# MySQL中的特性-扩展

讲师: 伊川

# MySQL存储过程

## 什么是存储过程?

> 目前使用的大多数SQL语句都是针对一个或多个表的单条语句。并非所有操作都这么简单，经常会有一个完整的操作需要多条语句 才能完成。

例如以下的情形。

- 为了处理订单，需要核对以保证库存中有相应的物品。
- 如果库存有物品，需要预定以便不将它们再卖给别的人， 并减少可用的物品数量以反映正确的库存量。
- 库存中没有的物品需要订购，这需要与供应商进行某种交互。

> 执行这个处理需要针对许多表的多条MySQL语句。可能需要执行的具体语句及其次序也不是固定的。
>
> 那么，怎样编写此代码?可以单独编写每条语句，并根据结果有条件地执行另外的语句。
>
> 在每次需要这个处理时(以及每个需要它的应用中)都必须做这些工作。

可以创建存储过程。

- 存储过程简单来说，就是为以后的使用而保存 的一条或多条MySQL语句的集合。
- 储存过程是一组为了完成特定功能的SQL语句集，经过编译之后存储在数据库中，在需要时直接调用。
- 存储过程就像脚本语言中函数定义一样。

## 为什么要使用存储过程?

优点:

- 可以把一些复杂的sql进行封装,简化复杂操作
- 保证了数据的完整性,防止错误
- 简单的变动只需要更改存储过程的代码即可
- 提高性能。因为使用存储过程比使用单独的SQL语句要快。(预先编译)

缺点:

- 存储过程的编写比SQL语句复杂
- 一般可能还没有创建存储过程的权限,只能调用

## 个人观点:

- 业务逻辑不要封装在数据库里面,应该由应用程序(JAVA、Python、PHP)处理。
- 让数据库只做它擅长和必须做的，减少数据库资源和性能的消耗。
- 维护困难，大量业务逻辑封装在存储过程中，造成业务逻辑很难剥离出来。动A影响B。
- 人员也难招聘，因为既懂存储过程，又懂业务的人少。使用困难。

在电信、银行业、金融方面以及国企都普遍使用存储过程来熟悉业务逻辑，但在互联网中相对较少。

### 创建存储过程

> `\d //` 修改MySQL默认的语句结尾符`;`，改为`//`。
>
> `create procedure` 创建语句
>
> BEGIN和END语句用来限定存储过程体

```
-- 定义存储过程
\d //
create procedure p1()
begin
set @i=10;
while @i<90 do
insert into users values(null,concat('user:',@i),@i,0);
set @i=@i+1;
end while;
end;
//
```

### 执行储存

`call p1()`

### 查看存储过程

`show create procedure p1\G`

### 删除存储过程

`drop procedure p1`

# MySQL的触发器

MySQL语句在需要时被执行，存储过程也是如此。

但是，如果你想要某条语句(或某些语句)在事件发生时自动执行，怎么办呢?

例如:

- 每当增加一个顾客到某个数据库表时，都检查其电话号码格式是否正确;
- 每当订购一个产品时，都从库存数量中减去订购的数量;
- 无论何时删除一行，都在某个存档表中保留一个副本。

所有这些例子的共同之处是它们都需要在某个表发生更改时自动处理。这确切地说就是触发器。

## 触发器的定义

> 触发器是MySQL响应写操作(增、删、改)而自动执行的一条或一组定义在BEGIN和END之间的MySQL语句
>
> 或可理解为：提前定义好一个或一组操作,在指定的SQL操作前或后来触发指定的SQL自动执行
>
> 触发器就像是JavaScript中的事件一样

举例: 定义一个update语句,在向某个表中执行insert添加语句时来触发执行,就可以使用触发器

## 触发器语法：

```
 CREATE TRIGGER trigger_name trigger_time trigger_event
   ON tbl_name FOR EACH ROW trigger_stmt
说明：
# trigger_name：触发器名称
# trigger_time:触发时间，可取值：BEFORE或AFTER
# trigger_event：触发事件，可取值：INSERT、UPDATE或DELETE。
# tbl_name：指定在哪个表上
# trigger_stmt：触发处理SQL语句。

-- 查看所有的 触发器
show triggers\G;

-- 删除触发器
drop trigger trigger_name;
```

## 触发器Demo

注意:如果触发器中SQL有语法错误,那么整个操作都会报错

```
-- 创建一个删除的触发器,在users表中删除数据之前,往del_users表中添加一个数据

-- 1,复制当前的一个表结构
create table del_users like users;

-- 2,创建 删除触发器 注意在创建删除触发器时,只能在删除之前才能获取到old(之前的)数据
\d //
create trigger deluser before delete on users for each row
begin
```

```
insert into del_users values(old.id,old.name,old.age,old.account);
end;
//
\d ;

-- 3 删除users表中的数据去实验
```

tips：

- 在INSERT触发器代码内，可引用一个名为NEW的虚拟表，访问被 插入的行;
- 在DELETE触发器代码内，可以引用一个名为OLD的虚拟表，访问被删除的行;
  - OLD中的值全都是只读的，不能更新。
  - 在AFTER DELETE的触发器中无法获取OLD虚拟表
- 在UPDATE触发器代码中
  - 可以引用一个名为OLD的虚拟表访问更新以前的值
  - 可以引用一个名为NEW的虚拟表访问新 更新的值;

### 练习题

用触发器来实现数据的统计

```
-- 1.创建一个表, users_count 里面有一个 num的字段 初始值为0或者是你当前users表中的count
-- 2,给users表创建一个触发器
-- 当给users表中执行insert添加数据之后,就让users_count里面num+1,
-- 当users表中的数据删除时,就让users_count里面num-1,
-- 想要统计users表中的数据总数时,直接查看 users_count
```

# MySQL中的视图

### 什么是视图?

视图是虚拟的表。与包含数据的表不一样，视图只包含使用时动态检索数据的查询。

视图仅仅是用来查看存储在别处的数据的一种设施或方法。

视图本身不包含数据，因此它们返回的数据是从其他表中检索出来的。

在添加或更改这些表中的数据时，视图将返回改变过的数据。

因为视图不包含数据，所以每次使用视图时，都必须处理查询执行时所需的任一个检索。

如果你用多个联结和过滤创建了复杂的视图或者嵌套了视图，可能会发现性能下降得很厉害。

### 视图的作用

1. 重用SQL语句。
2. 简化复杂的SQL操作。在编写查询后，可以方便地重用它而不必知道它的基本查询细节。
3. 使用表的组成部分而不是整个表。
4. 保护数据。可以给用户授予表的特定部分的访问权限而不是整个表的访问权限。
5. 更改数据格式和表示。视图可返回与底层表的表示和格式不同的数据。
6. 注意:视图不能索引，也不能有关联的触发器或默认值。

## 视图的基础语法

```
创建视图：
create view v_users as select id,name,age from users where age >= 25 and age
<= 35;
-- Query OK, 0 rows affected (0.00 sec)

view视图的帮助信息：
mysql> ? view
ALTER VIEW
CREATE VIEW
DROP VIEW

查看当前库中所有的视图
show tables; --可以查看到所有的表和视图
show table status where comment='view'; --只查看当前库中的所有视图

删除视图v_t1：
mysql> drop view v_t1;
```