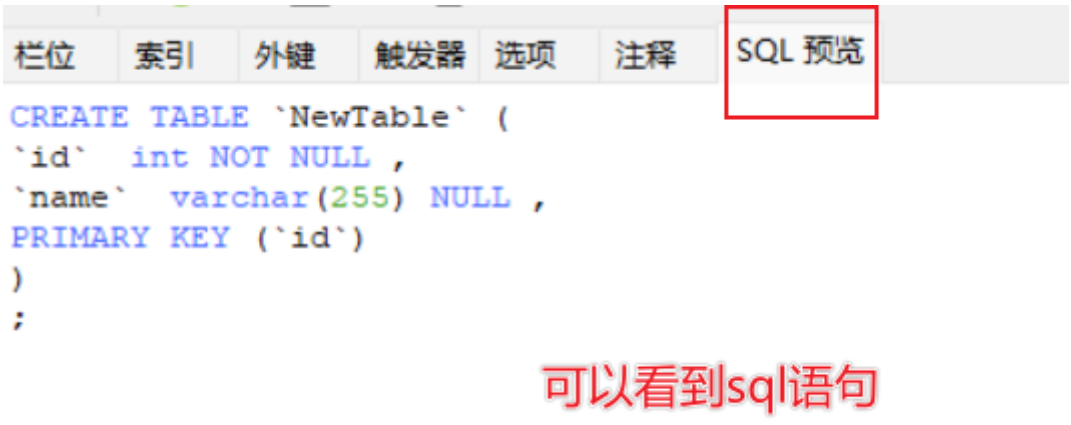
```
"""
查询平均年龄再25岁以上的部门名称

"""
#1.联表操作
1. 先拿到部门和员工表，拼接后的结果
2. 分析语义，得出要分组
select * from emp inner join dep on emp.dep_id=dep.id
group by dep.name
having avg(age)>25
;
涉及到多表操作的时候，一定要加上前缀
#2.子查询
select name from dep where id in
(select dep_id from emp group by dep_id
having avg(age) >25);

#关键字exists
    只返回布尔值 True False
    返回True的时候外层查询语句执行
    返回False的时候外层查询语句不在执行
select * from emp where exists
    (select id from dep where id>3);

select * from emp where exists
    (select id from dep where id>10);
```

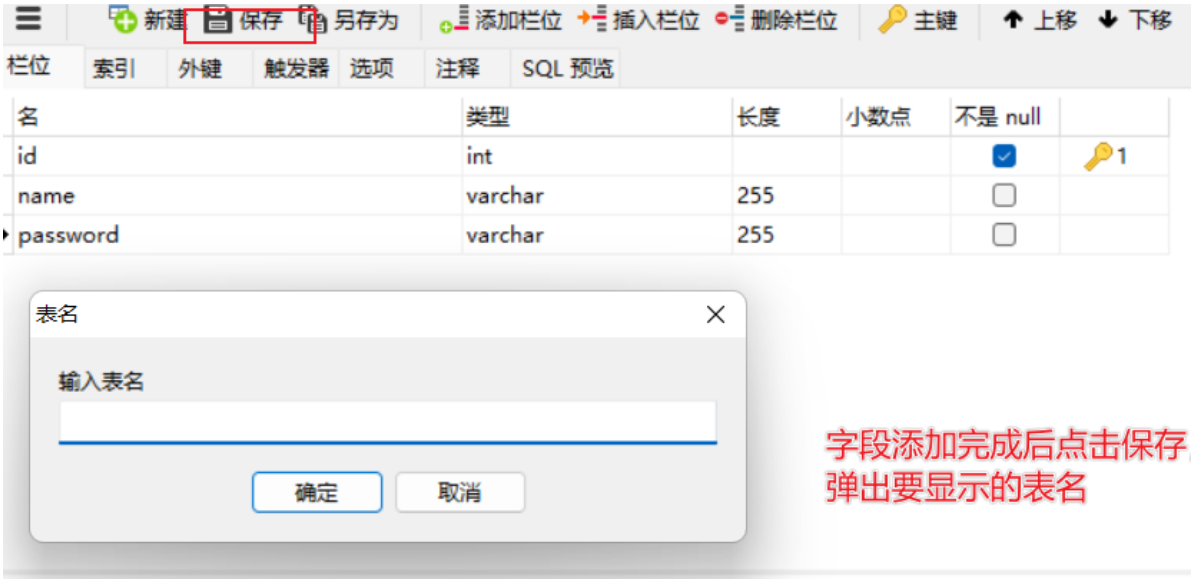
day03

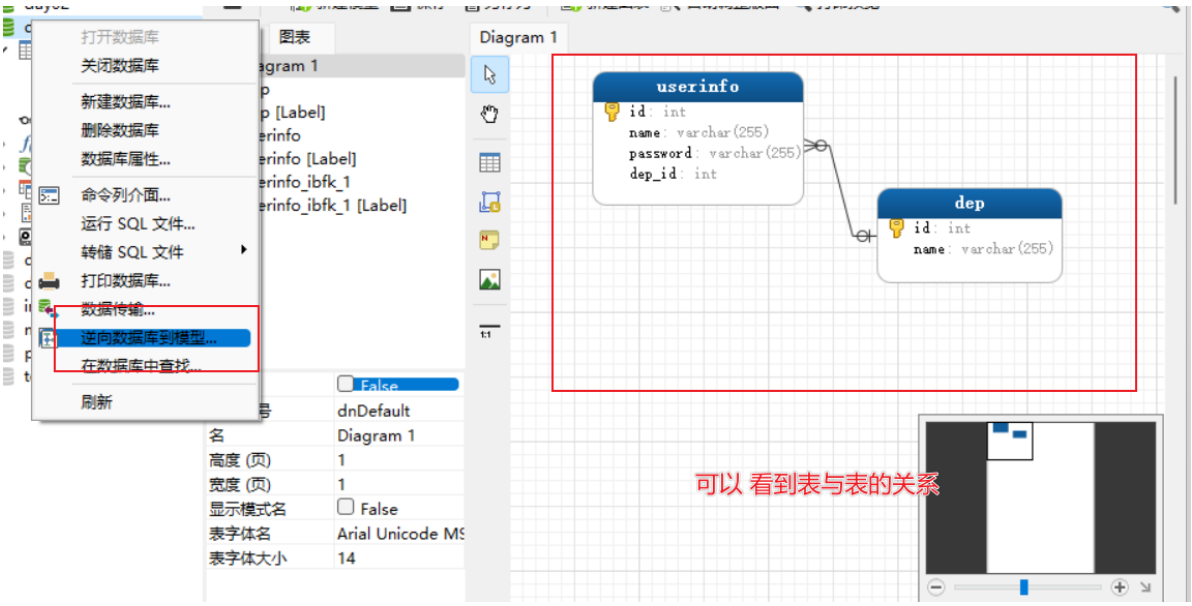
navicat可视化界面操作数据库


常规
数据库名:
day03
字符集:
utf
排序规则:
utf16 -- UTF-16 Unicode
utf16le -- UTF-16LE Unicode
utf32 -- UTF-32 Unicode
utf8 -- UTF-8 Unicode
utf8mb4 -- UTF-8 Unicode
可以存储表情
确定
取消
小数点
0
0


新建
保存
另存为
添加栏位
插入栏位
删除栏位
主键
上移
下移
栏位
索引
外键
触发器
选项
注释
SQL 预览
名
类型
长度
小数点
不是 null
id
int
1
设置主键的时候一定不要忘了
点击下面的自动递增
默认:
注释:
自动递增


栏位
索引
外键
触发器
选项
注释
SQL 预览
CREATE TABLE `NewTable` (
`id` int NOT NULL ,
`name` varchar(255) NULL ,
PRIMARY KEY (`id`)
)
;
可以看到sql语句





MySQL注释有两种

```
--
#
```

navicat快速注释

```
ctrl +?

ctrl +shift +?#老版本情况下
新版本就直接ctrl+？加注释，解注释也是ctrl+?
```

pymysql模块

```
支持python代码操作数据库
```

```
# @Time : 2022/8/16 14:21
# @File : 01pymysql使用.py
import pymysql
import pymysql.cursors
```

```python

conn = pymysql.connect(
    host='127.0.0.1',
    port=3306,
    user='root',
    password='123.com',
    database='day02',
    charset='utf8'  # 编码不要加-

)  # 连接数据库
"""
pymysql.cursors.DictCursor将查询结果以字典的形式返回
"""
# cursor = conn.cursor()  # 生成一个游标对象,用来执行输入的命令
cursor=conn.cursor(cursor=pymysql.cursors.DictCursor)
sql = 'select * from emp;'
res=cursor.execute(sql)
# print(res)#5  返回当前sql语句所影响的行数

#获取命令执行的查询结果
# print(cursor.fetchone())#只拿一条
# print(cursor.fetchall())#拿所有数据
# print(cursor.fetchmany(2))#可以指定拿几条数据
print(cursor.fetchone())
print(cursor.fetchone())#类似于文件光标的移动
cursor.scroll(1,'relative')#相对于光标所在的位置继续往后移动1位
cursor.scroll(1,'absolute')#相对于数据的开头往后移动1位
print(cursor.fetchall())
```

sql注入

```
"""
利用一些语法的特性，书写一些特点的语句实现固定的语法

MySQL利用的是MySQL的注释语法
select * from user where name="zhao" -- jflsdajfl" and password=""
select * from user where name="sfgs" or 1=1 -- faldkf" and password=""

"""
日程生活中，很多软件都不能含有特殊的符号，因为怕构造出特定的语句入侵数据库，不安全


#敏感的数据不要做拼接，交给execute帮你拼接
```

```python
# @Time : 2022/8/16 14:47
# @File : 02sql注入.py


#结合数据库完成用户的登录功能
import  pymysql
import pymysql.cursors

conn= pymysql.connect(
    host='127.0.0.1',
```

```
    port=3306,
    user='root',
    passwd='123.com',
    database='day03',
    charset='utf8'
)
cursor=conn.cursor(cursor=pymysql.cursors.DictCursor)#游标对象
username=input('>>>:')
password=input(">>:")
# sql='select * from user where name=%s" and password="%s"'%
(username,password)
#不要手动的拼接数据，先用%占位，之后将需要的拼接的数据直接交给execute方法即可
sql='select * from user where name =%s and password=%s '
print(sql)
#rows=cursor.execute(sql)#返回受影响的行数
rows=cursor.execute(sql,(username,password))#自动识别sql里面的%s 用后面元组里面的
数据替换
if rows:
    print('登录OK')
    print(cursor.fetchall())
else:
    print('登陆失败')

```

pymysql补充

```
"""增删改查"""
#针对增删改 pymysql需要二次确认才能真正的操作数据
```

```
# @Time : 2022/8/16 16:37
# @File : 03pymyql补充.py
import configparser

import pymysql
import pymysql.cursors

conn =pymysql.connect(
    host='127.0.0.1',
    port=  3306,
    user='root',
    passwd='123.com',
    database='day03',
    charset='utf8',
    autocommit=True
)
cursor =conn.cursor(cursor=pymysql.cursors.DictCursor)

#增
sql='insert into user(name,password) values(%s,%s)'
# rows=cursor.execute(sql,('xijie',456))
rows=cursor.executemany(sql,[('yyy',455),('eww',455),('wg',784)])#添加多条数据
print(rows)
# conn.connect()#确认，

```

```
26
27
28
29 # #修改
30 # sql='update user set name="王五" where id =1'
31 # rows=cursor.execute(sql)
32 # print(rows)
33 # conn.commit()
34
35
36 # #删除
37 # sql='delete from user where id=2'
38 # rows=cursor.execute(sql)
39 # print(rows)
40 # conn.commit()
41
42 #查
43 sql='select * from user'
44 cursor.execute(sql)
45 print(cursor.fetchall())
46
47
48 """
49 增删改查中
50 增删改操作涉及到数据的修改，需要二次确认
51 """
```

一次性插入多条数据

```
1 rows=cursor.executemany(sql,[('ww',112),('rre',754),('ew',45)])
```

二次确认

```
1 添加autocommiy=True就不需要每次conn.commit()了
```

视图

- 视图就是通过查询到的一张虚拟表，然后保存下来，下次可以直接使用
- 如果要频繁的使用一张虚拟表（拼接组成的），就可以制作成视图，后续直接操作

```
1 """固定写法"""
2 create view 表名 as 虚拟表的查询sql语句
3 """具体操作"""
4 create view virtual as
5 SELECT * from user INNER JOIN  userinfo
6 on user.id=userinfo.dep_id;
```

注意：

1. 创建视图在硬盘上只会有表结构，没有数据，（数据还是来自原原来的表）
2. 视图一般只用来查询，里面的数据不要继续修改，可能影响真正的数据

触发器

在满足对表的数据进行增、删、改的情况下，自动出发的功能

使用触发器可以帮助我们实现监控，日志，自动处理...

触发器可以在六种情况下自动触发，增前，增后，删前，删后 ，改前，改后

基本语法结构

```
create trigger 触发器名字 before/after/insert/update/delete
on
表名
for each row
begin
    sql语句
end

```

具体使用

针对触发器的名字，通常做到见名知意

```
#增
create trigger rei_before before insert on t1
for each row
begin
    sql语句
end

create trigger rei_after after insert on t1
for each row
begin
    sql语句
end

"""删除修改，格式一致"""
```

修改MySQL默认的语句结束符、

```
delimiter $$  将默认的结束符号由分号改为$$
delimiter ; 再改回来
只作用于当前窗口
```

案例

```
create table cmd(
    id int primary key auto_increment,
    user char(32),
    priv char(10),
    cmd char(64),
    sub_time datetime, #提交时间
    success enum('yes','no')#0代表执行失败
):
create table err_log(
    id int primary key auto_increment,
```

```
    err_cmd char(64),
    err_time datetime
);

""" 当cmd表中的secces字段是no那么就触发触发器的执行去err_log表中插入数据
 """
# NEW指代的就是一条条数据对象

delimiter $$
create trigger tri_after_insert_cmd after insert on cmd
for each row
begin
    if NEW.success='no' then
        insert into err_log(err_cmd.err_time)
values(NEW.cmd,NEW.sub_time);
    end if;
end $$
delimiter ;

#往cmd表插入数据
insert into cmd(
    user,
    priv,
    cmd,
    sub_time,
    seccess
)
values
    ('jason','1755','ls  -l/etc',NOW(),'yes'),
    ('jason','1755','cat /etc/password',NOW(),'no'),
    ('jason','1755','useradd xxx',NOW(),'no'),
    ('jason','1755','ps aux',NOW(),'yes');

```

事务

概念:

- 开启一个事务，可以包含多条sql语句，这些sql语句要么同时成功，要么一个都别想成功，称之为事务的原子性

作用:

- 保证对数据操作的安全性

```
eg:还钱例子
    egon用银行卡给我的支付宝转账1000
         1.将egon银行卡账户的数据减1000快
        2. 将我的支付宝账户的数据加1000块

你在操作多条数据的时候可能出现莫几条操作不成功的情况
```

事务的四大特性

```
"""
```

```
ACID
A:原子性
    一个事务是一个不可分割的单位，事务中包含的诸多操作要么同时成功，要么同时失败
C:一致性
    事务必须是使数据库从一致性的状态变到另外一个一致性的状态
    一致性跟原子性是密切相关的
I:隔离性、
    一个事务的执行不能被其他事务干扰，
    (即一个事务内部的操作及使用到的数据对并发的其他事务是隔离的，并发执行的事务之间是互相不干扰的)
D:持久性
    也叫”永久性“，
    指一个事务一旦提交成功，那么他对数据库中数据中数据的修改应该是永久的
    接下来的其他操作或者故障不应该对其有任何的影响

"""
```

如何使用事务

```
#   事务相关的关键字
#1.开启事务
start transaction;
#2.回滚（回到事务之前的状态）
rollback;
#3.二次确认（确认之后无法回滚）
commit ;
```

模拟转账功能

```
create table user(
    id int primary key auto_increment,
    name char(16),
    balance int
);
insert into user(name,balance) values
('zhao',1000),('lisi',1000),('tank',1000);

#1. 开启事务
start transaction;
#2. 修改多条sql语句
updatge user set balance=900 where name='zhao';
updatge user set balance=1900 where name='lisi';
updatge user set balance=800 where name='tank';#
#回滚
rollback;
```

存储过程

存储过程类似于Python中的自定义函数

包含了一些列可以执行的sql语句，存储过程存放于MySQL服务器中，你可以直接通过调用存储过程触发内部的sql语句的执行

基本使用

```
create procedure 存储过程名字(形参1,形参2....)
begin
    sql语句
end
#调用
call 存储过程的名字();
```

三种开发模式

1

```
"""
应用程序程序员写代码
MySQL：提前编写好存储过程，最后供应用程序调用


好处：开发效率提升，执行效率也上去了
缺点，后续的存储过程扩展性差
"""
```

2

```
"""
应用程序:程序员写代码之外，，涉及到数据库操作也可以自己写
优点：扩展性高
缺点：开发效率低，编写sql语句太繁琐，后续还需要考虑sql优化的问题

"""
```

3 一般都是第三种

```
"""
应用程序:只写程序代码，不写sql语句，基于他人写好的操作MySQL的python框架直接调用    ORM框架


优点：开发效率提高
缺点：语句的扩展性差，可能出现效率低下的东西

"""
```

存储过程具体演示：

```
delimeter $$
create procedure p1(
    in m int,  #只进不出，m不能返回
    in n int,
    out res int  #该形参可以返回出去
)
begin
    select * tname from teacher where tid>m and tid<n;
    set res=0 #将res修改，用来标识当前的存储过程代码确实执行了
end $$
delimiter ;
```

```
12
13 #调用
14 call p1()
15
16 #针对形参res 不能直接传数据，应该传一个变量名
17 #定义变量
18 set @ret=10;
19 #再查看变量对应的值
20 select @ret;
```

再pymysql中如何调用存储过程

```
1  import  pymysql
2  import pymysql.cursors
3
4  conn= pymysql.connect(
5      host='127.0.0.1',
6      port=3306,
7      user='root',
8      passwd='123.com',
9      database='day03',
10     charset='utf8'
11 )
12 cursor=conn.cursor(cursor=pymysql.cursors.DictCursor)
13 #调用存储过程
14 cursor.callproc('p1',(1,5,10))
15 print(cursor.fetchall())
```

函数

跟存储过程是有区别的，存储过程是自定义函数，而函数就类似于内置函数

```
1 ('jason','1755','ls  -l/etc',NOW(),'yes'),
```

流程控制

```
1  #if判断
2  delimiter //
3  create procedure proc_if()
4  begin
5      declare i int default 0;
6      if i = 1 then
7          select 1;
8      elseif i =2 then
9          select 2:
10     else
11         select 7;
12     end if;
13 end //
14 delimiter ;
15
16 #while循环
17 delimiter //
```

```
create procedure proc_while ()
begin
    declare num int;
    set num =0;
    while num <10 do
        select
            num;
        select num =num + 1;
    end while;


```

索引

- 数据都是存在硬盘上的，查询数据不可避免的需要进行IO操作
- 索引就是一种数据结构，类似于书的目录，意味着以后再查询数据的时候，应该先找目录，再找数据，而不是一页一页的翻书，从而提升查询速度降低IO操作
- 索引在MySQL中也叫“键”，是存储引擎用于快速查找记录的的一种数据结构
    - primary key
    - unique key
    - index key
- 注意foreign key 不是用来加速查询用的，
- 上面的三种key，前面两种可以增加查询速度之外各自还具有约束条件，而最后一中index key 没有任何约束条件，只是用来帮助快速查询数据

本质

通过不断的缩小像哟啊的数据范围筛选出最终的结果，同时将随机事件（一页一页的翻）变成顺序事件（先找目录，再找数据）

也就是说有了索引机制，可以总是用一种固定的方式查找数据

一张表中可以有多个索引（多个目录）

缩影虽然能够快速的加快查询速度，但是也有缺点

```
1. 当表中有大量数据存在的前提下，创建索引速度会非常慢，
2. 在索引创建完毕后，对表的查询性能会大大提升，但是写的性能也会大大的降低

索引不要随意的创建！！！！！！！！！！！！！！
```

b+树

只有叶子节点存放的是真实的数据，其他节点存放的是虚拟数据，仅仅是用来指路的

树的层级越高查询数据所需要经历的步骤就越多，（树有几层查询数据就需要几步）

一个磁盘块存储是有限制的

为社么将id字段作为索引

占用空间少，一个磁盘块能够存储的数据多

那么就降低了树的高度，从而减少查询次数

聚集索引 (primary key)

聚集索引指的就是主键，

innodb 只有两个文件，直接将主键存放在idb表中

MyIsam 三个文件，单独将索引存在一个文件

辅助索引 （unique ,index)

查询数据的时候，不肯能一直使用主键，也有可能使用到name ,password等其他字段，那么这个时候是没有办法利用聚集索引，这个时候就可以根据情况给其他字段设置辅助索引（也是一个b+树)

```
"""
叶子节点存放的是数据对应的主键值
    先按照辅助索引拿到数据的主键值
    之后还是需要去主键的聚集索引里面查询数据
"""

```

覆盖索引

在辅助索引的叶子节点就已经拿到了需要的数据

```
#给name设置辅助索引
select name from user where name ="zhao";
select age from user where name='zhao';#非覆盖索引
```