**本节所讲内容：**

* **存储过程**
* **触发器**
* **事务**

# **一、存储过程**

## 什么是存储过程

**大多数SQL语句都是针对一个或多个表的单条语句。并非所有的操作都怎么简单。经常会有一个完整的操作需要多条才能完成。存储过程（Stored Procedure）是在大型数据库系统中，一组为了完成特定功能的SQL 语句集，存储在数据库中经过第一次编译后再次调用不需要再次编译，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。存储过程是数据库中的一个重要对象，任何一个设计良好的数据库应用程序都应该用到存储过程。**

## 为什么要使用存储过程 （就是优点）

**(1).增强SQL语言的功能和灵活性：存储过程可以用控制语句编写，有很强的灵活性，可以完成复杂的判断和较复杂的运算。**

**(2).标准组件式编程：存储过程被创建后，可以在程序中被多次调用，而不必重新编写该存储过程的SQL语句。而且数据库专业人员可以随时对存储过程进行修改，对应用程序源代码毫无影响。**

**(3).较快的执行速度：如果某一操作包含大量的Transaction-SQL代码或分别被多次执行，那么存储过程要比批处理的执行速度快很多。因为存储过程是预编译的。在首次运行一个存储过程时查询，优化器对其进行分析优化，并且给出最终被存储在系统表中的执行计划。而批处理的Transaction-SQL语句在每次运行时都要进行编译和优化，速度相对要慢一些。**

**(4).减少网络流量：针对同一个数据库对象的操作（如查询、修改），如果这一操作所涉及的Transaction-SQL语句被组织进存储过程，那么当在客户计算机上调用该存储过程时，网络中传送的只是该调用语句，从而大大减少网络流量并降低了网络负载。**

**(5).作为一种安全机制来充分利用：通过对执行某一存储过程的权限进行限制，能够实现对相应的数据的访问权限的限制，避免了非授权用户对数据的访问，保证了数据的安全。**

## 为什么不使用存储过程：

1. **可移植性差**
2. **对于简单的SQL语句，存储过程没什么优势**
3. **如果存储过程中不一定会减少网络传输**
4. **如果只有一个用户使用数据库，那么存储过程对安全也没什么影响**
5. **团队开发时需要先统一标准，否则后期维护成本大**
6. **在大并发量访问的情况下，不宜写过多涉及运算的存储过程**
7. **业务逻辑复杂时，特别是涉及到对很大的表进行操作的时候，不如在前端先简化业务逻辑**

## 定义存储过程

**语法：**

**create procedure 过程名（参数1，参数2....）**

**begin**

**sql语句;**

**end**

**创建存储过程之前我们必须修改mysql语句默认结束符 ; 要不然我们不能创建成功**

**使用delimiter可以修改执行符号**

**DELIMITER是分割符的意思，因为MySQL默认以";"为分隔符，如果我们没有声明分割符，那么编译器会把存储过程当成SQL语句进行处理，则存储过程的编译过程会报错，所以要事先用DELIMITER关键字申明当前段分隔符，这样MySQL才会将";"当做存储过程中的代码，不会执行这些代码，用完了之后要把分隔符还原。**

**语法：**

**delimiter 新执行符号 可以是%或//**

**mysql>use book;**

**mysql> delimiter % 这样结束符就为%**

**mysql> create procedure selCg()**

**-> begin**

**-> select \* from category;**

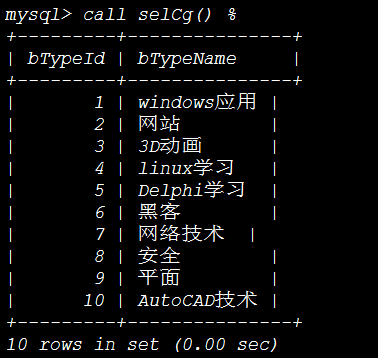
**-> end %**

## 调用存储过程

**语法：**

**call 过程名（参数1，参数2）;**

**mysql> call selCg() %**



## 存储过程参数类型

### In参数 传入参数

**特点：读取外部变量值，且有效范围仅限存储过程内部**

**例子：**

**mysql>use book;**

**mysql> delimiter //**

**mysql> create procedure xx(in abc int)**

**-> begin**

**-> select abc;**

**-> set abc=2;**

**-> select abc;**

**-> end;**

**-> //**

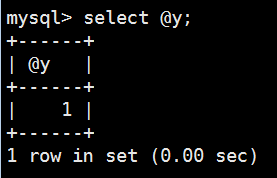
**mysql> delimiter ; 使用完马上恢复默认的**

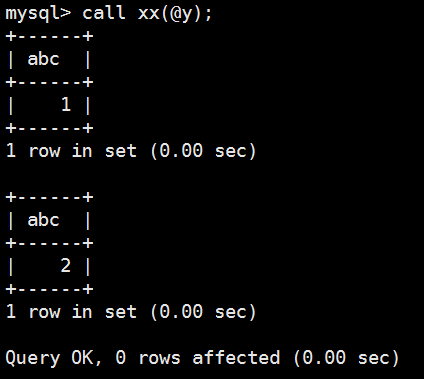
**mysql定义变量方法**

**语法格式：** **set @字段名=值**

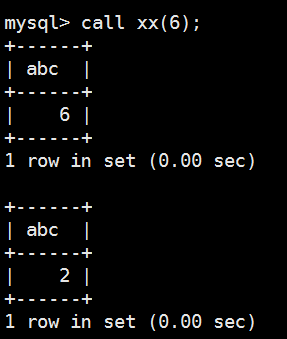
**例如 set @num=1**

**mysql> set @y=1; 局部变量**

**@**



**直接写数值传入值：**



**例：定义一个存储过程 getOneBook，当输入某书籍 id 后，可以调出对应书籍记录**

mysql>delimiter //

mysql> create procedure getOneBook(in b int) # b是参数名

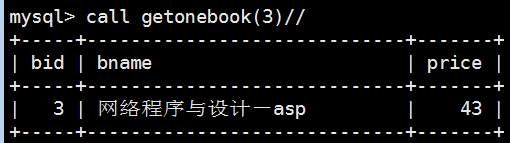
-> begin

-> select bid,bname,price from books where bId=b;

-> end //

Query OK, 0 rows affected (0.01 sec)

mysql> call getOneBook(3)//



### Out参数 传出参数

**特点：不读取外部变量值，在存储过程执行完毕后保留新值**

**mysql> delimiter //**

**mysql> create procedure pout(out p\_out int)**

**-> begin**

**-> select p\_out;**

**-> set p\_out=2;**

**-> select p\_out;**

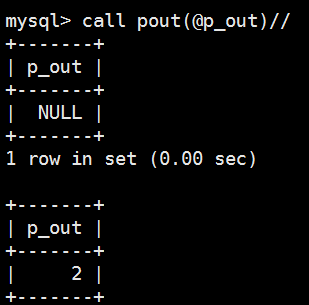
**-> end;**

**-> //**

**mysql> delimiter ;**

**mysql> set @p\_out=1;**

**mysql> call pout(@p\_out);**



**In传入参数,是外部将值传给存储过程来使用的,而out传出参数是为了让存储过程的执行结果回传给调用他的程序来使用的.**

**再举个例子**

**mysql> create procedure demo(out pb varchar(200))**

**>begin**

**>select bname into @pa from books where bid=3;**

**>select @pa as '图书名';**

**>end //**

**调用，执行：**

**mysql> call demo(@a); //**

**查看变量@pa 中的值：**

**mysql> select @pa;//**

**+-----------------------------+**

**|图书名 |**

**+-----------------------------+**

**| 网络程序与设计－asp |**

**+-----------------------------+**

### Inout参数

**特点：读取外部变量，在存储过程执行完后保留新值<类似银行存款>**

**传进来，又传出去。**

**mysql> delimiter //**

**mysql> create procedure pinout(inout p\_inout int)**

**-> begin**

**-> select p\_inout;**

**-> set p\_inout=2;**

**-> select p\_inout;**

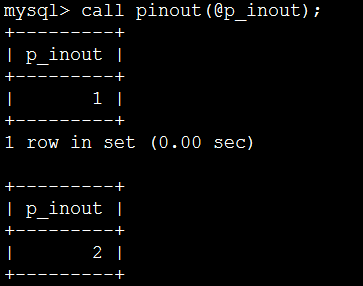
**-> end;**

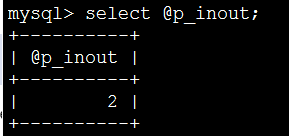
**-> //**

**mysql> delimiter ;**

**mysql> set @p\_inout=1;**

**mysql> call pinout(@p\_inout);**





### 不加参数的情况

**如果在创建存储过程时没有指定参数类型，则需要在调用的时候指定参数值**

**mysql> create table t2(id int(11)); 创建表**

**mysql> create procedure t2(n1 int)**

**-> begin**

**-> set @x=0;**

**-> repeat set @x=@x+1; 作了循环**

**-> insert into t2 values(@x);**

**-> until @x>n1**

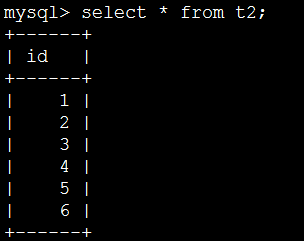
**-> end repeat;**

**-> end;**

**-> //**

**mysql> delimiter ;**

**mysql> call t2(5); 循环5次**



## 存储过程变量的使用

**MySQL中使用declare进行变量定义**

**变量定义：DECLARE variable\_name [,variable\_name...] datatype [DEFAULT value];**

**datatype为MySQL的数据类型，如:int, float, date, varchar(length)**

**变量赋值： SET 变量名 = 表达式值 [,variable\_name = expression ...]**

**变量赋值可以在不同的存储过程中继承**

**mysql> create procedure decl()**

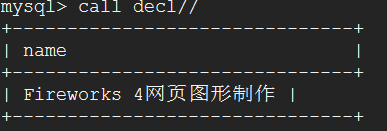
**-> begin**

**-> declare name varchar(200);**

**-> set name=(select bName from books where bId=12);**

**-> select name;**

**-> end//**



## 存储过程语句的注释

**做过开发的都知道，写注释是个利人利己的事情。便于理解维护**

**MySQL注释有两种风格**

**“--“：单行注释**

**“/\*…..\*/”:一般用于多行注释**

**例子：**



## 存储过程流程控制语句

### 变量作用域：

**内部的变量在其作用域范围内享有更高的优先权，当执行到end。变量时，内部变量消失，此时已经在其作用域外，变量不再可见了，因为在存储过程外再也不能找到这个申明的变量，但是你可以通过out参数或者将其值指派给会话变量来保存其值。**

**mysql > DELIMITER //**

**两层嵌套**

**mysql > CREATE PROCEDURE proc3()**

**-> begin**

**-> declare x1 varchar(5) default 'outer';**

**-> begin**

**-> declare x1 varchar(5) default 'inner';**

**-> select x1;**

**-> end;**

**-> select x1;**

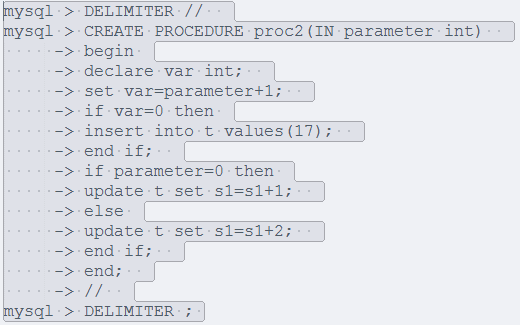
**-> end;**

**-> //**

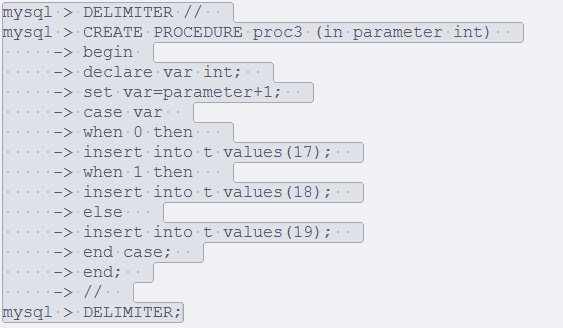
**mysql > DELIMITER ;**

### 条件语句

**1：if-then -else语句**

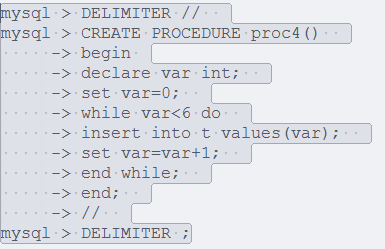
****

**2：case语句：**



### 循环语句：

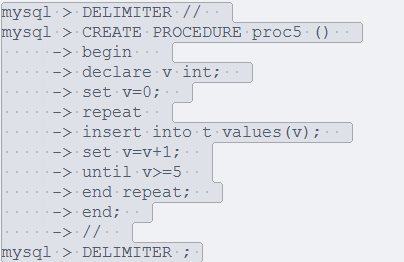
**1：while ···· end while：**



**while 1 do ....... if \*\*\*\* then break; end while;**

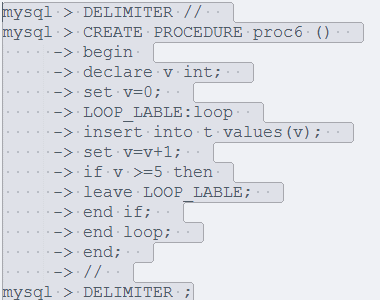
**2：repeat···· end repeat：**

**执行操作后检查结果，而while则是执行前进行检查。**



**3：loop ·····end loop:**

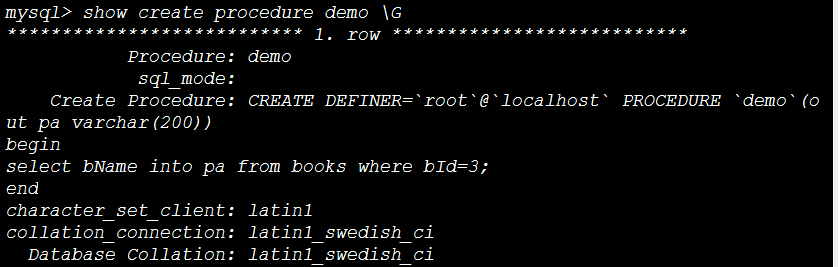
**loop循环不需要初始条件，这点和while 循环相似，同时和repeat循环一样不需要结束条件, leave语句的意义是离开循环。**



## 查看存储过程

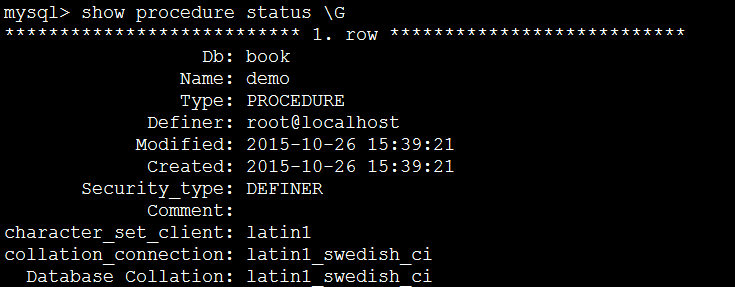
**查看存储过程内容：**

**mysql> show create procedure demo \G**



**查看存储过程状态：**

**mysql> show procedure status \G 查看所有存储过程**



## 修改存储过程：

**使用alter语句修改(一般不建议这样改)**

**ALTER {PROCEDURE | FUNCTION} sp\_name [characteristic ...]   
characteristic:   
{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }   
| SQL SECURITY { DEFINER | INVOKER }   
| COMMENT 'string'**

**sp\_name参数表示存储过程或函数的名称**

**characteristic参数指定存储函数的特性**

**CONTAINS SQL表示子程序包含SQL语句，但不包含读或写数据的语句；**

**NO SQL表示子程序中不包含SQL语句**

**READS SQL DATA表示子程序中包含读数据的语句**

**MODIFIES SQL DATA表示子程序中包含写数据的语句**

**SQL SECURITY { DEFINER | INVOKER }指明谁有权限来执行**

**DEFINER表示只有定义者自己才能够执行**

**INVOKER表示调用者可以执行**

**COMMENT 'string'是注释信息。**

**--**

**/\*\*/**

**就是说只能改名字和定义，不能改里面的内容。要删了重新建**

**通过第三方工具 修改**

## 删除存储过程

**语法：**

**方法一：DROP  PROCEDURE  过程名**

**mysql> drop procedure p\_inout;**

**方法二：DROP PROCEDURE  IF  EXISTS存储过程名**

**这个语句被用来移除一个存储程序。不能在一个存储过程中删除另一个存储过程，只能调用另一个存储过程**

# **二：触发器**

## 什么是触发器： 我们之前级联更新删除 生产比较少用触发器

**触发器是一种特殊的存储过程，它在插入，删除或修改特定表中的数据时触发执行，它比数据库本身标准的功能有更精细和更复杂的数据控制能力**

## 触发器的作用：

1. **安全性**

**可以基于数据库的值使用户具有操作数据库的某种权利。**

**可以基于时间限制用户的操作，例如不允许下班后和节假日修改数据库数据**

**可以基于数据库中的数据限制用户的操作，例如不允许股票的价格的升幅一次超过10%**

1. **审计**

**可以跟踪用户对数据库的操作**

**审计用户操作数据库的语句**

**把用户对数据库的操作写入审计表**

1. **实现复杂的数据完整性规则**

**实现非标准的数据完整性检查和约束。触发器可产生比规则更为复杂的限制。与规则不同，触发器可以引用列或数据库对象。**

**例如，触发器可回退任何企图吃进超过自己保证金的期货。**

**4．实现复杂的非标准的数据库相关完整性规则。**

**触发器可以对数据库中相关的表进行连环更新。**

**例如，在auths表author\_code列上的删除触发器可导致相应删除在其它表中的与之匹配的行。**

**触发器能够拒绝或回退那些破坏相关完整性的变化，取消试图进行数据更新的事务**

**5．实时同步地复制表中的数据**

**6．自动计算数据值**

**如果数据的值达到了一定的要求，则进行特定的处理。**

**例如，如果公司的帐号上的资金低于5万元则立即给财务人员发送警告数据**

## 创建触发器：

**语法：**

**create trigger 触发器名称 触发的时机 触发的动作 on 表名 for each row 触发器状态。**

**参数说明：**

**触发器名称: 自己定义**

**触发的时机： before /after 在执行动作之前还是之后**

**触发的动作 :指的激发触发程序的语句类型<insert ,update,delete>**

**each row:操作第一行我都监控着**

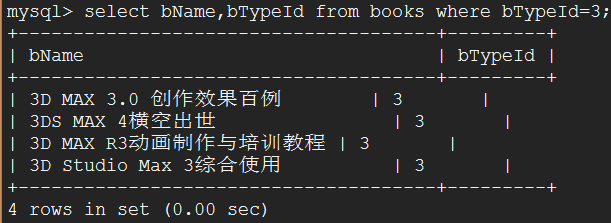
**触发器创建语法四要素：1.监视地点(table) 2.监视事件(insert/update/delete) 3.触发时间(after/before) 4.触发事件(insert/update/delete)**

**例：当category表中，删除一个bTypeid=3的图书分类时，books表中也要删除对应分类的图书信息（类似级联删除）**

**mysql> use book;**

**在category执行删除前，查看bTypeId=3的图书分类：**

**mysql> select bName,bTypeId from books where bTypeId=3;**



**创建触发 （实际跟创建存储过程一样，关键字换而已）**

**mysql> delimiter //**

**mysql> create trigger delCategory after delete on category for each row**

**-> delete from books where bTypeId=3;**

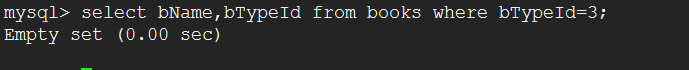
**-> //**

**删除bTypeId=3的记录**

**mysql> delete from category where bTypeId=3;**

**查看：是否还有bTypeId=3的图书记录。可以看出已经删除。**

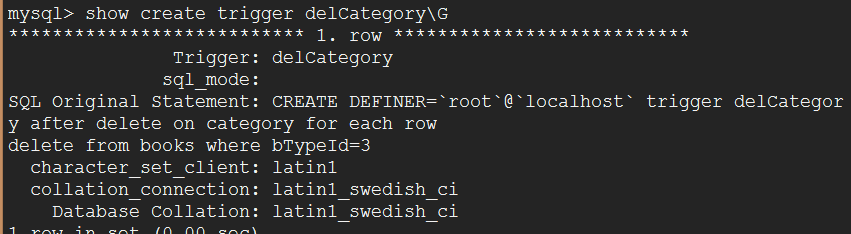
Select bname,btypeid from books where btypeid=3//

****

## 查看触发器：

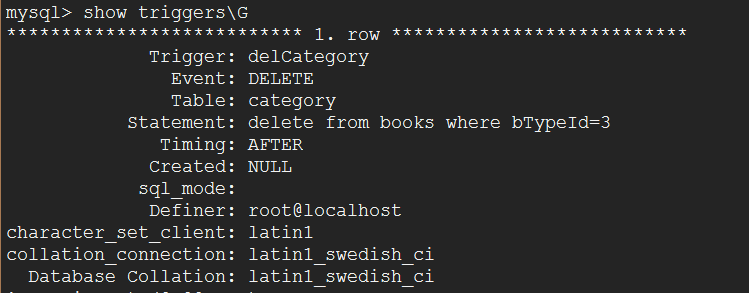
**1：查看创建过程**

**mysql> show create trigger delCategory\G**



**2：查看触发器详细信息**

**mysql> show triggers\G 这个查看所有的**

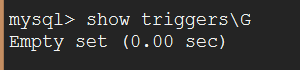


## 删除触发器:

**语法：**

**drop trigger 触发器名称;**

**mysql> drop trigger delCategory;**



# **三：事务**

## 什么是事务：

**数据库事务：（database transaction）: 事务是由一组SQL语句组成的逻辑处理单元，要不全成功要不全失败。**

**※ MYSQL中只有INNODB和BDB类型的数据表才能支持事务处理！其他的类型不支持！**

**事务处理：可以确保非事务性单元的多个操作都能成功完成，否则不会更新数据资源。**

**数据库默认事务是自动提交的， 也就是发一条 sql 它就执行一条。如果想多条 sql 放在一个事务中执行，则需要使用事务进行处理。当我们开启一个事务，并且没有提交，mysql 会自动回滚事务。或者我们使用 rollback 命令手动回滚事务。**

**优点：通过将一组操作组成一个，执行时，要么全部成功，要么全部失败的单元。**

**使程序更可靠，简化错误恢复。**

**例：**

* **A汇款给B 1000元**
  + **A账户-1000**
  + **B账户+1000**
* **以上操作对应数据库为两个update。这两个操作属于一个事物。否则，可能会出现A账户钱少了，B账户钱没增加的情况。**

## 事务四大特性：

**事务是必须满足4个条件（ACID）**

1. **原子性（Autmic）：事务在执行性，要做到“要么不做，要么全做！”，就是说不允许事务部分得执行。即使因为故障而使事务不能完成，在rollback时也要消除对数据库得影响！**
2. **一致性（Consistency）：事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的。在事务开始之前和结束之后，数据库的完整性约束没有被破坏**
3. **隔离性（Isolation）：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的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰，这些通过锁来实现。**
4. **持久性（Durability）：指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障（比如说宕机等）不应该对其有任何影响。**

**事务的ACID特性可以确保银行不会弄丢你的钱，而在应用逻辑中，要实现这点非常难，甚至可以说是不可能完成的任务。**

## MySQL事务处理的方法：

1. **用BEGIN,ROLLBACK,COMMIT来实现**

**START TRANSACTION | BEGIN [WORK] 开启事务**

**COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE] 提交当前事务，执行永久操作。**

**ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE] 回滚当前事务到开始点，取消上一次开始点后的所有操作。**

**SAVEPOINT 名称 折返点**

1. **直接用set来改变mysql的自动提交模式**

**MYSQL默认是自动提交的，也就是你提交一个QUERY，它就直接执行！**

**SET AUTOCOMMIT = {0 | 1} 设置事务是否自动提交，默认是自动提交的。**

**0：禁止自动提交**

**1：开启自动提交。**

**※ MYSQL中只有INNODB和BDB类型的数据表才能支持事务处理！其他的类型是不支持！**

**mysql> set autocommit=0;**

**mysql> delimiter //**

**mysql> start transaction;**

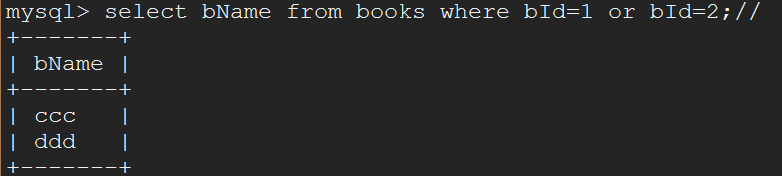
**-> update books set bName="ccc" where bId=1;**

**-> update books set bName="ddd" where bId=2;**

**-> commit;//**

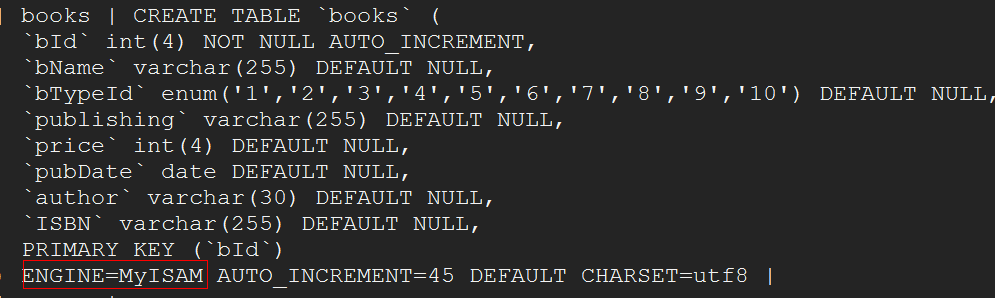
**测试，查看是否完成修改：**

**mysql> select bName from books where bId=1 or bId=2;//**

****

**我们测试回滚操作，首先看我们的数据库存储引擎是否为innodb**

**mysql> show create table books//\G**

****

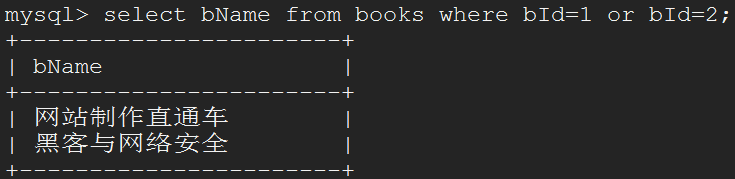
**为MyISAM无法成功启动事务，虽然提交了，却无法回滚**

**修改数据库存储引擎为innodb**

**mysql> alter table books engine=innodb;**

**mysql> alter table category engine=innodb;**

****

****

**重新开启事务，并测试回滚**

**mysql> set autocommit=0;**

**mysql> delimiter //**

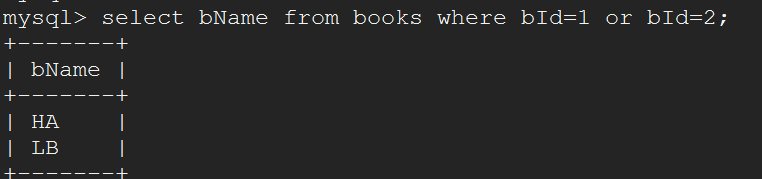
**mysql> start transaction;**

**-> update books set bName="HA" where bId=1;**

**-> update books set bName="LB" where bId=2;**

**-> commit;//**

**mysql> delimiter ;**

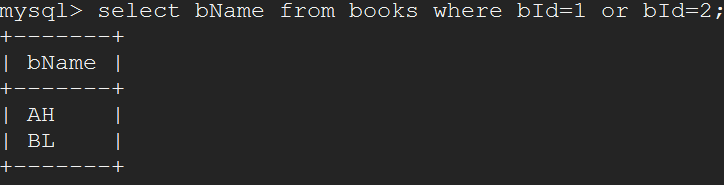
****

**无法回滚，因为我们commit已经提交了**

**mysql> delimiter //**

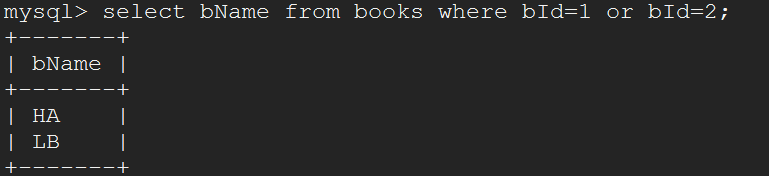
**mysql> start transaction; update books set bName="AH" where bId=1; update books set bName="BL" where bId=2;// 不提交**

**mysql> delimiter ;**

****

**回滚：**

**mysql> rollback;**

****

**恢复了**