存储过程

一、定义：存储过程(Stored Procedure),是一组为了完成特定功能的SQL 语句，集经编译后存储在数据库中，用户通过指定存储过程的名字并给出参数来执行。

二、分类：在SQL Server 的系列版本中，存储过程分为两类：系统提供的存储过程和用户自定义存储过程。

　　系统存储过程，主要存储master 数据库中，并以sp\_为前缀，并且系统存储过程主要是从系统表中获取信息，从而为系统管理员管理SQL Server。

常用系统存储过程有：

exec sp\_databases; --查看数据库

exec sp\_tables; --查看表

exec sp\_columns student;--查看列

exec sp\_helpIndex student;--查看索引

exec sp\_helpConstraint student;--约束

exec sp\_stored\_procedures;

exec sp\_helptext 'sp\_stored\_procedures';--查看存储过程创建、定义语句 经常用到这句话来查看存储过程

exec sp\_rename student;--修改表、索引、列的名称

exec sp\_renamedb myTempDB, myDB;--更改数据库名称

exec sp\_defaultdb 'master', 'myDB';--更改登录名的默认数据库

exec sp\_helpdb;--数据库帮助，查询数据库信息

 用户自定义存储过程是由用户创建，并能完成某一特定功能，如：查询用户所需数据信息的存储过程。

存储过程的优点；

      (1)重复使用。存储过程可以重复使用，从而可以减少数据库开发人员的工作量。

        (2)提高性能。存储过程在创建的时候就进行了编译，将来使用的时候不用再重新编译。一般的SQL语句每执行一次就需要编译一次，所以使用存储过程提高了效率。

        (3)减少网络流量 存储过程位于服务器上，调用的时候只须传递存储过程的名称以及参数，不用每次访问都传递很长的sql 语句。

        (4)安全性。参数化的存储过程可以防止SQL注入式的攻击。

(5)可维护性高。 更新存储过程通常比更改、测试以及重新部署程序集需要较少的时间和精力。

存储过程的缺点；

（1）开发调试差：无良好的IDE开发工具，存储过程的调试比一般SQL要复杂的多。

（2）可移植性差：由于存储过程将应用程序绑定到数据库上，因此使用存储过程封装业务逻辑将限制应用程序的可移植性。

三、语法：

定义存储过程的语法

CREATE PROC[EDURE] --存储过程名

@参数1 数据类型 ,

@参数n 数据类型 OUTPUT

AS

BEGIN

SQL语句

END

执行我们的存储过程exec 存储过程 参数1,参数2

注意，在SQL SERVER中，所有用户定义的变量都以“@”开头，OUTPUT关键字表示这个参数是用来输出的，AS之后就是存储过程内容了。只要将以上代码在“查询分析器”里执行一次，SQL SERVER就会在当前数据库中创建一个名为“UserLogin”的存储过程。 看到了在数据中调用存储过程的方法有两种（这里EXEC与EXECUTE等效）；

如果要删除一个存储过程，用drop像这样 DROP PROCEDURE UserLogin

如果要修改一个存储过程，用alter像这样 alter PROCEDURE UserLogin

# **四、[SQL 存储过程入门（变量）](https://www.cnblogs.com/lideng/archive/2013/04/11/3014407.html)**

变量分为局部变量和全局变量

局部变量是@开头，全局变量是@@开头，这里我们主要讲局部变量

看看简单的声明语法：

declare @ID varchar(**20**) --定义一个变量叫@ID

set @ID=**3** --变量赋值为3

print @ID --打印

select @ID=**1** --变量赋值为1

print @ID --打印

从上面看出来了，Set,与select都可以给变量赋值。

然后我们看看变量的运算，运算其实很简单，下面看看加减法

declare @ID varchar(**20**)

set @ID=**3**

print @ID

select @ID=**1**+@ID --将变量@id加1

print @ID

select @ID=(select **1**+**5**) --类似于@ID=1+5

print @ID

select @ID=(select **1**-@ID) --类似于@ID=1-@I

Dprint @ID

**区别**

1，表达式返回多个值时

表达式返回多个值时，使用 SET 赋值

declare @name varchar(128)

set @name= (select pm from px\_pm)

print @name

--出错信息为

[Err] 21000 - [SQL Server]子查询返回的值不止一个。当子查询跟随在 =、!=、<、<=、>、>= 之后，或子查询用作表达式时，这种情况是不允许的。

declare @name varchar(128)

set @name= (select pm from px\_pm where hh='0104472')

print @name

表达式返回多个值时，使用 SELECT 赋值

declare @name varchar(128)

select @name= pm from px\_pm

print @name @name --结果集中最后一个 username 列的值

2,表达式未返回值时

--表达式未返回值时，使用 SET 赋值

declare @name varchar(20)

set @name='jack'

set @name= (select pm from px\_pm where hh='0104472')

print @name

declare @name varchar(20)

set @name='jack'

set @name= (select pm from px\_pm where 1=2)

print @name --Null值

--表达式未返回值时，使用 SELECT 赋值

declare @name varchar(20)

set @name='jack'

select @name= pm from px\_pm where hh=’0177704472’

select @name= pm from px\_pm where hh=’0104472’

print @name --jack,保存原来的值

结果-------

jack

这里简单总结下

下表列出 SET 与 SELECT 的区别。

|  |  |  |
| --- | --- | --- |
| 条件 | set | select |
| 表达式返回多个值 | 出错 | 将返回的最后一个值赋给变量 |
| 表达式未返回值 | 变量被赋为null值 | 变量保持原值 |

在常规情况下，我们通常用Set给标量赋值，如初始化时，Set  @User = 'Admin'

 在sql语句中赋值一般用Select,如，select @name = username from userinfo where username='not'

# **五、[SQL 存储过程入门（流程控制）](https://www.cnblogs.com/lideng/archive/2013/04/12/3016583.html)**

存储过程的流程控制, if else,case ,while,这里没有for循环的 ，这些跟c,c++,c#等语言都差不过的。

**首先来看if  else使用**

if 条件begin

sql 语句end

else begin

sql语句end

declare @id int --声明个变量

set @id=**5 --设置厨初始值**

if(@id=**1**)

begin

print 'right'

End

else if(@id=**0**)

begin

print 'error'

End

else

begin

print 'default'

end

这里的begin ,end 也可以不写，不过建议写，这个像c++代码的{},表示一块，写出来层次更加清晰。

当然if 嵌套也是可以的。

**二,While的使用**

**Declare @i int,@sum int**

**set @i=0**

**set @sum=0**

**while @i<=100 --开始循环**

**Begin**

**set @sum=@sum+@i**

**set @i=@i+1 --自增一**

**End**

**print @sum**

**三,Case多重选择的使用**

Declare @iret int,@var varchar(**10**)

set @var='A'

select @iret=case when @var='A' then **0**

when @var='B' then **1**

when @var='C' then **2**

when @var='D' then **3**

else -**1** end

print @iret

上面的这些基本流程控制，基本能满足我们日常的开发需求了

六、事务

为什么要使用事务呢，事务有什么用呢，举个例子。

假设我们现在有个业务，当做成功某件事情的时候要向2张表中插入数据，A表，B表，我们插入的顺序是先插入A，再插入B表，如果都顺利插入成功了，当然没有问题，如果任意一张表插入失败了，而另一张表插入成功了，插入成功的表就是垃圾数据了。我们要判断，任意一张表插入失败都回滚，就是都不插入，这就是事务的基本使用。

**sql事务定义**

所谓**事务**是用户定义的一个数据库操作序列，是一个不可分割的工作单位。他包含的所有数据库操作命令作为一个整体一起向系提交或撤消 ，这些操作要么全做，要么全不做,

 例如在关系数据库中，一个事务可以是一条sql语句，或者是一组sql语句或者是整个程序。

**二，sql事务语句**

开始事务：BEGIN TRAN

提交事务：COMMIT TRANSACTION  
回滚事务：ROLLBACK TRANSACTION

事务通常是以begin transaction开始的，以commit或rollback 结束，commit表示提交，既提交事务的所有操作。具体的说就是将事务中所有对数据库的更新写到磁盘上的物理数据中去，事务正常结束。

Rollback表示回滚，在事务运行的过程中发生了某种故障，事务不能继续执行，系统将事务中对数据库的所有已完成的操作全部取消，回滚到事务开始时的状态。这里的操作

指对数据库的更新操作。

@@error是系统函数，当没有发生错误时返回0，如果发生错误时@@error != 0，并返回错误号。

并且每条语句下的@@error都会变化。

举例：有bank\_in 和bank\_out两个表，一个是转出表，一个是转入表。

create proc test

@ID\_out INT,

@ID\_in INT,

@num int

as

declare @error int =**0** --声明一个记录错误的变量

begin transaction

update bank\_out set num=num-@num where ID=@ID\_out

set @error =@error+**@@ERROR** --记录执行语句是否报错

update bank\_in set num=num+@num where ID=@ID\_in

set @error =@error+**@@ERROR**

if(@error !=**0**) --判断是否有@@error

begin

rollback transaction

end

else

begin

commit transaction

end

作业：

1. 计算门店月平均销量（每个货号）。

公式：门店月平均销量=本月销量\*50%+次月销量\*30%+次次月销量\*20%

提示：KDRQ BETWEEN DATEADD(MM,-1,GETDATE()) AND GETDATE()

KDRQ BETWEEN DATEADD(MM,-2,GETDATE()) AND DATEADD(MM,-1,GETDATE())

KDRQ BETWEEN DATEADD(MM,-3,GETDATE()) AND DATEADD(MM,-2,GETDATE())

1. 根据月平均销量/30计算天平均销量，然后根据商品销售天数上下限（px\_ts）计算门店备货的上下限数量。
2. 如果库存小于下限，则生成报货计划（货号，申请数量，品名，门店编号，申请日期，申请人），数量为上线数量-库存。如果库存大于下限，则不生成计划。