

MySQL (八)

1. 准备数据

```
create database p_t_v default character set utf8;
use p_t_v;

create table class
(
    id int unsigned not null auto_increment comment '编号',
    name varchar(20) not null comment '班级',
    pcount smallint unsigned not null default '0' comment '班级人数',
    primary key(id)
) comment='班级表';

insert into class
values
(1, '全栈开发班', 3),
(2, 'Java 开发班', 2),
(3, '服装设计班', 1),
(4, '美容美发班', 1),
(5, '挖掘机精英班', 0),
(6, '美国总统速成班', 0);

# 学生表
create table students
(
    id int unsigned not null auto_increment comment '编号',
    name varchar(20) not null comment '姓名',
    gender enum('男', '女') not null comment '性别',
    tel bigint unsigned not null comment '手机号',
    class_id int unsigned not null comment '班级 Id',
    age tinyint unsigned not null comment '年龄',
    height tinyint unsigned not null comment '身高, 单位: 厘米',
    primary key(id)
) comment='学生表';

insert into students
values
(1, '八戒', '男', 13912345555, 1, 20, 180),
(2, '大师兄', '男', 13912346666, 1, 21, 175),
```



```
(3, '金角大王', '男', 13922222222, 2, 204, 195),  
(4, '西施', '女', 13912342343, 4, 17, 165),  
(5, '大乔', '女', 13912346666, 3, 22, 168),  
(6, '貂蝉', '女', 13922233222, 1, 21, 164),  
(7, '大乔', '男', 13922225454, 2, 19, 183);
```

2. 存储过程

2.1 什么是存储过程

存储过程（Stored Procedure）和函数类似都是定义在数据库中用来实现一个功能的 SQL 语句集合，但函数实现的功能针对性比较强，一般只实现一个明确的小功能，比如：截取字符串、加密、向上取整、随机数等，而存储过程用来实现更加复杂的业务层面的功能，比如：注册、购买商品、下定单等。

2.2 函数和存储过程的比较

- 相同点

- 1、预先定义在 MySQL 中的为了实现一个特定功能的 SQL 语句集合。
- 2、需要调用才可以执行。
- 3、调用时可以传参数。

- 不同点

- 1) 存储过程实现的功能要复杂一点，而函数的实现的功能针对性比较强。
- 2) 存储过程的参数可以有 IN, OUT, INOUT 三种类型，而函数只能有 IN 类~~
- 3) 存储过程声明时不需要返回类型，而函数声明时需要描述返回类型，且函数体中必须包含一个有效的 RETURN 语句。
- 3) 存储过程需要单独执行，不能用在 sql 语句中，而函数需要使用 select 来执行，可以用在 SQL 语句中。

2.3 存储过程操作

2.3.1 创建存储过程

语法：

```
create procedure 过程名 (参数列表)
begin
    过程体
end
结束符
```

注意：如果过程体中只有一行指令，可以省略 begin 和 end。

示例：创建一个存储过程，用来输出 Hello World !

```
delimiter $$
create procedure prec1()
begin
    select 'Hello World !';
end
$$
delimiter ;
```

2.3.2 查看存储过程

查看所有存储过程

```
show procedure status [like 'xxx'];
```

查看某一个存储过程

```
show create procedure 过程名;
```

2.3.3 调用存储过程

需要使用 call 指令调用：

```
call 过程名 ();
```

注意：存储过程是没有返回值的。

2.3.4 删除存储过程

语法：

```
drop procedure 过程名
```

2.4 存储过程的参数

参数有三种类型 in（进入、输入）、out（出来、输出）、inout（即输入又输出）：

- in

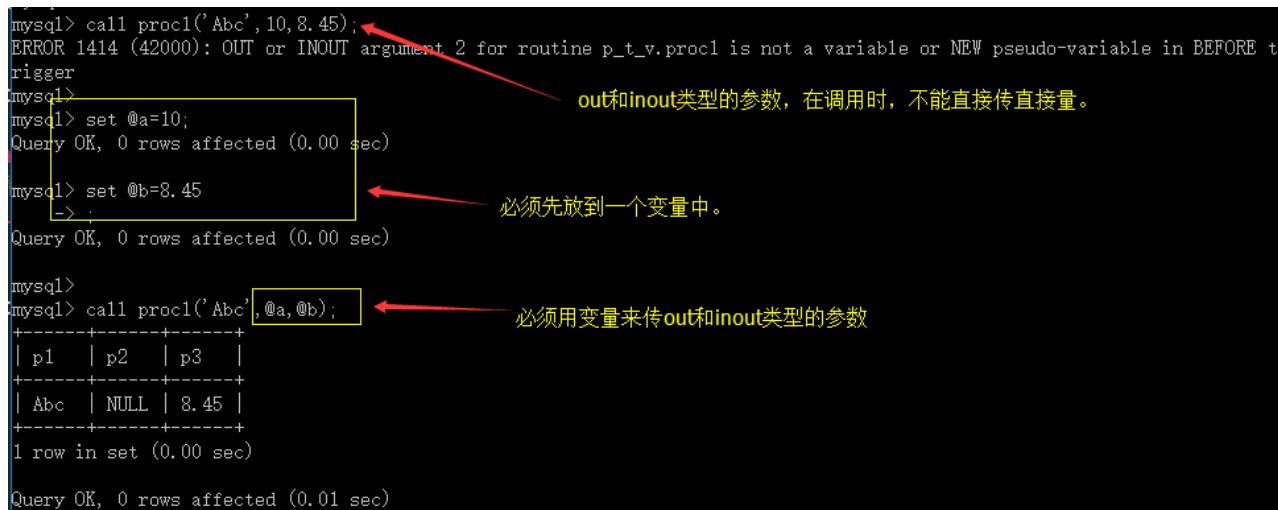
把数据从存储过程外面拿到存储过程里面。

- out

从存储过程里面把数据拿到外面来。

输出类型的参数，特点：

1. 调用时必须传一个变量，不能是一个直接量。



The screenshot shows a MySQL terminal session. The first command is `mysql> call proc1('Abc', 10, 8.45);`, which results in an error: `ERROR 1414 (42000): OUT or INOUT argument 2 for routine p_t_v.proc1 is not a variable or NEW pseudo-variable in BEFORE trigger`. A red arrow points from the text "out和inout类型的参数，在调用时，不能直接传直接量。" to the value `10` in the command. The second command is `mysql> set @a=10;`, followed by `mysql> set @b=8.45;`. A red arrow points from the text "必须先放到一个变量中。" to the variable `@b`. The third command is `mysql> call proc1('Abc', @a, @b);`, which is successful. A red arrow points from the text "必须用变量来传out和inout类型的参数" to the variable `@a`. The output shows a table with columns `p1`, `p2`, and `p3`, and one row with values `Abc`, `NULL`, and `8.45`.

```
mysql> call proc1('Abc', 10, 8.45);
ERROR 1414 (42000): OUT or INOUT argument 2 for routine p_t_v.proc1 is not a variable or NEW pseudo-variable in BEFORE trigger
mysql>
mysql> set @a=10;
Query OK, 0 rows affected (0.00 sec)

mysql> set @b=8.45;
Query OK, 0 rows affected (0.00 sec)

mysql> call proc1('Abc', @a, @b);
+-----+-----+-----+
| p1   | p2   | p3   |
+-----+-----+-----+
| Abc  | NULL | 8.45 |
+-----+-----+-----+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.01 sec)
```

2. out 不能用来接收传入的值，如果传了也会被设置为 null

```
mysql> set @a=10;
mysql> set @b=8.45;
mysql> call proc1('Abc', @a, @b);
+----+-----+-----+
| p1 | p2 | p3 |
+----+-----+-----+
| Abc | NULL | 8.45 |
+----+-----+-----+
1 row in set (0.00 sec)

mysql>
280 主体
281 end
282
283 注意：如果主体中只有一行SQL语句，那么begin和end可以省略。
284
285
286 delimiter $$
287 create procedure proc1(in p1 varchar(10), out p2 int, inout p3 decimal(5,2))
288 begin
289 # 输出三个参数
290 select p1,p2,p3;
291 end
292 $$
293 delimiter ;
```

out类型的参数，是用来向外传值的，所以这里@a的值无法传到存储过程里面来。
所有out类型传进来的值都会被设置为null

3. 当执行到 end 时过程会把过程内的值覆盖过程外对应的变量

```
mysql> set @a='abc';
mysql> set @b=10;
mysql> set @c=8.45;
mysql> call proc1(@a,@b,@c);
+----+-----+-----+
| p1 | p2 | p3 |
+----+-----+-----+
| abc | NULL | 8.45 |
+----+-----+-----+
1 row in set (0.00 sec)

mysql> select @a,@b,@c;
+----+-----+-----+
| @a | @b | @c |
+----+-----+-----+
| abc | 200 | 9.99 |
+----+-----+-----+
1 row in set (0.00 sec)
```

```
285
286
287 create procedure proc1(in p1 varchar(10), out p2 int, inout p3 decimal(5,2))
288 begin
289 # 输出三个参数
290 select p1,p2,p3;
291 # 修改这三个参数的值
292 set p1='bcd';
293 set p2=200;
294 set p3='9.99';
295 end
296 $$
297 delimiter ;
```

因为p1是in类型的参数，所以不会修改外面的@a的值
把@b的值修改了
把对应的@c的值修改了
out、和inout类型的参数，在存储过程中修改时，也会同时修改存储过程外面对应的变量。
b,c的值变了，因为b是out类型的，c是inout类型的。
a的值没变，因为a是in类型的，存储中无法修改。

● inout

既可以输入又可以输出类型的值。

特点：

1. 调用时必须传一个变量，不能是一个直接量。

```
mysql> call proc1('Abc', 10, 8.45);
ERROR 1414 (42000): OUT or INOUT argument 2 for routine p_t_v.proc1 is not a variable or NEW pseudo-variable in BEFORE trigger
mysql>
mysql> set @a=10;
mysql> set @b=8.45;
mysql> call proc1('Abc', @a, @b);
+----+-----+-----+
| p1 | p2 | p3 |
+----+-----+-----+
| Abc | NULL | 8.45 |
+----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

out和inout类型的参数，在调用时，不能直接传直接量。
必须先放到一个变量中。
必须用变量来传out和inout类型的参数

2. 当执行到 end 时过程会把过程内的值覆盖过程外对应的变量



```
mysql> set @a='abc';
Query OK, 0 rows affected (0.00 sec)

mysql> set @b=10;
Query OK, 0 rows affected (0.00 sec)

mysql> set @c=8.45;
Query OK, 0 rows affected (0.00 sec)

mysql> call proc1(@a,@b,@c);
+----+-----+-----+
| p1 | p2 | p3 |
+----+-----+-----+
| abc | NULL | 8.45 |
+----+-----+-----+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> select @a,@b,@c;
+----+-----+-----+
| a | b | c |
+----+-----+-----+
| abc | 200 | 9.99 |
+----+-----+-----+
1 row in set (0.00 sec)
```

```
285
286 delimiter $$
287 create procedure proc1(in p1 varchar(10), out p2 int, inout p3 decimal(5,2))
288 begin
289     # 输出三个参数
290     select p1,p2,p3;
291     # 修改这三个参数的值
292     set p1='bcd';
293     set p2=200;
294     set p3='9.99';
295 end
296 $$
297 delimiter ;
```

因为p1是in类型的参数，所以不会修改外面的@a的值

把@b的值修改了

把对应的@c的值修改了

out、和inout类型的参数，在存储过程中修改时，也会同时修改存储过程外面对应的变量。

b,c的值变了，因为b是out类型的，c是inout类型的。

a的值没变，因为a是in类型的，存储中无法修改。

总结：

in：只能向存储过程中传值，存储过程中无法修改外面对应的变量。【向里传值】

out：无法向存储过程中传值，但是在存储过程里面可以修改外面对应的变量。【向外写值】

inout：即可以向过程中传值，又可以向存储过程外面写值。

2.5 案例、定义一个存储过程实现学生转班

3. 触发器





触发器（trigger），是一种特殊类型的存储过程，不同于之前的存储过程需要使用 call 来调用，触发器是在特定时机自动被调用执行的。【类似于 JS 里面的事件】

3.1 创建触发器

语法：

```
create trigger 触发器名称 触发时机 触发事件 on 表 for each row
begin
...
end
```

- 触发时机

before: 在事件之前触发。

after: 在事件之后触发。

- 触发事件

insert: 插入事件。

update: 修改事件。

delete: 删除事件。

- 触发器中的关键字

表中的每条记录在进行操作时都会触发一个触发器，而表中的每条记录都有操作前和操作后两个状态，分别保存在 new 和 old 两个关键字中。

示例：当向一个班级添加一个学生时，把这个班级的人数+1，当一个班级里面减少一个学生时，这个班级的人数-1。

```
delimiter $$
create trigger update_stu_count after insert on students for each row
begin
    #这段SQL在向students表中插入数据之后执行。
    #把这个学生所在的班级的人数+1
    # 在触发器中可以使用new和old两个关键字。
    # new: 新插入的数据的信息
    update class set pcount=pcount+1 where id=new.class_id;
end
$$
delimiter ;
```



```
mysql> insert into students values(8,'白马龙','男','19943438989',6,18,170);  
Query OK, 1 row affected (0.00 sec)
```

```
mysql> select * from students;
```

id	name	gender	tel	class_id	age	height
1	八戒	男	13912345555	1	20	180
2	大师兄	男	13912346666	1	21	175
3	金角大王	男	13922222222	2	204	195
4	西施	女	13912342343	4	17	165
5	大乔	女	13912346666	3	22	168
6	貂蝉	女	13922233222	1	21	164
7	大乔	男	13922225454	2	19	183
8	白马龙	男	19943438989	6	18	170

```
8 rows in set (0.00 sec)
```

```
mysql> select * from class;
```

id	name	pcount
1	全栈开发班	3
2	Java开发班	2
3	服装设计班	1
4	美容美发班	1
5	挖掘机精英班	0
6	美国总统速成班	1

```
6 rows in set (0.00 sec)
```

当我们向学生表中插入一条记录时，这时会触发一个触发器中的代码，这个触发器根据这条记录学生所在的班级ID，修改了这个班级的学生人数。

注意：一张表的同一类触发器只能有一个，所以一张表最多可以定义六个触发器：before insert【插入前】、after insert【插入后】、before update【修改前】、after update【修改后】、before delete【删除前】、after delete【删除后】。

3.2 查看触发器

查看所有触发器：

```
show triggers
```

查看一个触发器详情：

```
show create trigger 触发器名
```




3.3 删除触发器

语法:

```
drop trigger 触发器名;
```

练习、当删除一个学生时，让这个学生所在的班级人数自动-1。

4. 视图

视图（view），是由 select 语句组成的一张虚拟表，可以当做表来使用。

4.1 什么是视图

视图是一个虚拟表，其内容由查询定义。同真实的表一样，视图包含一系列带有名称的列和行数据。但是，视图并不在数据库中以存储的数据值集形式存在。行和列数据来自自定义视图的查询所引用的表，并且在引用视图时动态生成。

视图的好处:

1. 简单性，可以把复杂的 SQL 简单化。



```
# 取出所有的班级的名称以及每个班级中年龄最大的学生信息。
#第一步：先根据年龄排序
select * from students order by age desc;
```

id	name	gender	tel	class_id	age	height
3	金角大王	男	13922222222	2	204	195
5	大乔	女	13912346666	3	22	168
2	大师兄	男	13912346666	1	21	175
6	貂蝉	女	13922233222	1	21	164
1	八戒	男	13912345555	1	20	180
7	大乔	男	13922225454	2	19	183
8	白马龙	男	19943438989	6	18	170
4	西施	女	13912342343	4	17	165

```
8 rows in set (0.00 sec)
#第二步:根据班级分组
select * from (select * from students order by age desc) a group by class_id;
```

id	name	gender	tel	class_id	age	height
1	八戒	男	13912345555	1	20	180
3	金角大王	男	13922222222	2	204	195
5	大乔	女	13912346666	3	22	168
4	西施	女	13912342343	4	17	165
8	白马龙	男	19943438989	6	18	170

```
#第三步: 根据class_id外连班级表, 取出班级名称
select b.name class_name,a.* from (select * from students order by age desc) a left join class b on a.class_id=b.id group by a.class_id;
```

class_name	id	name	gender	tel	class_id	age	height
全栈开发班	1	八戒	男	13912345555	1	20	180
Java开发班	3	金角大王	男	13922222222	2	204	195
服装设计班	5	大乔	女	13912346666	3	22	168
美容美发班	4	西施	女	13912342343	4	17	165
美国总统速成班	8	白马龙	男	19943438989	6	18	170

以上这个功能比较复杂, 如果再添加更多功能这个 SQL 会越来越复杂, 以至不便于维护, 所以我们可以把这个复杂的 SQL 定义的一个视图, 以后再用时直接使用视图即可。

2. 安全性, 通过视图我们可以限制用户在一个数据的子集上。

4.2 视图的基本操作

4.2.1 创建视图

语法:

```
create view 视图名称 as select 语句
```

```
mysql> create view class_max_age as select b.name class_name,a.* from (select * from students order by age desc) a left
join class b on a.class_id=b.id group by a.class_id;
Query OK, 0 rows affected (0.01 sec)
```

说明: 就是使用一个 select 语句的结果制作一张虚拟表。



4.2.2 查看视图

视图就是一张虚拟表，所以对视图的操作基本和表操作一样。

语法：

```
desc|show create table 视图名;    # 查看某个视图  
show tables;                      # 查看所有视图
```

4.2.3 修改视图

语法：

```
alter view 视图名 as 新的 sql 语句
```

4.2.4 删除视图

语法：

```
drop view 视图名称
```

4.3 视图数据操作

就是使用一个 select 语句的结果制作一张虚拟表，所以操作视图中的数据和操作表的语法基本一样。

4.3.1 视图数据操作原理

视图是虚拟表，本身并没有数据，数据的操作实际上是通过视图找到对应的基表【select 语句中的表】进行操作的。

4.3.2 查看数据

```
mysql> select * from class_max_age;
+-----+-----+-----+-----+-----+-----+-----+-----+
| class_name | id | name | gender | tel | class_id | age | height |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 全栈开发班 | 1 | 八戒 | 男 | 13912345555 | 1 | 20 | 180 |
| Java开发班 | 3 | 金角大王 | 男 | 13922222222 | 2 | 204 | 195 |
| 服装设计班 | 5 | 大乔 | 女 | 13912346666 | 3 | 22 | 168 |
| 美容美发班 | 4 | 西施 | 女 | 13912342343 | 4 | 17 | 165 |
| 美国总统速成班 | 8 | 白马龙 | 男 | 19943438989 | 6 | 18 | 170 |
+-----+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> select * from class_max_age where height between 170 and 180;
+-----+-----+-----+-----+-----+-----+-----+-----+
| class_name | id | name | gender | tel | class_id | age | height |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 全栈开发班 | 1 | 八戒 | 男 | 13912345555 | 1 | 20 | 180 |
| 美国总统速成班 | 8 | 白马龙 | 男 | 19943438989 | 6 | 18 | 170 |
+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> _
```

插入、修改、删除视图中的数据时有一个要求：视图必须是一个单表构成的视图。所以我们刚刚创建的视图是不能执行插入、修改和删除操作的，因为视图是由学生表和班级表两个表构成的。

4.3.3 插入数据

向视图中插入数据实际上会插入到 `select` 的基表中，不过要注意，如果 `select` 是从多表中查询出的数据，那么这个视图不允许插入数据。

4.3.4 修改数据

和操作表一样。

4.3.5 删除数据

和操作表一样，但是如果视图是由多张表中取的数据，那么不允许删除数据。

4.3.6 with check option

视图特有的一个属性，可以在创建视图时添加 **with check option** 属性，加了这个之后，对视图所有的数据操作时都会先使用这个条件验证一下，只有满足条件时才允许操作。

其实就是向这个表又额外加了一个限制。

语法：

```
create view 视图名称 as select 语句 [where 条件 with check option]
```

5. sql_mode

6. 今日总结

