MySQL 高级

复习

查询条件

聚合函数

分组与分页

连接查询

子查询

保存查询结果

导入

前面学习的数据库基础，只能满足基本的需求，面对更复杂的功能需求或优化需求，我们需要使用事务、视图、索引、函数、存储过程等 MySQL 的高级部分来完成，以及如何通过 Python 来连接、操作数据库也是我们学习的重点。

目录

1. 用户与权限管理
2. 事务
3. 视图和索引
4. 存储过程
5. 函数
6. 查询实战
7. Python DB-API

目标

1. 通过创建用户并赋予权限，来满足后续开发的需要。

2. 通过事务处理，来保证数据的一致性。（重点、难点）
3. 使用视图简化 SQL 语句的频繁编写，并提高效率。
4. 合理增加索引，优化查询速度。（重点）
5. 使用函数、存储过程完成更加复杂的功能需求。（重点、难点）
6. 使用 Python 的 pymysql 模块完成数据库的连接、数据的操作。（重点、难点）

一、用户与权限管理

1、MySQL 账户管理概述

MySQL 的账户管理包括登录和退出 MySQL 服务器、创建用户、删除用户、密码管理和权限管理等内容。通过账户管理，可以保证 MySQL 数据库的安全性。

MySQL 中 root 账号拥有最高权限，包括删库，删表。所以在生产环境下一般不会使用 root 账号登录数据库的。MySQL 中的用户信息保存在 mysql 库下的 user 表中。

进入到 msyql 库查看 msyql 中所有用户信息

```
use mysql;
select host,user from user;
```

```
mysql> select user,host from user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| root             | %         |
| debian-sys-maint | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
| root             | localhost |
+------------------+-----------+
5 rows in set (0.00 sec)

mysql>
```

2、创建用户并分配权限

```
grant 权限列表 on 数据库 to '用户名'@'访问主机' identified by '密码';
```

MySQL 中权限有 create、alter、drop、insert、update、delete、select 等，如果要分配所有权限，直接使用：all privileges

下面给数据库创建一个 python 用户分配 select 权限，允许所有主机登录，密码为 msyql

```
grant select on *.* TO 'python'@'%' IDENTIFIED BY 'MySQL' with grant option;
flush privileges;
```

说明：

. ：表示所有数据库的所有表

with grant option：表示它具有 grant 权限， 可以创建用户。

flush privileges：表示让赋予的权限立即生效。

3、查看、回收权限

查看用户的权限

```
show grants for 用户名;
```

```
show grants for python;
```

回收权限

```
revoke select on  *.*  from 'python'@'%';
```

4、修改密码

普通用户修改自己的密码

在终端上修改不需要进到数据库

```
MySQLadmin -upython -p password 新密码
```

Root 账号修改普通用户的密码：

```
1. 修改 MySQL.user 表 update MySQL.user set authentication_string=password(123)(新密码)  where user='python'(用户名)

2.刷新权限    flush privileges;
```

5、删除用户

方法 1：drop user '用户名'@'主机';

方法 2：进入到 MySQL 这个库 delete from user where user='用户名';

二、事务

1、 事务概述

事务：也称工作单元，是由一个或多个 SQL 语句所组成的操作序列，这些 SQL 语句作为一个完整的工作单元，要么全部执行成功，要么全部执行失败。在数据库中，通过事务来保证数据的一致性。

事务处理语言：Transaction Process Language ,简称 TPL，主要用来对组成事务的 DML 语句的操作结果进行确认或取消。确认也就是使 DML 操作生效，使用提交(COMMIT)命令实现；取消也就是使 DML 操作失效，使用回滚(ROLLBACK)命令实现。

通过事务的使用，能防止数据库中出现数据不一致现象。如两个银行账户进行转账，涉及到两条更新操作，这两条更新操作只允许全部成功或失败，否则数据会出现不一致的现象。

MySQL 是支持事务的，跟使用的引擎相关。MySQL 中支持多种引擎，默认使用 Innodb 引擎支持事务。

- MyISAM：不支持事务，用于只读程序提高性能
- InnoDB：支持 ACID 事务、支持行级锁
- Berkeley DB：支持事务

2、事务的特性(ACID)

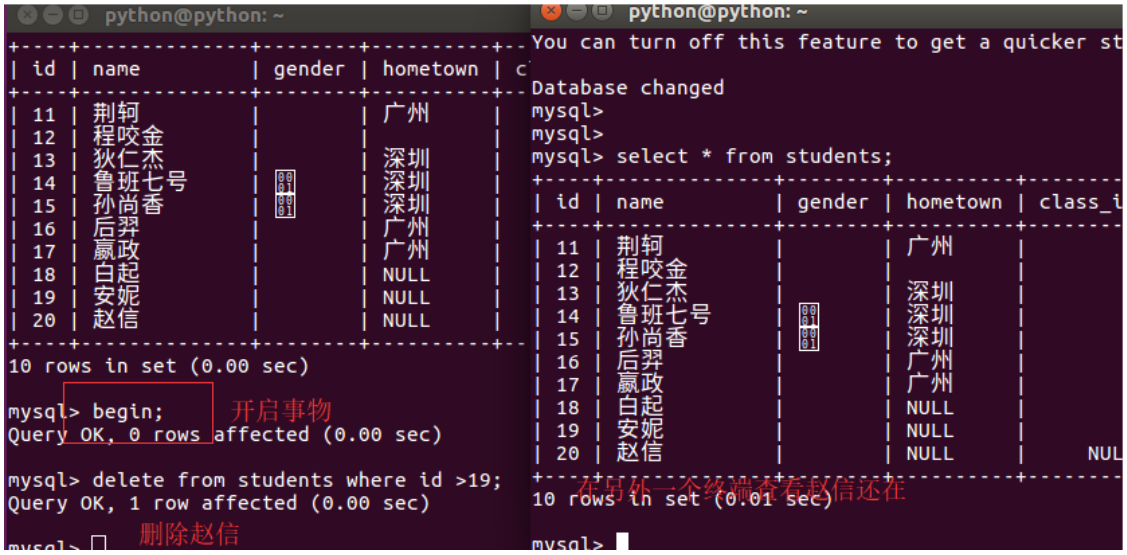
原子性（Atomicity）：事务就像“原子”一样，不可被分割，组成事务的 DML 操作语句要么全成功，要么全失败，不可能出现部分成功部分失败的情况。 一致性（Consistency）：一旦事务完成，不管是成功的，还是失败的，整个系统处于数据一致的状态。 隔离性（Isolation）：一个事务的执行不会被另一个事务所干扰。比如两个人同时从一个账户从取钱，通过事务的隔离性确保账户余额的正确性。 持久性（Durability）：也称为永久性，指事务一旦提交，对数据的改变就是永久的，不可以再被回滚。

3、事务处理

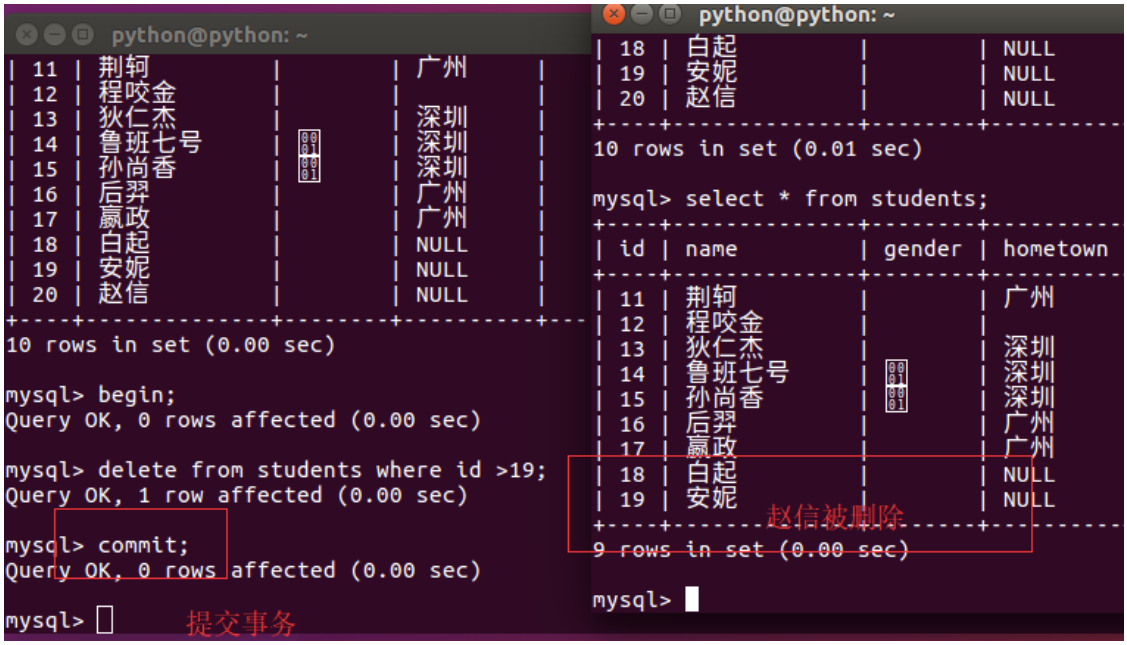
（1）手动提交事务

用 begin,rollback,commit 来实现，用 begin 开启事务后在没有 commit 提交之前执行修改命令，变更会维护到本地缓存中，而不维护到务理表中，只有在 commit 提交后才会更新到务理表中。如果中间执行错误，那么用 rollback 回滚事务，恢复到执行事务前的状态。

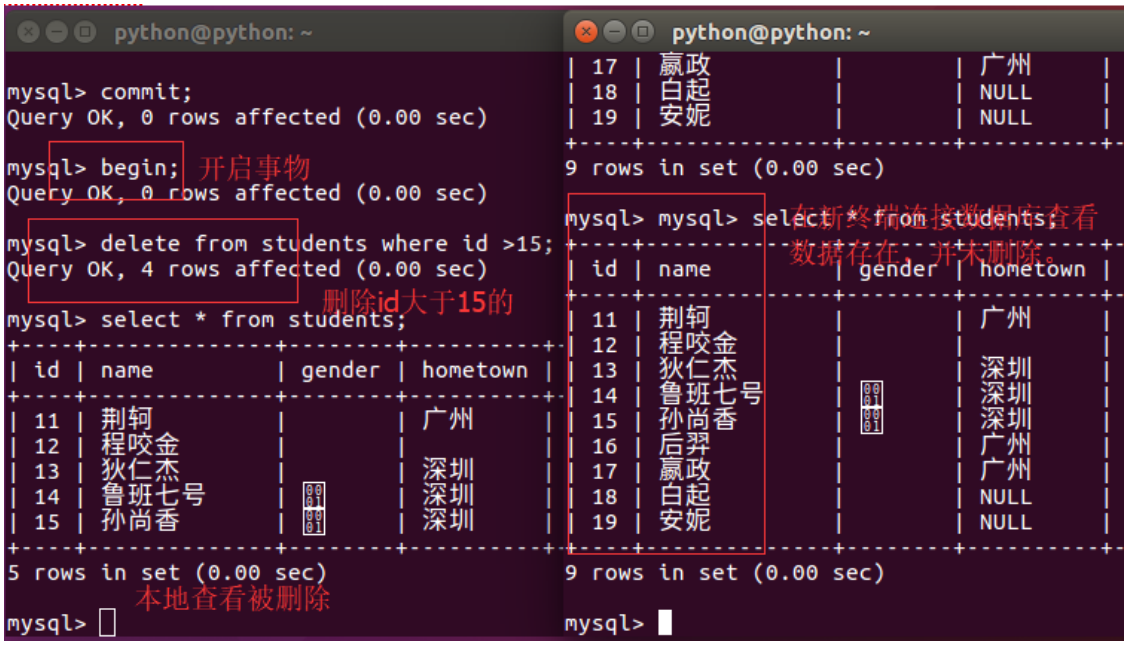
begin 开启事务



commit 提交事务



rollback 事务回滚



执行 rollback 回滚

```
mysql> rollback;
Query OK, 0 rows affected (0.00 sec)
       执行事物回滚后，刚才执行删除的数据又回来了
mysql> select * from students;
+----+--------------+--------+----------+----------+------+
| id | name         | gender | hometown | class_id | age  |
+----+--------------+--------+----------+----------+------+
| 11 | 荆轲         |        | 广州     |        1 |   18 |
| 12 | 程咬金       |        |          |        1 |   20 |
| 13 | 狄仁杰       |        | 深圳     |        1 |   18 |
| 14 | 鲁班七号     |        | 深圳     |        1 |   15 |
| 15 | 孙尚香       |        | 深圳     |        2 |   12 |
| 16 | 后羿         |        | 广州     |        2 |   18 |
| 17 | 嬴政         |        | 广州     |        2 |   14 |
| 18 | 白起         |        | NULL     |        2 |   22 |
| 19 | 安妮         |        | NULL     |        2 |   21 |
+----+--------------+--------+----------+----------+------+
9 rows in set (0.01 sec)
```

（2）自动提交模式

MySQL 默认是自动提交的，也就是你提交一个 sql，就直接执行。可以通过 set autocommit = 0 禁止自动提交，set autocommit = 1 开启自动提交， 来实现事务的处理。

但要注意当用 set autocommit = 0 的时候，以后所有的 sql 都将作为事务处理，直到用 commit 确认或 rollback 结束，注意当结束这个事务的同时也开启了新的事务。按第一种方法只将当前的做为一个事务。

（3）隐式处理

隐式提交：当下列任意一种情况发生时，会发生隐式提交

- 执行一个 DDL(CREATE、ALTER、DROP、TRUNCATE、RENAME）语句；
- 执行一个 DCL(GRANT、REVOKE)语句；

隐式回滚：当下列任意一种情况发生时，会发生隐式回滚

- 客户端强行退出
- 客户端连接到服务器端异常中断
- 系统崩溃

三、视图和索引

1、视图

对于复杂的查询，在多个地方被使用，如果需求发生了改变，需要更改 sql 语句，则需要在多个地方进行修改，维护起来非常麻烦。这种情况下可以定义视图解决，视图本质上就是对查询语句的封装。视图实际上就是查询。

创建视图：

```
create view 视图名称 as select 语句;
```

例：创建视图，查询学生对应的成绩信息

```
create view v_stu_sco as
      select students.*,scores.score from scores
      inner join students on scores.stuid=students.id;
```

查看视图：查看表会将所有的视图也列出来

```
show tables;
SHOW TABLE STATUS;
```

删除视图：

```
drop view 视图名称;
```

调用视图：

```
select * from v_stu_score;
```

2、索引

（1）索引概述

一般的应用对数据库大部分操作都是查询，所以查询速度显得尤为重要。当数据库中数据量很大时，查找数据会变得很慢，这个时候就需要做相应的优化处理。

建立索引是一个有效的优化方案，索引就好比一本书的目录，它会让你更快的找到内容，显然目录（索引）并不是越多越好，假如这本书 1000 页，有 500 也是目录，它当然效率低，目录是要占纸张的，而索引是要占磁盘空间的。

选择索引的数据类型，越小的数据类型越好，越小的数据类型通常在磁盘、内存和 CPU 缓存中都需要更少的空间，处理起来更快。简单的数据类型更好，整型数据比起字符串，处理开销更小，因为字符串的比较更复杂。

MySQL 常见索引有：主键索引、唯一索引、普通索引、全文索引、组合索引

（2）创建索引

PRIMARY KEY（主键索引）：

```
alter table 表名 add primary key (列名);
```

UNIQUE(唯一索引)：

```
alter table 表名 add unique (列名);
```

INDEX(普通索引)：

```
alter table 表名 add index 索引名称 （列名);
```

FULLTEXT(全文索引)：

```
alter table add fulltext (列名);
```

组合索引：

```
alter table 表名 add index 索引名 (列 1，列 2，列 2);
```

（3）查看、删除索引

查看索引

```
show index from 表名;
```

删除索引

```
drop index 索引名 on 表名;
```

（4）查询速度测试

创建一个表 myindex 表，向里面插入 50 万条数据

```
create table myindex(
id int auto_increment primary key  not null,
test varchar(10)
)
```

写一段 python 脚本插入将 50 万条数据插入

```
# coding=utf-8
from pymysql import *
conn = connect(host='127.0.0.1',port=3306,database='python',user='root',
password='mysql',charset='utf8')
cs1=conn.cursor()
```

```
for i in range(0,500000):
    str_ = 'test' +str(i)
    sql = "insert into myindex values(0,%s);"
    cs1.execute(sql,[str_])
conn.commit()
cs1.close()
conn.close()
```

创建好表之后开始查询不建索引与建立索引需要的时间

开启 profiling 参数，MySQL 的 Query Profiler 是一个使用非常方便的 Query 诊断分析工具

```
set profiling=1;
```

没有给 test 字段创建索引的时候,查找第 400000 条数据

```
select * from myindex where test='test400000';
```

查看执行时间

```
show profiles;
```

```
mysql> set profiling=1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> select * from myindex where test='test400000';
+--------+------------+
| id     | test       |
+--------+------------+
| 400001 | test400000 |
+--------+------------+
1 row in set (0.11 sec)

mysql> show profiles;
+----------+------------+--------------------------------------------+
| Query_ID | Duration   | Query                                      |
+----------+------------+--------------------------------------------+
|        1 | 0.11141775 | select * from myindex where test='test400000' |
+----------+------------+--------------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

建立索引

```
alter table myindex add index index_name test;
```

查找第 400000 条数据

```
select * from myindex where test='test400000';
```

查看执行时间

```
show profiles;
```

```
mysql> alter table myindex add index index_test(test);
Query OK, 0 rows affected (0.71 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> set profiling=1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> select * from myindex where test='test400000';
+--------+------------+
| id     | test       |
+--------+------------+
| 400001 | test400000 |
+--------+------------+
1 row in set (0.00 sec)

mysql> show profiles;
+----------+------------+-------------------------------------------+
| Query_ID | Duration   | Query                                     |
+----------+------------+-------------------------------------------+
|        1 | 0.00023125 | select * from myindex where test='test400000' |
+----------+------------+-------------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

四、存储过程

1、存储过程概述

存储过程，也翻译为存储程序，是一条或者多条 SQL 语句的集合，可以视为批处理，但是其作用不仅仅局限于批处理。 本节主要介绍如何创建存储过程，以及如何调用、查看、修改、删除存储过程，存储过程也可以调用其他存储过程。

2、创建储存过程

（1）语法

```
delimiter //
create procedure 存储过程名称(参数列表)
begin
sql 语句
end
//
delimiter ;
```

- delimiter 用于设置 sql 语句分割符，默认为分号
- 在“sql 语句”部分编写的语句需要以分号结尾，此时回车会直接执行，所以要创建存储过程前需要指定其它符号作为分割符，此处使用//，也可以使用其它字符

示例：创建储存过程查询学生信息

```
delimiter //
create procedure proc_stu()
begin
select * from students;
end
//
delimiter ;
```

（2）查看创建的存储过程

所有存储过程和函数，都存储在 mysql 数据库下的 proc 表中

proc 表中的字段说明

- name 表示名称
- type 表示类型，为存储过程、函数
- body 表示正文脚本
- db 表示属于的数据库

查询刚才创建的储存过程

```
select name,type,body from mysql.proc where db='python';
```

查询 python 表中的存储过程和函数

```
select name,type,body from mysql.proc where db='python';
```

（3）调用储存过程

调用存储过程语法：

```
call 存储过程(参数列表);
```

调用 proc_stu 储存过程

```
call proc_stu();
```

（4）删除储存过程

语法：

```
drop procedure 存储过程名称;
```

3、存储过程中的变量

要在存储过程中声明一个变量，可以使用 DECLARE 语句，如下所示：

```
DECLARE variable_name datatype DEFAULT default_value;
```

- 在 DECLARE 关键字后面要指定变量名。变量名必须遵循 MySQL 表列名称的命名规则。
- 指定变量的数据类型及其大小。变量可以有任何 MySQL 数据类型，如 INT，VARCHAR，DATETIME 等。
- 当声明一个变量时，它的初始值为 NULL。但是可以使用 DEFAULT 关键字为变量分配默认值。

示例：可以声明一个名为 total_sale 的变量，数据类型为 INT，默认值为 0，如下所示：

```
DECLARE total_sale INT DEFAULT 0;
```

设置变量值

```
DECLARE total_count INT DEFAULT 0;
SET total_count = 10;
```

除了 SET 语句之外，还可以使用 SELECT INTO 语句将查询的结果分配给一个变量。

```
DECLARE total_products INT DEFAULT 0;
```

```
SELECT COUNT(*) INTO total_products FROM students;
```

示例

```
delimiter //
create procedure proc_stu_2()
begin
DECLARE total_count INT DEFAULT 0;
SET total_count = 10;
select * from students;
end
//
delimiter ;
```

4、条件语句

（1）IF 语句

语法：

```
IF expression THEN
   statements;
END IF;
```

如果表达式(expression)计算结果为 TRUE，那么将执行 statements 语句，否则控制流将传递到 END IF 之后的下一个语句。

（2）IF...ELSE 语句

如果表达式计算结果为 FALSE 时执行语句，请使用 IF ELSE 语句，如下所示：

```
IF expression THEN
   statements;
ELSE
   else-statements;
END IF;
```

（3）IF...ELSEIF...ELSE 语句.

如果要基于多个表达式有条件地执行语句，则使用 IF ELSEIF ELSE 语句如下：

```
IF expression THEN
   statements;
ELSEIF elseif-expression THEN
   elseif-statements;
...
ELSE
   else-statements;
END IF;
```

5、WHILE 循环语句

WHILE 语句的语法如下：

```
WHILE expression DO
   statements
END WHILE
```

WHILE 循环在每次迭代开始时检查表达式。 如果 expression 为 TRUE，MySQL 将执行 WHILE 和 END WHILE 之间的语句，直到 expression 为 FALSE。 WHILE 循环称为预先测试条件循环，因为它总是在执行前检查语句的表达式。

示例：

```
DELIMITER $$
 DROP PROCEDURE IF EXISTS test_mysql_while_loop$$
 CREATE PROCEDURE test_mysql_while_loop()
 BEGIN
 DECLARE x  INT;
 DECLARE str  VARCHAR(255);

 SET x = 1;
 SET str =  '';

 WHILE x  <= 5 DO
 SET  str = CONCAT(str,x,',');
 SET  x = x + 1;
 END WHILE;

 SELECT str;
 END$$
DELIMITER ;
```

CONCAT(str1,str2，',') 字符串拼接方法。 利用逗号对字符串进行拼接

五、函数

1、内置函数

字符串函数

查看字符的 ascii 码值 ascii(str)，str 是空串时返回 0

```
select ascii('a');
```

查看 ascii 码值对应的字符 char(数字)

```
select char(97);
```

拼接字符串 concat(str1,str2...)

```
select concat(12,34,'ab');
```

包含字符个数 length(str)

```
select length('abc');
```

截取字符串

left(str,len)返回字符串 str 的左端 len 个字符
right(str,len)返回字符串 str 的右端 len 个字符
substring(str,pos,len)返回字符串 str 的位置 pos 起 len 个字符

```
select substring('abc123',2,3);
```

去除空格

ltrim(str)返回删除了左空格的字符串 str
rtrim(str)返回删除了右空格的字符串 str
trim([方向 remstr from str)返回从某侧删除 remstr 后的字符串 str，方向词包括 both、leading、trailing，表示两侧、左、右

```
select trim('  bar   ');
select trim(leading 'x' FROM 'xxxbarxxx');
select trim(both 'x' FROM 'xxxbarxxx');
select trim(trailing 'x' FROM 'xxxbarxxx');
```

返回由 n 个空格字符组成的一个字符串 space(n)

```
select space(10);
```

替换字符串 replace(str,from*str*,to*str*)

```
select replace('abc123','123','def');
```

大小写转换，函数如下

```
lower(str)
upper(str)
select lower('aBcD');
```

日期时间函数

获取子值，值为整数类型，函数如下

year(date)返回 date 的年份(范围在 1000 到 9999)
month(date)返回 date 中的月份数值
day(date)返回 date 中的日期数值
hour(time)返回 time 的小时数(范围是 0 到 23)
minute(time)返回 time 的分钟数(范围是 0 到 59)
second(time)返回 time 的秒数(范围是 0 到 59)

```
select year('2016-12-21');
```

日期计算，使用+-运算符，数字后面的关键字为 year、month、day、hour、minute、second

```
select '2016-12-21'+interval 1 day;
```

日期格式化 date_format(date,format)

参数 format 可选值如下

%Y 获取年，返回完整年份
%y 获取年，返回简写年份
%m 获取月，返回月份
%d 获取日，返回天值
%H 获取时，返回 24 进制的小时数
%h 获取时，返回 12 进制的小时数
%i 获取分，返回分钟数
%s 获取秒，返回秒数

示例如下：将使用-拼接的日期转换为使用空格拼接

```
select date_format('2016-12-21','%Y %m %d');
```

当前日期 current_date()

```
select current_date();
```

当前时间 current_time()

```
select current_time();
```

当前日期时间 now()

```
select now();
```

2、自定义函数

（1）创建自定义函数

语法：

```
delimiter $$
create function 函数名称(参数列表) returns 返回类型
begin
sql 语句
end
$$
delimiter ;
```

示例：

```
delimiter $$
create function py_trim(str varchar(100)) returns varchar(100)
begin
declare x varchar(100);
```


```
set x=ltrim(rtrim(str));
return x;
end
$$
delimiter ;
```

（2）查看创建的自定义函数

所有函数存储在 mysql 数据库下的 proc 表中，存储过程与函数都存储在 proc 表中，区别在 type 字段，函数的 type 字段为 FUNCTION

查看 python 数据库中的函数

```
select name,type from mysql.proc where db='python';
```

（3） 调用自定义函数

select 函数名称(参数列表);

```
mysql> select py_trim(' ss');
+----------------+
| py_trim(' ss') |
+----------------+
| ss             |
+----------------+
1 row in set (0.00 sec)
```

六、查询实战

1、准备数据

创建表：

```
create table goods(
    id int unsigned primary key auto_increment not null,
    name varchar(150) not null,
    cate varchar(40) not null,
    brand_name varchar(40) not null,
    price decimal(10,3) not null default 0
);
```

```
insert into goods values(0,' Apple MacBook Air 13.3 英寸笔记本电脑','笔记本','苹果','6588');
```

```
insert into goods values(0,'联想(Lenovo)拯救者 R720 15.6 英寸大屏','笔记本','联想','6099');
insert into goods values(0,'法国酒庄直采原瓶原装进口 AOC 级艾落干红葡萄酒','红酒','法国','499');
insert into goods values(0,'x550cc 15.6 英寸笔记本','笔记本','华硕','2799');
insert into goods values(0,'清扬(CLEAR)洗发水','洗发水','清扬','35');
insert into goods values(0,'荣耀 MagicBook 14 英寸轻薄窄边框笔记本','笔记本','联想','4299');
insert into goods values(0,'svp13226scb 触控超极本','超级本','索尼','7999');
insert into goods values(0,'海飞丝洗发水清爽去油 750ml','洗发水','海飞丝','98');
insert into goods values(0,'ipad air 9.7 英寸平板电脑','平板电脑','苹果','3388');
insert into goods values(0,'轩尼诗（Hennessy）洋酒 新点干邑白兰地 200ml','白酒','轩尼诗','199');
insert into goods values(0,'ideacentre c340 20 英寸一体电脑 ','台式机','联想','3499');
insert into goods values(0,'vostro 3800-r1206 台式电脑','台式机','戴尔','2899');
insert into goods values(0,'imac me086ch/a 21.5 英寸一体电脑','台式机','苹果','9188');
insert into goods values(0,'阿道夫（ADOLPH）轻柔丝滑洗护组合 3 件套','洗发水','阿道夫','3699');
insert into goods values(0,'z220sff f4f06pa 工作站','服务器/工作站','惠普','4288');
insert into goods values(0,'poweredge ii 服务器','服务器/工作站','戴尔','5388');
insert into goods values(0,' 三星（SAMSUNG）C27F390FHC 27 英寸 1800R 曲率 ','显示器','三星','1300');
insert into goods values(0,'戴尔（DELL） U2417H 23.8 英寸四边微边框旋转升降 IPS 屏','显示器','戴尔,'1500');
```

2、查询

1、查询 goods 表中所有的商品

```
select * from goods;
```

2、查询所有产品的平均价格,并且保留两位小数

```
select round(avg(price),2) as avg_price from goods;
```

3、通过子查询来实现，查询所有价格大于平均价格的商品，并且按价格降序排序


```
select id,name,price from goods
where price > (select round(avg(price),2) as avg_price from goods)
order by price desc;
```

4、查询所有 "联想" 的产品

```
select * from goods where brand_name='联想';
```

5、查询价格大于或等于"联想"价格的商品，并且按价格降序排列

```
select id,name,price from goods  where price >= any(select price from g
oods where brand_name = '联想')  order by price desc;
```

6、查询每个产品类型的最低价格的，通过 cate 字段进行分组。

```
select cate,min(price) from goods group by cate;
```

7、查询价格区间在 4500-6500 之间的笔记本

```
select * from goods where price between 4500 and 6500 and cate='笔记本';
```

3、查询数据分表

创建一个商品表

```
create table if not exists goods_cates(
    cate_id int unsigned primary key auto_increment,
    cate_name varchar(40)
);
```

1、查询 goods 表中所有的商品，并且按"类别"分组

```
select cate from goods group by cate;
```

2、将分组后的结果写入到刚才创建的表中

```
insert into goods_cates (cate_name) select cate from goods group by cat
e;
```

3、通过 goods*cates 数据表来更新 goods 表，将 goods 表中的 cate 字段，修改成 goods*cates 的 id 字段

```
update goods as g inner join goods_cates as c on g.cate = c.cate_name
set cate = cate_id;
```

4、字段 brand_name 进行分表。

```
 create table if not exists goods_brands(
    brand_id int unsigned primary key auto_increment,
    brand_name varchar(40)
```

```
);
```

```
insert into goods_brands(brand_name) select brand_name from goods group
 by brand_name;
```

5、通过 goods*brands 数据表来更新 goods 表，将 goods 表中的 barnd*name 字段，修改成 goods_brands 的 id 字段

```
update goods as g inner JOIN goods_brands as j on g.brand_name=j.brand_
name set g.brand_name=j.brand_id;
```

6、查看 goods 表结构，发现 cate， 、brand_name 两个字段都是 varchar 字段，需要修改成 int 类型字段。

```
desc goods;
```

```
alter table goods
change cate cate_id int unsigned not null,
change brand_name brand_id int unsigned not null;
```

7、通过左连接查询所有商品的信息

```
select id,name,cate_name,brand_name,price from goods as g
left join goods_cates as c on g.cate_id = c.cate_id
left join goods_brands as b on g.brand_id = b.brand_id;
```

8、通过右连接查询所有商品的信息

```
select id,name,cate_name,brand_name,price from goods as g
right join goods_cates as c on g.cate_id = c.cate_id
right join goods_brands as b on g.brand_id = b.brand_id;
```

七、 Python DB-API

1、概述

Python 标准数据库接口为 Python DB-API，Python DB-API 为开发人员提供了数据库应用编程接口。 而 PyMySQL 是在 Python3.x 版本中用于连接 MySQL 服务器的一个实现库，Python2 中则使用 mysqldb。

PyMySQL 遵循 Python 数据库 API v2.0 规范，并包含了 pure-Python MySQL 客户端库。

2、安装 PyMySQL

在使用 PyMySQL 之前，我们需要确保 PyMySQL 已安装。

PyMySQL 下载地址：https://github.com/PyMySQL/PyMySQL。

如果还未安装，我们可以使用以下命令安装最新版的 PyMySQL：

```
pip install PyMySQL
```

3、连接数据库

数据库准备，连接数据库之前，请确保已经创建了 python 数据库，以及 students 表

创建 Connection 对象：用于建立与数据库的连接

```
from pymysql import *  # 导入 pymysql 模块
# 创建连接对象 Connection 对象
# host:数据库主机地址
# user:数据库账号
# password: 数据库密码
# database : 需要连接的数据库的名称
# port: mysql 的端口号
# charset: 通信采用编码格式
```

```
conn = connect(host='127.0.0.1', user='root', password='123456',
               database='python', port=3306, charset='utf8');
```

Connection 连接对象拥有的方法

- close 关闭连接，连接数据库跟打开文件一样，操作完成之后需要关闭，否则会占用连接。
- commit()提交，pymysql 默认开启事物，所以每次更新数据库都要提交
- rollback()回滚，事物回滚
- cursor()返回 Cursor 对象，用于执行 sql 语句并获得结果

获取 cursor 对象

```
cur = conn.cursor()  # cursor 对象用于执行 sql 语句
```

cursor 对象拥有的方法

- close()关闭 cursor 对象
- execute(operation [, parameters]) 执行语句，返回受影响的行数，可以执行所有语句
- fetchone()获取查询结果集的第一个行数据，返回一个元组

- fetchall()执行查询时，获取结果集的所有行，一行构成一个元组，再将这些元组装入一个元组返回

插入数据：

```
from pymysql import *
def insert_mysql():
    try:
        # 创建连接对象
        conn = connect(host='192.168.20.82',user='root',password='mysql
',
                       database='python',port=3306,charset='utf8',
                       )
        # 获取 cursor 对象
        cur = conn.cursor()
        try:
            sql = 'insert into students VALUES (0,"露娜",1,"深圳",2,19);
'
            #执行 sql 语句
            res = cur.execute(sql)
            # 打印受影响的行数
            print(res)
            cur.close() # 关闭 cursor 对象
            conn.commit() # 提交事务：todo：涉及到插入，删除，修改，更新操
作都要 commit
        except Exception as e:
            print(e)
        finally:
            # 如果中间执行有错误，导致不能关闭连接对象
            # 所以讲连接对象放到 finally 语句里
            conn.close()
    except Exception as e:
        print(e)

if __name__ == '__main__':
    insert_mysql()
```

查询数据：

```
from pymysql import *
```

```
def select_mysql():
    try:
        # 创建连接对象
        conn = connect(host='192.168.20.82',user='root',password='mysql
',
                       database='python',port=3306,charset='utf8',
```

```
                        )
        # 获取 cursor 对象
        cur = conn.cursor()
        try:
            sql = 'select * from students where id=15;'
            #执行 sql 语句
            cur.execute(sql)
            res = cur.fetchone()  # 获取查询集的第一条数据
            # res = cur.fetchall()  # 获取查询集所有数据
            print(res)
        except Exception as e:
            print(e)
        finally:
            # 如果中间执行有错误，导致不能关闭连接对象
            # 所以讲连接对象放到 finally 语句里
            cur.close()
            conn.close()
    except Exception as e:
        print(e)

if __name__ == '__main__':
    insert_mysql()
```

执行 sql 语句参数化，参数化 sql 语句中使用%s 占位。

execute(operation [parameters]) 执行语句，返回受影响的行数，可以执行所有语句

[parameters] 参数列表

```
from pymysql import *
def select_mysql():
    try:
        # 创建连接对象
        conn = connect(host='192.168.20.82',user='root',password='mysql
',
                       database='python',port=3306,charset='utf8',
                       )
        # 获取 cursor 对象
        cur = conn.cursor()
        try:
            sql = 'select * from students where id=%s and gender= %s;'
# sql 语句中使用%s 占位
            #执行 sql 语句
            cur.execute(sql,[15,0])
            # res = cur.fetchone()  # 获取查询集的第一条数据
            res = cur.fetchall()  # 获取查询集所有数据
            print(res)
            for i in res:  # 将每一条数据打印出来
```

```
                print(i)
        except Exception as e:
            print(e)
        finally:
            # 如果中间执行有错误，导致不能关闭连接对象
            # 所以讲连接对象放到 finally 语句里
            cur.close()
            conn.close()
    except Exception as e:
        print(e)

if __name__ == '__main__':
    insert_mysql()
```

小结

用户与权限管理

MySQL 账户管理概述

创建用户并分配权限

查看、回收权限

修改密码

删除用户

事务

事务概述

事务的特性（ACID）

事务处理

视图和索引

视图

索引

存储过程

存储过程概述

创建存储过程

存储过程中的变量

条件语句

while 循环语句

函数

内置函数

自定义函数

查询实战

Python DB-API

概述

安装 PyMySQL

连接数据库

课后作业

课后问答题

1、那些场景下需要使用事物。

2、请描述事务的四大特性。

3、现在有一个数据库数据量非常大，查询速度很慢，怎么样可以优化查询速度？

4、请写出创建普通索引，组合索引的的 sql 语句，如何查看已经建立好的索引？索引越多越好吗？为什么？

5、请列举在 python 标准 DB-API 中，Connection 对象的常用方法以及作用？

6、请列举在 python 标准 DB-API 中，Cursor 对象方法及作用？

课后实操题

1、假如现在有 10 万条数据需要插入到数据库：

- 1、先创建一个表，保存数据
- 2、请编写 python 代码，在程序中连接 mysql，将数据插入到数据库

2、创建文件 code*insert*stu.py，使用参数化的方式向学生表中添加一条学生信息