# 07 MySQL之视图、触发器、事务、存储过程、函数

MySQL之视图、触发器、事务、存储过程、函数

**阅读目录**



MySQL这个软件想将数据处理的所有事情，能够在mysql这个层面上全部都做了，也就是说它想要完成一件事，以后想开发的人，例如想写python程序的人，你就专门的写你自己的python程序，以后凡是关于数据的增删改查，全部都在MySQL里面完成，也就是说它想实现一个数据处理与应用程序的一个完全的解耦和状态，比如说，如果我是个应用程序员，我想要查询数据，我不需要自己写sql语句，只需要调用mysql封装好的一些功能，直接调用这个功能就可以了，之前我们使用sql来进行数据的增删改查，其实sql也可以算作一个开发语言，有专门招数据库开发的岗位，也就是说mysql想做这么一个事儿，以后啊，专门有人写应用程序的开发，专门有人来写sql，来开发sql部分，在数据库层面根据应用层的程序员的要求，把sql语句全部写好，各种复杂的需求全部帮你封装好，封装成一个一个的功能，应用程序开发程序员在根据自己的需求来使用这些功能，直接调用就可以了，这是mysql想要完成的事情，但是咱们以后做开发，一般不会这么搞，一般招聘需求里面都会有一项是要会sql，浅显的说是因为花最少的钱，做最多的事儿，但是往深了说是因为公司里面一般不会用这些内置的功能去sql的工作，至于为什么，咱们学完mysql之后再说吧~~~

# 一 视图

视图是一个虚拟表（非真实存在），是跑到内存中的表，真实表是硬盘上的表，怎么就得到了虚拟表，就是你查询的结果，只不过之前我们查询出来的虚拟表，从内存中取出来显示在屏幕上，内存中就没有了这些表的数据，但是下次我要是想用这个虚拟表呢，没办法，只能重新查一次，每次都要重新查。其本质是【根据SQL语句获取动态的数据集，并为其命名】，用户使用时只需使用【名称】即可获取结果集，可以将该结果集当做表来使用。如果我们想查询一些有关联的表，比如我们前面的老师学生班级什么的表，我可能需要几个表联合查询的结果，但是这几张表在硬盘上是单独存的，所以我们需要通过查询的手段，将这些表在内存中拼成一个虚拟表，然后是不是我们再基于虚拟表在进行进一步的查询，然后我们如果以后想重新再查一下这些关系数据，还需要对硬盘上这些表进行再次的重新加载到内容，联合成虚拟表，然后再筛选等等的操作，意味着咱们每次都在写重复的sql语句，那有没有好的方法啊，其实很简单，我们把重复用的这些sql逻辑封装起来，然后下次使用的时候直接调用这个封装好的操作就可以了，这个封装起来的操作就类似我们下面要说的视图

为什么要用视图：使用视图我们可以把查询过程中的临时表摘出来，保存下来，用视图去实现，这样以后再想操作该临时表的数据时就无需重写复杂的sql了，直接去视图中查找即可，但视图有明显地效率问题，并且视图是存放在数据库中的，如果我们程序中使用的sql过分依赖数据库中的视图，即强耦合，那就意味着扩展sql极为不便，因此并不推荐使用

临时表应用举例：

```
#两张有关系的表
mysql> select * from course;
+-----+--------+------------+
| cid | cname  | teacher_id |
+-----+--------+------------+
|   1 | 生物   |          1 |
|   2 | 物理   |          2 |
|   3 | 体育   |          3 |
|   4 | 美术   |          2 |
+-----+--------+------------+
4 rows in set (0.00 sec)

mysql> select * from teacher;
+-----+-----------------+
```

```
| tid | tname          |
+-----+----------------+
|   1 | 张磊老师       |
|   2 | 李平老师       |
|   3 | 刘海燕老师     |
|   4 | 朱云海老师     |
|   5 | 李杰老师       |
+-----+----------------+
5 rows in set (0.00 sec)

#查询李平老师教授的课程名
mysql> select cname from course where teacher_id = (select tid from teacher where tname='李平老师'); #子查询的方式
+--------+
| cname  |
+--------+
| 物理   |
| 美术   |
+--------+
2 rows in set (0.00 sec)

#子查询出临时表，作为teacher_id等判断依据
select tid from teacher where tname='李平老师'
```

**一 创建视图**

```
#语法：CREATE VIEW 视图名称 AS  SQL语句
create view teacher_view as select tid from teacher where tname='李平老师';
```

```
mysql> show tables;
+---------------+
| Tables_in_crm |
+---------------+
| class |
| course |
| score |
| student |
| teacher |
| teacher_view |  #看这里
+---------------+
```

```
mysql> desc teacher_view;  #有表结构
+-------+---------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| tid | int(11) | NO | | 0 | |
+-------+---------+------+-----+---------+-------+
1 row in set (0.04 sec)
```

```
mysql> select * from teacher_view; #有表数据
+-----+
| tid |
+-----+
| 2 |
```

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

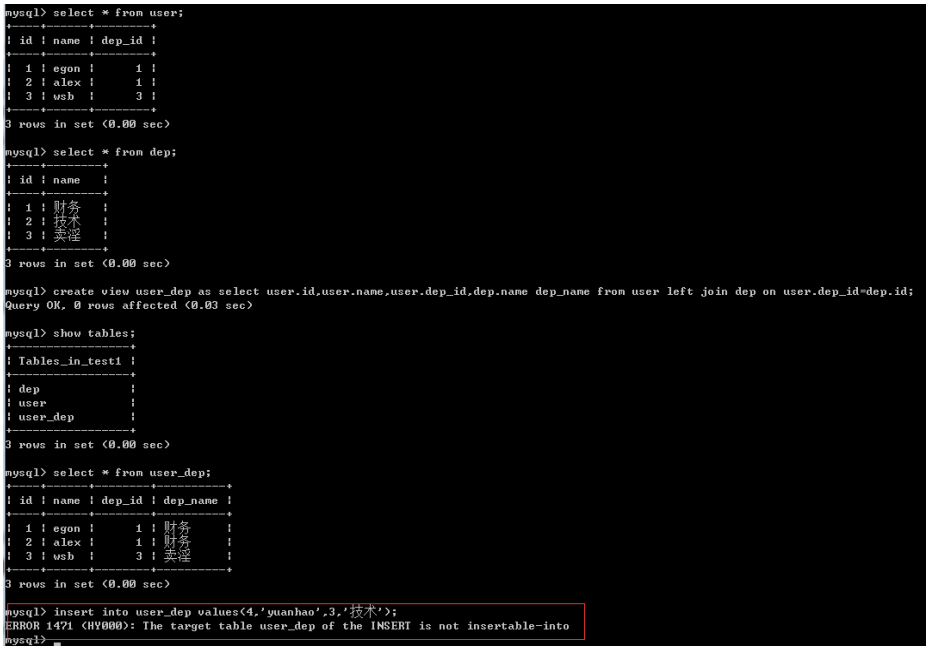
```
注意：但是在硬盘上你找到自己的mysql安装目录里面的data文件夹里面的对应的那个库的文件夹，这个文件夹里面存着咱们的表信息
#于是查询李平老师教授的课程名的sql可以改写为
mysql> select cname from course where teacher_id = (select tid from teacher_view);
+---------+
| cname   |
+---------+
| 物理    |
| 美术    |
+---------+
2 rows in set (0.00 sec)

#！！！注意注意注意：
#1. 使用视图以后就无需每次都重写子查询的sql，开发的时候是方便了很多，但是这么效率并不高，还不如我们写子查询的效率高

#2. 而且有一个致命的问题：视图是存放到数据库里的，如果我们程序中的sql过分依赖于数据库中存放的视图，那么意味着，一旦sql
```

**二 使用视图**

```
#修改视图，原始表也跟着改
mysql> select * from course;
+-----+--------+------------+
| cid | cname  | teacher_id |
+-----+--------+------------+
|   1 | 生物   |          1 |
|   2 | 物理   |          2 |
|   3 | 体育   |          3 |
|   4 | 美术   |          2 |
+-----+--------+------------+
4 rows in set (0.00 sec)

mysql> create view course_view as select * from course; #创建表course的视图
Query OK, 0 rows affected (0.52 sec)

mysql> select * from course_view;
+-----+--------+------------+
| cid | cname  | teacher_id |
+-----+--------+------------+
|   1 | 生物   |          1 |
|   2 | 物理   |          2 |
|   3 | 体育   |          3 |
|   4 | 美术   |          2 |
+-----+--------+------------+
4 rows in set (0.00 sec)

mysql> update course_view set cname='xxx'; #更新视图中的数据
Query OK, 4 rows affected (0.04 sec)
Rows matched: 4  Changed: 4  Warnings: 0

mysql> insert into course_view values(5,'yyy',2); #往视图中插入数据
Query OK, 1 row affected (0.03 sec)

mysql> select * from course; #发现原始表的记录也跟着修改了
+-----+-------+------------+
| cid | cname | teacher_id |
```

```
+-----+-------+------------+
|   1 | xxx   |          1 |
|   2 | xxx   |          2 |
|   3 | xxx   |          3 |
|   4 | xxx   |          2 |
|   5 | yyy   |          2 |
+-----+-------+------------+
5 rows in set (0.00 sec)
```

我们不应该修改视图中的记录，而且在涉及多个表的情况下是根本无法修改视图中的记录的，如下图




**三 修改视图**

语法：ALTER VIEW 视图名称 AS SQL语句，这基本就和删掉视图重新创建一个视图的过程是一样的，修改视图没什么好讲的，这里就

```
mysql> alter view teacher_view as select * from course where cid>3;
Query OK, 0 rows affected (0.04 sec)

mysql> select * from teacher_view;
+-----+-------+------------+
| cid | cname | teacher_id |
+-----+-------+------------+
|   4 | xxx   |          2 |
|   5 | yyy   |          2 |
+-----+-------+------------+
2 rows in set (0.00 sec)
```

**四 删除视图**

语法：DROP VIEW 视图名称

```
DROP VIEW teacher_view
```

# 二 触发器

使用触发器可以定制用户对某一张表的数据进行【增、删、改】操作时前后的行为，注意：没有查询，在进行增删改操作的时候，触发的某个操作，称为触发器，也就是增删改的行为触发另外的一种行为，触发的行为无非就是sql语句的事情，及自动运行另外一段sql语句。来看一下触发器怎么来创建：

**一 创建触发器**

```
# 插入前
CREATE TRIGGER tri_before_insert_tb1 BEFORE INSERT ON tb1 FOR EACH ROW
BEGIN #begin和end里面写触发器要做的sql事情，注意里面的代码缩进，并且给触发器起名字的时候，名字的格式最好这样写，有表
    ...
END

# 插入后
CREATE TRIGGER tri_after_insert_tb1 AFTER INSERT ON tb1 FOR EACH ROW
BEGIN
    ...
END

# 删除前
CREATE TRIGGER tri_before_delete_tb1 BEFORE DELETE ON tb1 FOR EACH ROW
BEGIN
    ...
END

# 删除后
CREATE TRIGGER tri_after_delete_tb1 AFTER DELETE ON tb1 FOR EACH ROW
BEGIN
    ...
END

# 更新前
CREATE TRIGGER tri_before_update_tb1 BEFORE UPDATE ON tb1 FOR EACH ROW
BEGIN
    ...
END

# 更新后
CREATE TRIGGER tri_after_update_tb1 AFTER UPDATE ON tb1 FOR EACH ROW
BEGIN
    ...
END
```

插入后触发触发器：

```
#准备表
CREATE TABLE cmd（ #这是一张指令信息表，你在系统里面执行的任何的系统命令都在表里面写一条记录
    id INT PRIMARY KEY auto_increment，#id
    USER CHAR (32)，#用户
    priv CHAR (10)，#权限
    cmd CHAR (64)， #指令
    sub_time datetime, #提交时间
    success enum ('yes', 'no') #是否执行成功，0代表执行失败
);
```

```
CREATE TABLE errlog ( #指令执行错误的信息统计表，专门提取上面cmd表的错误记录
    id INT PRIMARY KEY auto_increment, #id
    err_cmd CHAR (64)，#错误指令
    err_time datetime   #错误命令的提交时间
);
#现在的需求是：不管正确或者错误的cmd，都需要往cmd表里面插入，然后，如果是错误的记录，还需要往errlog表里面插入一条记录
#创建触发器
delimiter //       (或者写$$，其他符号也行，但是不要写mysql不能认识的，知道一下就行了)，delimiter 是告诉mysql，遇到这句
CREATE TRIGGER tri_after_insert_cmd AFTER INSERT ON cmd FOR EACH ROW                #在你cmd表插入一条记录之后触发的。
BEGIN          #每次给cmd插入一条记录的时候，都会被mysql封装成一个对象，叫做NEW，里面的字段都是这个NEW的属性
    IF NEW.success = 'no' THEN          #mysql里面是可以写这种判断的，等值判断只有一个等号，然后写then
          INSERT INTO errlog(err_cmd, err_time) VALUES(NEW.cmd, NEW.sub_time)；    #必须加分号，并且注意，我们必须用de
      END IF；      #然后写end if，必须加分号
END//      #只有遇到//这个完成的sql才算结束
delimiter；       #然后将mysql的结束符改回为分号


#往表cmd中插入记录，触发触发器，根据IF的条件决定是否插入错误日志
INSERT INTO cmd (
    USER,
    priv,
    cmd,
    sub_time,
    success
)
VALUES
    ('chao','0755','ls -l /etc',NOW(),'yes'),
    ('chao','0755','cat /etc/passwd',NOW(),'no'),
    ('chao','0755','useradd xxx',NOW(),'no'),
    ('chao','0755','ps aux',NOW(),'yes');


#查询错误日志，发现有两条
mysql> select * from errlog;
+----+-----------------+---------------------+
| id | err_cmd         | err_time            |
+----+-----------------+---------------------+
|  1 | cat /etc/passwd | 2017-09-14 22:18:48 |
|  2 | useradd xxx     | 2017-09-14 22:18:48 |
+----+-----------------+---------------------+
2 rows in set (0.00 sec)
```



**特别的：NEW表示即将插入的数据行，OLD表示即将删除的数据行。**

**二 使用触发器**

触发器无法由用户直接调用，而由对表的【增/删/改】操作被动引发的。

**三 删除触发器**

```
drop trigger tri_after_insert_cmd;
```

## 三 事务

事务用于将某些操作的多个SQL作为原子性操作，也就是这些sql语句要么同时成功，要么都不成功，事务的其他特性在我第一篇博客关于事务的介绍里面有，这里就不多做介绍啦，一旦有某一个出现错误，即可回滚到原来的状态，

从而保证数据库数据完整性。

简单来说：我给一个姑娘转账，姑娘那儿收到了200，你的账户上扣了200，这两个操作是不是两个sql语句，这两个sql语句是你的应用程序发给mysql服务端的，并且这两个sql语句都要一起执行，不然数据就错了，你想想是不是。并且如果你通过应用程序发送这两条sql的时候，由于网络问题，你只发送了一个sql过来，那只有一个账户改了数据，另外一个没改，那数据是不是就出错了啊。这就是事务要完成的事情。

```
create table user(
id int primary key auto_increment,
name char(32),
balance int
);

insert into user(name,balance)
values
('wsb',1000),
('chao',1000),
('ysb',1000);

#原子操作
start transaction;
update user set balance=900 where name='wsb'; #买支付100元
update user set balance=1010 where name='chao'; #中介拿走10元
update user set balance=1090 where name='ysb'; #卖家拿到90元
commit;  #只要不进行commit操作，就没有保存下来，没有刷到硬盘上

#出现异常，回滚到初始状态
start transaction;
update user set balance=900 where name='wsb'; #买支付100元
update user set balance=1010 where name='chao'; #中介拿走10元
uppdate user set balance=1090 where name='ysb'; #卖家拿到90元,出现异常没有拿到
rollback;  #如果上面三个sql语句出现了异常，就直接rollback，数据就直接回到原来的状态了。但是执行了commit之后，rollback这
commit;#通过存储过程来捕获异常：(shit！，写存储过程的是，注意每一行都不要缩进！！！按照下面的缩进来写，居然让我翻车了！
```

```
delimiter //
create PROCEDURE p5()
BEGIN
DECLARE exit handler for sqlexception
BEGIN
rollback;
END;

START TRANSACTION;
update user set balance=900 where name='wsb'; #买支付100元
update user set balance=1010 where name='chao'; #中介拿走10元
#update user2 set balance=1090 where name='ysb'; #卖家拿到90元
update user set balance=1090 where name='ysb'; #卖家拿到90元
COMMIT;

END //
delimiter ;
```

```

```

```
mysql> select * from user;
+----+------+---------+
| id | name | balance |
+----+------+---------+
|  1 | wsb  |    1000 |
|  2 | chao |    1000 |
```

```
| 3 | ysb |   1000 |
+----+------+----------+
3 rows in set (0.00 sec)
```

# 四 存储过程

## 一 介绍

存储过程包含了一系列可执行的sql语句，存储过程存放于MySQL中，通过调用它的名字可以执行其内部的一堆sql。到目前为止，我们上面学的视图、触发器、事务等为我们简化了应用程序级别写sql语句的复杂程度，让我们在应用程序里面写sql更简单方便了，但是我们在应用程序上还是需要自己写sql的，而我们下面要学的存储过程，它是想让我们的应用程序不需要再写sql语句了，所有的sql语句，全部放到mysql里面，被mysql封装成存储过程，说白了它就是一个功能，这个功能对应着一大堆的sql语句，这些语句里面可以包括我们前面学的视图啊、触发器啊、事务啊、等等的内容，也就是说存储过程其实是什么？是一堆sql的集合体，可以直接用mysql里面提供的一堆功能，有了存储过程以后，它的好处是我项目逻辑中需要的各种查询都可以让DBA或者你自己封装到存储过程里面，以后使用的时候直接调用存储过程名就可以了，在开发应用的时候就简单了，就不要应用程序员进行sql语句的开发了，但是你想如果你真的这么做了，确实很有好处，简单很多，应用程序的开发和数据库sql语句的开发，完全的解耦了，这样，专门的人做专门的事情，专门招一个应用开发的人开发应用程序，招一个开发型DBA，会sql的开发，他把sql写完之后，封装成一个个的存储过程，给应用程序员用就行了，对不对，这个DBA就不单纯的是管理数据库系统了，还需要会写sql语句，那这样你的应用程序开发的效率就高了，运行效率也提高了，你开发应用程序的时候如果写了一堆的sql语句，这些语句是不是要通过网络传输，传输到mysql服务端来执行，然后将结果返回给你的应用程序，那么在传输的时候，你说好多的sql语句和简单的一个存储过程的名字，哪个传输的速度快，哪个发送给服务端的速度快，当然是单纯的一个存储过程的名字更快。

所以摆在你面前有两种开发模式：

第一种是招一个会开发应用程序的并且这个人还要会sql开发，这样的人既写应用程序，还写sql语句，这种情况你可以招两个人，一个是前面说的，还有一个是数据库管理人员，单纯只会管理数据库的而不会sql开发的人，这样好招人，工资也不高。（应用程序员-->只需要开发应用程序的逻辑 。  sql开发人员-->编写存储过程）

第二种：招一个应用程序开发的，只需要会应用程序级别的开发，再招一个会sql开发的DBA。（应用程序员-->开发应用+写原生sql 。 数据库人员负责维护数据库的正常运行）

我们比较一下这两种的开发模式：第二种：解耦和，开发效率高，运行效率也高，所以以后最好采用第一种开发模式，哈哈，是不是神反转，原因是什么呢，钱只是一个方面，主要还是因为以后如果你想扩展，那就很不方便了，为什么呢，因为通常sql开发人员，不如你的应用程序员更懂你的业务逻辑，一旦你要扩展一个功能，还需要跨部门沟通，导致这种工作方式受限的不只是技术层面了，这种方式在技术层面肯定是效率高的，但是要考虑人为因素，还有成本方面的考虑，所以通常咱们以后做开发，不要想着会有人给你写sql，需要你自己写的很熟练。这样，你一个部门就能搞定这两件事情。

第二种方式其实也比较麻烦，你开发程序员自己需要写sql，并且写出来的sql还存在效率问题，那么有没有一种方式可以不让开发程序员自己再写sql了，搞一个封装程度更高的东西让你来调用，有没有这种方式呢？有，就是第三种方式

第三种：应用程序除了开发应用程序的逻辑，不需要编写原生sql，只需要使用别人写好的框架，基于框架来处理数据，框架提供的功能是ORM：对象关系映射，和对象有关系，就是在应用程序里面，只需要定义一堆的类，每个类对应数据库里面的一张表，这个类一实例化，也就是一个类对象对应表里面的一条记录，得到对象以后，这个对象除了有数据之外，还有处理该条信息的方法，增删改查都有了，全都封装成了对象的一个一个的方法了，意味着你以后再想进行查询，就没必要写原生sql了，直接基于面向对象的思想来处理类与对象就行了，但是这种方式本质上还是使用了原生sql，只不过对于应用程序员来说，你不用直接写sql了，别人写好的ORM框架就帮你处理这件事儿了，帮你把你调用的那些接口方法和你传入的参数等等帮你转换为了原生sql，然后再往mysql里面提交。所以这种方式和第二种方式有些类似，但是比第二种方式要好(前提是第二种方式应用程序员的sql水平比较low的情况下，一般会比较low)：

这种方式的优点：应用程序员不需要再写原生的sql了，这意味着开发效率比第二种要高，同时还兼顾了第二种方式扩展性高的好处，因为本质上还是原生sql

缺点是：执行效率还不如第二种方式高，因为你现在再想运行需要做什么事 情，首先你想，你程序里面用的是别人写好的ORM框架或者模块，你的sql要想执行，你需要做什么事儿，你的ORM框架需要把类或者类对象先翻译成原生的sql，再沿着网络发到mysql服务端，中间对了一个转换的过程，所以执行效率其实连方式二都比不上。

总结：其实单单从技术层面上看，第一种方式肯定是最好的，开发和执行效率是最高的，扩展性单纯技术层面来看也比较高，所以单单从技术层面来考虑，这种方式肯定是优选的。但是就目前的现状而言，多数还是需要你应用程序员

既做应用逻辑的开发，还要会原生sql的开发，所以应该尽可能的不让mysql来做了，所有关于数据的增删改查都交给应用程序级别来做，在应用程序中写原生的sql，这也是第二种方式和第三种方式的一个共性，所有事情基本都交给应用程序级别来做，mysql级别基本不做sql的开发，这样扩展性也好一些，因为所有的事情都交给你应用程序的开发部门来做了，自己部门内部进行扩展还是扩展性不错的，所以咱们一般从后面两种方式来选择，那么后面两种选哪一种呢？第二种执行效率比第三种高，因为比第三种少了一步类对象转换为sql的过程。第三种开发效率高，不要应用程序员再写原生的sql了。所以具体选哪一种，看你们自己公司的情况，需要快速开发，就找第三种，如果自己程序员的sql写的很溜(又快又优)，那么找第二种，一般大公司采用的第一种多一些，部门分工非常的明确，等你们大家学到Django框架，你们就会接触ORM啦~~~学完MySQL之后，我们在学框架之前，带大家咱们自己开发一个简单的ORM框架，能够在你自己的应用程序中使用的ORM，写这个ORM框架很重要，对你以后的框架和项目的学习很有指导意义，说哆啦，我们继续说存储过程：

使用存储过程的优点：

```
#1. 用于替代程序写的SQL语句，实现程序与sql解耦

#2. 基于网络传输，传别名的数据量小，而直接传sql数据量大
```

使用存储过程的缺点：

```
#1. 程序员扩展功能不方便
```

**上面一大堆话的总结：程序与数据库结合使用的三种方式**

```
#方式一：
    MySQL：存储过程
    程序：调用存储过程

#方式二：
    MySQL：
    程序：纯SQL语句

#方式三：
    MySQL:
    程序：类和对象，即ORM（本质还是纯SQL语句）
```

**二 创建简单存储过程（无参）**

```
delimiter //
create procedure p1()
BEGIN
    select * from blog;
    INSERT into blog(name,sub_time) values("xxx",now());
END //
delimiter ;

#在mysql中调用
call p1(); #类似于MySQL的函数，但不是函数昂，别搞混了，MySQL的函数(count()\max()\min()等等)都是放在sql语句里面用的，不
#MySQL的视图啊触发器啊if判断啊等等都能在存储过程里面写，这是一大堆的sql的集合体，都可以综合到这里面
#在python中基于pymysql调用
```

```
cursor.callproc('p1')
print(cursor.fetchall())
```

另外：存储过程是可以传参数的，看下面的内容

**三 创建存储过程（有参）**

```
对于存储过程，可以接收参数，其参数有三类：

#in         仅用于传入参数用
#out        仅用于返回值用
#inout      既可以传入又可以当作返回值
```

in:传入参数：

```
delimiter //
create procedure p2(
    in n1 int，#n1参数是需要传入的，也就是接收外部数据的，并且这个数据必须是int类型
    in n2 int
)
BEGIN

    select * from blog where id > n1；#直接应用变量
END //
delimiter ;
#调用存储过程的两种方式：或者说是两个地方吧
      #在mysql中调用
      call p2(3,2)

      #在python中基于pymysql调用
      cursor.callproc('p2',(3,2))
      print(cursor.fetchall())
```

#通过存储过程的传参来看，也能体现出我们学习的Python的灵活性，传参不需要指定类型，也不需要声明这个参数是传入的还是返回出来的，参数既可以传入，这个参数也可以直接通过return返回。

out：返回值：

```
#查看存储过程的一些信息：show create procedure p3; #查看视图啊、触发器啊都这么看，还可以用\G，show create procedure
create procedure p3(
    in n1 int,
    out res int
)
BEGIN
    select * from blog where id > n1;
    set res = 1；#我在这里设置一个res=1，如果上面的所有sql语句全部正常执行了，那么这一句肯定也就执行了，那么此时res=1，
END //
delimiter ;

#在mysql中调用
set @res=0; #这是MySQL中定义变量名的固定写法(set @变量名=值)，可以自己规定好，0代表假（执行失败），1代表真（执行成功
call p3(3,@res);#注意：不要这样写：call p3（3，1），这样out的参数值你写死了，没法确定后面这个1是不是成功了，也就是说随
select @res; #看一下这个结果，就知道这些sql语句是不是执行成功了，大家明白了吗~~~
```

```
#在python中基于pymysql调用，在python中只需要知道存储过程的名字就行了
cursor.callproc('p3',(3,0)) #0相当于set @res=0，为什么这里这个out参数可以写常数0啊，因为你用的pymysql,人家会帮你搞定，
print(cursor.fetchall()) #查询select的查询结果

cursor.execute('select @_p3_0,@_p3_1;') #@_p3_0代表第一个参数，@_p3_1代表第二个参数，即返回值
print(cursor.fetchall())#别忘了关掉：cursor.close()conn.close()#注意昂：存储过程在哪个库里面建的，就只能在哪个库里面用
```

inout：既可传入又可以返回值:

```
delimiter //
create procedure p4(
   inout n1 int
)
BEGIN
   select * from blog where id > n1;
   set n1 = 1;
END //
delimiter ;

#在mysql中调用
set @x=3;
call p4(@x);
select @x;


#在python中基于pymysql调用
cursor.callproc('p4',(3,))
print(cursor.fetchall()) #查询select的查询结果

cursor.execute('select @_p4_0;')
print(cursor.fetchall())
```

存储过程结合事务来写：

```
delimiter //
        create procedure p4(
           out status int
        )
        BEGIN
           1. 声明如果出现异常则执行{
              set status = 1;
              rollback;
           }

           开始事务
              -- 由秦兵账户减去100
              -- 方少伟账户加90
              -- 张根账户加10
              commit;
           结束

           set status = 2;
```

```
        END //
        delimiter ;

#实现
delimiter //
create PROCEDURE p5(
   OUT p_return_code tinyint
)
BEGIN
   DECLARE exit handler for sqlexception   #声明如果一旦出现异常则执行下面的这个begin和end里面的操作
   BEGIN
      -- ERROR   #--是什么啊，忘了吧，是注释的意思，就告诉你后面是对错误的处理
      set p_return_code = 1；#将out返回值改为1了，这是你自己规定的，1表示出错了
      rollback；#回滚事务
   END;

   DECLARE exit handler for sqlwarning  #声明了出现警告信息之后你的操作行为
   BEGIN
      -- WARNING
      set p_return_code = 2;
      rollback;
   END;

   START TRANSACTION；#其实咱们这个存储过程里面就是执行这个事务，并且一直检测着这个事务，一旦出错或者出现警告，就rc
      DELETE from tb1; #事务里面的任何一条sql执行失败或者执行出现警告，都会执行上面我们声明的那些对应的操作，如果没有任
      insert into blog(name,sub_time) values('yyy',now())；#拿我的代码进行测试的时候，别忘了改成你自己库里的表，还有表里面
   COMMIT;

   -- SUCCESS
   set p_return_code = 0; #0代表执行成功

END //
delimiter ;

#在mysql中调用存储过程
set @res=123;
call p5(@res);
select @res;

#在python中基于pymysql调用存储过程
cursor.callproc('p5',(123,)) #注意后面这个参数是个元祖，别忘了逗号，按照我们上面规定的，上面有三个值0，1，2：0成功、1失
print(cursor.fetchall()) #查询select的查询结果

cursor.execute('select @_p5_0;')
print(cursor.fetchall())#执行成功以后，查看一下结果就能看到执行后的值了
```

**四 执行存储过程**

在MySQL中执行存储过程：

```
-- 无参数
call proc_name()

-- 有参数，全in
call proc_name(1,2)

-- 有参数，有in，out，inout
```

```
set @t1=0;
set @t2=3;
call proc_name(1,2,@t1,@t2)

执行存储过程
```

在python中基于pymysql来执行存储过程：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
import pymysql

conn = pymysql.connect(host='127.0.0.1', port=3306, user='root', passwd='123', db='t1')
cursor = conn.cursor(cursor=pymysql.cursors.DictCursor)
# 执行存储过程
cursor.callproc('p1', args=(1, 22, 3, 4))
# 获取执行完存储的参数
cursor.execute("select @_p1_0,@_p1_1,@_p1_2,@_p1_3")
result = cursor.fetchall()

#conn.commit()
cursor.close()
conn.close()


print(result)
```

**五 删除存储过程**

```
drop procedure proc_name;
```

# 五 函数

MySQL中提供了许多内置函数，但是注意，这些函数只能在sql语句中使用，不能单独调用昂，例如：其实下面的有些函数我们都已经用过了，其他的如果你们用到了，咱们再过来查吧，好不?

```
一、数学函数
    ROUND(x,y)
        返回参数x的四舍五入的有y位小数的值

    RAND()
        返回 0 到 1 内的随机值,可以通过提供一个参数(种子)使RAND()随机数生成器生成一个指定的值。

二、聚合函数(常用于GROUP BY从句的SELECT查询中)
    AVG(col)返回指定列的平均值
    COUNT(col)返回指定列中非NULL值的个数
    MIN(col)返回指定列的最小值
    MAX(col)返回指定列的最大值
    SUM(col)返回指定列的所有值之和
```

GROUP_CONCAT(col) 返回由属于一组的列值连接组合而成的结果

三、字符串函数

CHAR_LENGTH(str)
返回值为字符串str 的长度，长度的单位为字符。一个多字节字符算作一个单字符。
CONCAT(str1,str2,...)
字符串拼接
如有任何一个参数为NULL ，则返回值为 NULL。
CONCAT_WS(separator,str1,str2,...)
字符串拼接（自定义连接符）
CONCAT_WS()不会忽略任何空字符串。 (然而会忽略所有的 NULL）。

CONV(N,from_base,to_base)
进制转换
例如:
SELECT CONV('a',16,2); 表示将 a 由16进制转换为2进制字符串表示

FORMAT(X,D)
将数字X 的格式写为'#,###,###.##',以四舍五入的方式保留小数点后 D 位， 并将结果以字符串的形式返回。若 D 为 0, 则返回结
例如:
SELECT FORMAT(12332.1,4); 结果为： '12,332.1000'
INSERT(str,pos,len,newstr)
在str的指定位置插入字符串
pos：要替换位置其实位置
len：替换的长度
newstr：新字符串
特别的:
如果pos超过原字符串长度，则返回原字符串
如果len超过原字符串长度，则由新字符串完全替换
INSTR(str,substr)
返回字符串 str 中子字符串的第一个出现位置。

LEFT(str,len)
返回字符串str 从开始的len位置的子序列字符。

LOWER(str)
变小写

UPPER(str)
变大写

REVERSE(str)
返回字符串 str ， 顺序和字符顺序相反。

SUBSTRING(str,pos) , SUBSTRING(str FROM pos) SUBSTRING(str,pos,len) , SUBSTRING(str FROM pos FOR len)
不带有len 参数的格式从字符串str返回一个子字符串，起始于位置 pos。带有len参数的格式从字符串str返回一个长度同len字符

```
mysql> SELECT SUBSTRING('Quadratically',5);
    -> 'ratically'

mysql> SELECT SUBSTRING('foobarbar' FROM 4);
    -> 'barbar'

mysql> SELECT SUBSTRING('Quadratically',5,6);
    -> 'ratica'

mysql> SELECT SUBSTRING('Sakila', -3);
    -> 'ila'

mysql> SELECT SUBSTRING('Sakila', -5, 3);
    -> 'aki'
```

```
mysql> SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
    -> 'ki'
```

四、日期和时间函数

CURDATE()或CURRENT_DATE() 返回当前的日期
CURTIME()或CURRENT_TIME() 返回当前的时间
DAYOFWEEK(date) 返回date所代表的一星期中的第几天(1~7)
DAYOFMONTH(date) 返回date是一个月的第几天(1~31)
DAYOFYEAR(date) 返回date是一年的第几天(1~366)
DAYNAME(date) 返回date的星期名，如：SELECT DAYNAME(CURRENT_DATE);
FROM_UNIXTIME(ts,fmt) 根据指定的fmt格式，格式化UNIX时间戳ts
HOUR(time) 返回time的小时值(0~23)
MINUTE(time) 返回time的分钟值(0~59)
MONTH(date) 返回date的月份值(1~12)
MONTHNAME(date) 返回date的月份名，如：SELECT MONTHNAME(CURRENT_DATE);
NOW() 返回当前的日期和时间
QUARTER(date) 返回date在一年中的季度(1~4)，如SELECT QUARTER(CURRENT_DATE);
WEEK(date) 返回日期date为一年中第几周(0~53)
YEAR(date) 返回日期date的年份(1000~9999)

重点:
DATE_FORMAT(date,format) 根据format字符串格式化date值

```
mysql> SELECT DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y');
 -> 'Sunday October 2009'
mysql> SELECT DATE_FORMAT('2007-10-04 22:23:00', '%H:%i:%s');
 -> '22:23:00'
mysql> SELECT DATE_FORMAT('1900-10-04 22:23:00',
 ->                '%D %y %a %d %m %b %j');
 -> '4th 00 Thu 04 10 Oct 277'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
 ->                '%H %k %I %r %T %S %w');
 -> '22 22 10 10:23:00 PM 22:23:00 00 6'
mysql> SELECT DATE_FORMAT('1999-01-01', '%X %V');
 -> '1998 52'
mysql> SELECT DATE_FORMAT('2006-06-00', '%d');
 -> '00'
```

五、加密函数

MD5()
计算字符串str的MD5校验和
PASSWORD(str)
返回字符串str的加密版本，这个加密过程是不可逆转的，和UNIX密码加密过程使用不同的算法。

六、控制流函数

CASE WHEN[test1] THEN [result1]...ELSE [default] END
如果testN是真，则返回resultN，否则返回default
CASE [test] WHEN[val1] THEN [result]...ELSE [default]END
如果test和valN相等，则返回resultN，否则返回default

IF(test,t,f)
如果test是真，返回t；否则返回f

IFNULL(arg1,arg2)
如果arg1不是空，返回arg1，否则返回arg2

NULLIF(arg1,arg2)
如果arg1=arg2返回NULL；否则返回arg1

七、控制流函数小练习

#7.1、准备表，将下面这些内容保存为一个.txt文件或者.sql，然后通过navicat的运行sql文件的功能导入到数据库中，还记得吗?

```
/*
Navicat MySQL Data Transfer

Source Server         : localhost_3306
Source Server Version : 50720
Source Host           : localhost:3306
Source Database       : student

Target Server Type    : MYSQL
Target Server Version : 50720
File Encoding         : 65001

Date: 2018-01-02 12:05:30
*/

SET FOREIGN_KEY_CHECKS=0;

-- ----------------------------
-- Table structure for course
-- ----------------------------
DROP TABLE IF EXISTS `course`;
CREATE TABLE `course` (
  `c_id` int(11) NOT NULL,
  `c_name` varchar(255) DEFAULT NULL,
  `t_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`c_id`),
  KEY `t_id` (`t_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of course
-- ----------------------------
INSERT INTO `course` VALUES ('1', 'python', '1');
INSERT INTO `course` VALUES ('2', 'java', '2');
INSERT INTO `course` VALUES ('3', 'linux', '3');
INSERT INTO `course` VALUES ('4', 'web', '2');

-- ----------------------------
-- Table structure for score
-- ----------------------------
DROP TABLE IF EXISTS `score`;
CREATE TABLE `score` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `s_id` int(10) DEFAULT NULL,
  `c_id` int(11) DEFAULT NULL,
  `num` double DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=12 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of score
-- ----------------------------
INSERT INTO `score` VALUES ('1', '1', '1', '79');
INSERT INTO `score` VALUES ('2', '1', '2', '78');
INSERT INTO `score` VALUES ('3', '1', '3', '35');
INSERT INTO `score` VALUES ('4', '2', '2', '32');
INSERT INTO `score` VALUES ('5', '3', '1', '66');
INSERT INTO `score` VALUES ('6', '4', '2', '77');
INSERT INTO `score` VALUES ('7', '4', '1', '68');
INSERT INTO `score` VALUES ('8', '5', '1', '66');
INSERT INTO `score` VALUES ('9', '2', '1', '69');
INSERT INTO `score` VALUES ('10', '4', '4', '75');
```

```
INSERT INTO `score` VALUES ('11', '5', '4', '66.7');

-- ----------------------------
-- Table structure for student
-- ----------------------------
DROP TABLE IF EXISTS `student`;
CREATE TABLE `student` (
  `s_id` varchar(20) NOT NULL,
  `s_name` varchar(255) DEFAULT NULL,
  `s_age` int(10) DEFAULT NULL,
  `s_sex` char(1) DEFAULT NULL,
  PRIMARY KEY (`s_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of student
-- ----------------------------
INSERT INTO `student` VALUES ('1', '鲁班', '12', '男');
INSERT INTO `student` VALUES ('2', '貂蝉', '20', '女');
INSERT INTO `student` VALUES ('3', '刘备', '35', '男');
INSERT INTO `student` VALUES ('4', '关羽', '34', '男');
INSERT INTO `student` VALUES ('5', '张飞', '33', '女');

-- ----------------------------
-- Table structure for teacher
-- ----------------------------
DROP TABLE IF EXISTS `teacher`;
CREATE TABLE `teacher` (
  `t_id` int(10) NOT NULL,
  `t_name` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`t_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of teacher
-- ----------------------------
INSERT INTO `teacher` VALUES ('1', '大王');
INSERT INTO `teacher` VALUES ('2', 'alex');
INSERT INTO `teacher` VALUES ('3', 'chao');
INSERT INTO `teacher` VALUES ('4', 'peiqi');

#7.2、统计各科各分数段人数.显示格式:课程ID,课程名称,[100-85],[85-70],[70-60],[ <60]

select  score.c_id,
        course.c_name,
     sum(CASE WHEN num BETWEEN 85 and 100 THEN 1 ELSE 0 END) as '[100-85]',
     sum(CASE WHEN num BETWEEN 70 and 85 THEN 1 ELSE 0 END) as '[85-70]',
     sum(CASE WHEN num BETWEEN 60 and 70 THEN 1 ELSE 0 END) as '[70-60]',
     sum(CASE WHEN num < 60 THEN 1 ELSE 0 END) as '[ <60]'
from score,course where score.c_id=course.c_id GROUP BY score.c_id;
```

需要掌握的函数：date_format ： 这个我们要讲一讲，将来你可能会用的到的，我们前面没有讲过的一个东西。

```
#1 基本使用
mysql> SELECT DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y');
       -> 'Sunday October 2009'
mysql> SELECT DATE_FORMAT('2007-10-04 22:23:00', '%H:%i:%s');
```

```
    -> '22:23:00'
mysql> SELECT DATE_FORMAT('1900-10-04 22:23:00',
    ->                 '%D %y %a %d %m %b %j');
    -> '4th 00 Thu 04 10 Oct 277'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
    ->                 '%H %k %I %r %T %S %w');
    -> '22 22 10 10:23:00 PM 22:23:00 00 6'
mysql> SELECT DATE_FORMAT('1999-01-01', '%X %V');
    -> '1998 52'
mysql> SELECT DATE_FORMAT('2006-06-00', '%d');
    -> '00'


#2 准备表和记录
CREATE TABLE blog (
    id INT PRIMARY KEY auto_increment,
    NAME CHAR (32),
    sub_time datetime
);

INSERT INTO blog (NAME, sub_time)
VALUES
    ('第1篇','2015-03-01 11:31:21'),
    ('第2篇','2015-03-11 16:31:21'),
    ('第3篇','2016-07-01 10:21:31'),
    ('第4篇','2016-07-22 09:23:21'),
    ('第5篇','2016-07-23 10:11:11'),
    ('第6篇','2016-07-25 11:21:31'),
    ('第7篇','2017-03-01 15:33:21'),
    ('第8篇','2017-03-01 17:32:21'),
    ('第9篇','2017-03-01 18:31:21');

#3. 提取sub_time字段的值，按照格式后的结果即"年月"来分组，统计一下每年每月的博客数量，怎么写呢，按照sub_time分组，但是
SELECT DATE_FORMAT(sub_time,'%Y-%m'),COUNT(1) FROM blog GROUP BY DATE_FORMAT(sub_time,'%Y-%m');

#结果
+------------------------------+----------+
| DATE_FORMAT(sub_time,'%Y-%m') | COUNT(1) |
+------------------------------+----------+
| 2015-03                      |        2 |
| 2016-07                      |        4 |
| 2017-03                      |        3 |
+------------------------------+----------+
3 rows in set (0.00 sec)
```

更多函数：中文猛击这里 OR 官方猛击这里

**一 自定义函数（自己简单看看吧）**

```
#！！！注意！！！
#函数中不要写sql语句（否则会报错），函数仅仅只是一个功能，是一个在sql中被应用的功能
#若要想在begin...end...中写sql，请用存储过程
```

```
delimiter //
create function f1(
```

```
    i1 int,
    i2 int)
returns int
BEGIN
    declare num int;
    set num = i1 + i2;
    return(num);
END //
delimiter ;
```

```
delimiter //
create function f5(
    i int
)
returns int
begin
    declare res int default 0;
    if i = 10 then
        set res=100;
    elseif i = 20 then
        set res=200;
    elseif i = 30 then
        set res=300;
    else
        set res=400;
    end if;
    return res;
end //
delimiter ;
```

**二 删除函数**

```
drop function func_name;
```

**三 执行函数**

```
# 获取返回值
select UPPER('chao') into @res;
SELECT @res;


# 在查询中使用
select f1(11,nid) ,name from tb2;
```

关于查看存储过程，函数，视图，触发器的语法：

```
查询数据库中的存储过程和函数
```

```
    select name from mysql.proc where db = 'xx' and type = 'PROCEDURE'   //查看xx库里面的存储过程
    select name from mysql.proc where db = 'xx' and type = 'FUNCTION'   //函数

    show procedure status; //存储过程
    show function status;     //函数

查看存储过程或函数的创建代码

    show create procedure proc_name;
    show create function func_name;

查看视图
    SELECT * from information_schema.VIEWS   //视图
    SELECT * from information_schema.TABLES   //表

查看触发器
    SHOW TRIGGERS [FROM db_name] [LIKE expr]
    SELECT * FROM triggers T WHERE trigger_name=”mytrigger” \G；其中triggers T就是triggers as T的意思，起别名
```

# 六 流程控制

### 一 条件语句

if条件语句：

```
delimiter //
CREATE PROCEDURE proc_if ()
BEGIN

    declare i int default 0;
    if i = 1 THEN
        SELECT 1;
    ELSEIF i = 2 THEN
        SELECT 2;
    ELSE
        SELECT 7;
    END IF;

END //
delimiter ;
```

### 二 循环语句

while循环：#后面讲索引的时候，咱们会用到while循环，注意语法

```
delimiter //
CREATE PROCEDURE proc_while ()
BEGIN

    DECLARE num INT ;
```

```
    SET num = 0 ;
    WHILE num < 10 DO
        SELECT
            num ;
        SET num = num + 1 ;
    END WHILE ;

END //
delimiter ;
```

到这里你会发现，其实sql也是一个开发语言，基本数据类型啊函数啊流程控制啊(if、while等)它都有。下面这两个我们简单看一下用法就行啦~~~

repeat循环：

```
delimiter //
CREATE PROCEDURE proc_repeat ()
BEGIN

    DECLARE i INT ;
    SET i = 0 ;
    repeat
        select i;
        set i = i + 1;
        until i >= 5
    end repeat;

END //
delimiter ;
```

loop：

```
BEGIN

    declare i int default 0;
    loop_label: loop

        set i=i+1;
        if i<8 then
            iterate loop_label;
        end if;
        if i>=10 then
            leave loop_label;
        end if;
        select i;
    end loop loop_label;

END
```

咱们本篇博客的重点是事务和存储过程，这是你将来和mysql打交道的时候会常用的内容。其他的内容是你了解知道的内容。那么MySQL的内容就还剩下索引及慢sql优化的问题了，我们下一篇博客再讲吧~~~