# Transact-SQL概述

Transact-SQL与其他的编程语言不同，它主要用于为处理大量数据提供必要的结构化处理能力，Transact-SQL语言侧重于对数据的操作以及对数据库的管理。

Transact-SQL主要包括：

数据定义语言(DDL)：创建数据表（关系）、删除数据表、修改数据表、建立索引和约束，以及创建其他数据库对象等。

数据操作语言(DML)：查询数据、添加数据、删除数据、修改数据。

数据控制语言(DCL)：控住安全性的命令。

Transact-SQL语句的语法元素组成：

* 关键字（系统已经定义的）
* 标识符（自己定义的）
* 数据类型（int(\*)、varchar(\*)、datetime、money…）
* 运算符(+、-、\*、/、=…)
* 表达式
* 函数(系统已经定义好的一些常用到的函数)
* 注释（单行注释用“--”、多行注释用”/\*”和”\*/”）

## 常量与变量

### 常量

其是表示特定数据值的符号，大部分的常量是由系统定义的。

其中字符串常量、二进制常量、Bit常量、Integer常量、Datetime常量、Decimal常量、Float和Real常量、Money常量为主要的使用对象。

### 变量

变量用来在内存中临时存储数据，所以变量的存在时间是有限制的，其存在的时间称为作用域。

1. 声明变量：

declare

@myContactID int,

@myTitle char(8);

go

--代码说明：声明一个int型变量@myContactID和一个8位char变量@myTitle

--要点解读：

--1、声明的变量必须要以“@”开头，且变量名要符合标识符的命名规范

--2、变量标识符后面要指定该变量的数据类型，且不能是text、ntext或image数据类型

--3、当需要定义多个变量时要使用逗号将他们隔开

1. 为变量赋值

--1.2.1 使用set语句进行赋值

declare @myContactID int,@myTitle char(8);

set @myContactID=12

set @myTitle='Mr.'

select @myContactID as myContactID,@myTitle as myTitle

go

--代码说明：使用declare声明两个变量，使用set分别为这两个变量进行赋值

--要点解读：

--1、变量赋值语句跟在变量声明语句之后，

--2、declare关键字可以声明多个变量，但set关键字一次只能为一个变量进行赋值

--3、多个set赋值语句之间无符号连接

--1.2.2 使用select语句进行赋值

declare @myContactID int,@myTitle char(8);

select @myContactID=12,@myTitle='Mr.'

select @myContactID as myContactID,@myTitle as myTitle

go

--代码说明：使用declare声明两个变量，使用select关键字分别为这两个进行赋值

--要点解读：

--1、变量赋值语句跟在变量声明语句之后

--2、declare关键字可以声明多个变量，且select关键字也可以同时为多个变量进行赋值

--3、复制变量之间需要使用逗号隔开

--4、在使用select关键字实现变量赋值时不能再由它实现查询功能

注意事项：

1、select的赋值功能与查询功能不能同时使用

2、在使用声明的变量时，必须要为其赋值（默认其值为null）

3、在使用变量时要注意其作用域

## 内置函数

内置函数是SQL Server中已经定义好的一些函数，在使用时可以直接进行调用。

表：内置函数类型：

|  |  |
| --- | --- |
| 函数 | 说明 |
| 行集函数 | 返回可在SQL语句中像表引用一样使用的对象 |
| 聚合函数 | 对一组数据进行运算，但返回一个汇总值 |
| 排名函数 | 对分区中的每一行均返回一个排名值 |
| 标量函数 | 对单一值进行运算，然后返回单一值。只要表达式有效，即可使用标量函数 |

表：标量函数

|  |  |
| --- | --- |
| 函数类别 | 说明 |
| 配置函数 | 返回当前配置信息 |
| 加密函数 | 支持加密、解密、数字签名和数字签名验证 |
| 游标函数 | 返回游标信息 |
| 数据类型函数 | 返回有关标识值和其他数据类型值的信息 |
| 日期和时间数据类型及函数 | 对日期和时间输入值执行运算，然后返回字符串、数字或日期和时间值 |
| 数学函数 | 对函数参数提供的输入值进行运算，然后返回数字值 |
| 元数据函数 | 返回有关数据库和数据库对象的信息 |
| ODBC标量函数 | 返回有关Transact-SQL语句中的标量ODBC函数的信息 |
| 复制函数 | 返回用于管理、监控和维护复制拓扑的信息 |
| 安全函数 | 返回有关用户和角色的信息 |
| 字符串函数 | 对字符串输入值执行运算，然后返回一个字符串或数字值 |
| 系统函数 | 执行运算后返回SQL Server事例中有关值、对象和设置的信息 |
| 系统统计函数 | 返回系统的统计信息 |
| 文本与图像函数 | 对文本或图像输入值或列执行运算，然后返回有关值的信息 |
| 触发器函数 | 返回触发器信息 |

表：常用内置函数

|  |  |  |  |
| --- | --- | --- | --- |
| 全局变量 | 返回类型 | 变量值 | 作用域 |
| @@CONNECTIONS | Integer | 自上次启动SQL Server以来连接或视图连接的次数 | 服务器 |
| @@CPU\_BUSY | Integer | 自上次启动SQL Server以来CPU的工作时间，单位为毫秒 | 服务器 |
| @@CURSOR\_ROWS | Integer | 最后打开的游标中当前存在的合格行的数量 | 连接 |
| @@DATEFIRST | Tinyint | 返回规定将周几设为每周的第一天：  Eg：1 对应星期一 2 对应星期二 | 连接 |
| @@DBTS | Varbinary | 当前数据库最后所使用的时间戳值 | 数据库 |
| @@ERROR | Integer | 最后执行的Transact-SQL语句的错误代码 | 连接 |
| @@FETCH\_STATUS | Integer | 最近一次被FETCH语句执行的最后游标的状态，而不是任何当前当前连接打开的游标的状态 | 连接 |
| @@IDENTITY | Numeric | 当前连接最后插入的标识值 | 连接 |
| @@IDLE | Integer | SQL Server自上次启动后闲置的时间，毫秒 | 服务器 |
| @@IO\_BUSY | Integer | SQL Server自上次启动后 用于执行输入和输出操作的时间，单位毫秒 | 服务器 |
| @@LANGID | Smallint | 当前所使用语言的本地语言标识符(ID) | 连接 |
| @@LANGUAGE | Nvarchar | 当前使用的语言名 | 连接 |
| @@LOCK\_TIMEOUT | Integer | 为当前连接设置的锁定超时时间，单位ms | 连接 |
| @@MAX\_CONNECTIONS | Integer | SQL Server上允许同时连接用户的最大数目 | 服务器 |
| @@MAX\_PRECISION | Tinyint | Decimal和numerical数据类型所用的最大精度值 | 服务器 |
| @@NESTLEVEL | Integer | 前当存储过程执行的嵌套层次(初始值为0) | 连接 |
| @@OPTIONS | Integer | 所有当前连接选项的信息 | 连接 |
| @@PACK\_RECEIVED | Integer | SQL Server自上次启动后，从网络上读取的输入数据包数目 | 服务器 |
| @@PACK\_SENT | Integer | SQL Server自上次启动后，写到网络上的输出数据包数目 | 服务器 |
| @@PACKET\_ERRORS | Integer | SQL Server自上次启动后，在连接时发生的网络数据包错误数目 | 服务器 |
| @@PROCID | Integer | 当前过程的存储过程标识符(ID) | 连接 |
| @@REMSERVER | Nvarchar(256) | 所使用的远程SQL Server数据库服务器的名称 | 连接 |
| @@ROWCOUNT | Integer | 最近一次执行T-SQL语句取得的行数 | 连接 |
| @@SERVERNAME | Nvarchar | 本地服务器名称 | 服务器 |
| @@SERVICENAME | Nvarchar | SQL Server的Windows服务名 | 服务器 |
| @@SPID | Smallint | 当前用户进程的服务器进程标识(ID) | 连接 |
| @@TEXTSIZE | Integer | 当前BLOB(text、image)数据的最大值 | 连接 |
| @@TIMETICKS | Integer | 一刻度的微秒数 | 连接 |
| @@TOTAL\_ERRORS | Integer | SQL Server自上次启动后，所遇到的磁盘读写错误次数 | 服务器 |
| @@TOTAL\_READ | Integer | SQL Server自上次启动后，读取磁盘的次数 | 服务器 |
| @@TOTAL\_WRITE | Integer | SQL Server上次启动后，写入磁盘的次数 | 服务器 |
| @@TRANCOUNT | Integer | 当前连接的活动事务数 | 连接 |
| @@VERSION | Nvarchar | SQL Server的产品信息 | 服务器 |
| GETANSINULL | Int | 返回当前数据库默认的null值；对ANSI空值返回1，如果没有定义ANSI空值则返回0 |  |

## 基本运算

基本运算包括：算术运算、逻辑运算、字符串处理、比较运算、日期运算以及对大值数据类型的支持。

### 算术运算

加(+)减(-)乘(\*)除(/)取模(%)运算

declare @myContactID1 int,@myContactID2 int

select @myContactID1=12,@myContactID2=@myContactID1+4

select @myContactID1 as myContactID1,@myContactID2 as myContactID2

--代码说明：声明两个int型变量，并使用select关键字为其赋值，其中@myContactID2的值为@myContactID1+4之后的值

### 逻辑运算

逻辑运算用于判断条件的真假。

And：逻辑与；

Or：逻辑或；

Not：逻辑非；

All：如果一系列的布尔表达式都为true，则返回true；

Any：如果一系列的布尔表达式有一个为true，则返回true；

Like：如果操作数与一种模式相匹配，则返回true；

In：如果操作数等于表达式列表中的一个，返回true。

select myContactID,

[myTitle]+'select' as my,

[myTitle]

from [dbo].[Table\_Test]

where [myContactID]=2 or [myContactID]=3

go

--代码说明：使用select关键字进行查询，并使用字符串的拼接功能和列的重命名

--要点解读：

--1、where和or关键字来指定查询的条件

--2、使用“+”进行字符串拼接

--3、使用as关键字为查询到的列命名

### 字符串处理

1. 使用字符串连接符(+)将两个及两个以上的字符串连接起来

select 'abc'+'def'+'ghi'

go

--代码说明：使用“+”将字符串连接起来 这一点与其他的编程语言一样

1. 常用的字符串处理函数

表：常用的字符串处理函数

|  |  |
| --- | --- |
| 函数描述 | 含义 |
| ASCII(character\_expression) | 返回字符串表达式最左端字符的ASCII代码值 |
| CHAR(integer\_expression) | 将intASCII代码转换为字符的字符串函数 |
| LEFT(character\_expression, integer\_expression) | 返回从字符串左边开始指定个数的字符串 |
| LEN(string\_expression) | 返回给定字符串表达式的字符个数，其中不包括尾随空格 |
| RIGHT(character\_expression, integer\_expression) | 返回从字符串右边开始指定个数的字符串 |
| LOWER(character\_expression) | 将大写全部转换为小写后返回 |
| UPPER(character\_expression) | 将小写全部转换为大写后返回 |
| LTRIM(character\_expression) | 删除字符串表达式左边的空格后返回 |
| RTRIM(character\_expression) | 删除字符串表达式右边的空格后返回 |
| REPLACE(‘string\_expression1’, ‘string\_expression2’,’ string\_expression3’) | 替代：用string\_expression3替代string\_expression1中出现的所有第二个给定字符串表达式 |
| SPACE(integer\_expression) | 返回由int个空格组成的字符串 |
| STR(float\_expression[,length[,decimal]]) | 由数字数据转换来的字符数据 |
| STUFF(character\_expression,start,length,  character\_expression) | 删除指定长度的字符串，并在指定的起始点插入另一组字符串 |
| SUBSTRING(expression,start,length) | 返回字符、binary、text、或image表达式的一部分 |

### 比较运算符

比较运算符是判断某一个条件是否成立，其返回值是一个bool值，条件为真则返回true、条件为假则返回false

比较运算符的分类：

等于：=

大于：>

大于等于：>=

小于：<

小于等于：<=

不等于：！= <>

### null与空判断

null被称为空值，空值是一个未知的值，空值可以是任何类型的值，所以不能用它来进行值判断，在SQL Server中提供is关键字来判断该值是否为空。

select \* from [dbo].[Table\_Test] where [myTitle] is null

go

--代码说明：查询[myTitle]为空的表项

### 日期运算

declare @mydate datetime

set @mydate = '2013-4-5'

select @mydate - 7 as mydate

go

--代码说明：使用declare关键字声明@mydate变量

-- 使用set关键字为@mydate变量赋值

-- 使用select关键字和“-”对日期进行运算

表：日期时间函数

|  |  |
| --- | --- |
| 函数描述 | 含义 |
| Year(date) | 返回表示指定日期中年份的整数 |
| Month(date) | 返回表示指定日期中月份的整数 |
| Day(date) | 返回表示指定日期中天的日期部分的整数 |
| Getdate() | 返回当前系统日期和时间（服务器） |
| Dateadd(datepart,munber,date) | 返回date值加上datepart参数指定的时间间隔得到的新的datetime值 |
| Datediff(datepart,date1,date2) | 返回date1和date2的时间间隔，其单位由datepart参数指定，返回数值的类型为整数 |
| Datename(datepart,date) | 返回代表指定日期的指定日期部分的字符串 |
| Datepart(datepart,date) | 返回代表指定日期的指定日期部分的整数 |

### 大值数据类型

varchar(max)、nvarchar(max)、varbinary(max)统称为大值数据类型。

## 流程控制

### if…else结构

declare @number int;

set @number = 50;

if @number > 100

print 'The number is large.';

else

if @number < 10

print 'The number is small.';

else

print 'The number is medium.';

go

--代码说明：使用declare关键字声明int型变量@number

-- 使用set关键字为声明的@number变量进行赋值

-- if...else的判断结构（当代码超过一行时要使用begin-end）

-- print关键字显示结果

### if exists结构

declare @avgWeight decimal(8,2),@BikeCount int

if exists(select \* from [dbo].[Table\_Storage] where [product] like 'bike1')

begin

set @BikeCount = (select count(\*) from [dbo].[Table\_Storage] where [product] like 'bike1');

set @avgWeight = (select avg(weight) from [dbo].[Table\_Storage] where [product] like 'bike1');

print 'There are ' + cast(@BikeCount as varchar(3)) + ' bike1.'

print 'The average weight of the bike1 is ' + cast(@avgWeight as varchar(8)) + '.';

end

else

begin

print 'there is not bike1';

end

go

--代码说明：if exists()如果表中存在bike1，则执行begin-end之间的代码（计算自行车数量及计算平均质量）

-- 否则执行else之后的begin-end之间的代码

### begin…end结构

当if…else后面跟的控制语句多于一条时，需要使用begin…end代码块。

begin…end将代码封装成一个过程，这个过程可以将多条命令看成一个整体，当作一条语句看待。想到与C#代码中的花括号

begin…end语句的适用范围:

1. while循环需要包含语句块
2. case函数的元素需要包含语句块
3. if或else子句需要包含语句块
4. begin和end还可以嵌套使用

### while循环

当while之后的条件为true时，将执行其后的循环体；当条件为false时，将跳出该循环。在该循环中可以使用break关键字结束该循环，continue关键字结束本次循环。

select \* from [dbo].[Table\_Storage]

while (select avg([weight]) from [dbo].[Table\_Storage]) < 80

begin

update [dbo].[Table\_Storage] set [weight] = [weight] \* 2

if(select max([weight]) from [dbo].[Table\_Storage]) > 60

break

else

continue

end

select \* from [dbo].[Table\_Storage]

print N'循环完毕'

go

--代码说明：第一个select展示执行循环之前的表值

-- while执行循环操作，当while之后的条件为真时执行该循环

-- 在begin-end代码块中：修改表中的数据，如果weight大于60时结束该循环，否则结束该循环

-- 第二个select展示执行循环之后的表值

### GOTO语句

GOTO语句实现代码间的跳转功能，

其结构体为：

标签：

程序体

GOTO 标签

declare @message char(10)

set @message = 'test'

label1:

begin

print N'GOTO到这里'

set @message = 'stop'

end

if @message = 'test'

GOTO label1

go

--代码说明：使用declare关键字来声明一个char型变量@message

-- 使用set关键字来为@message变量赋值

-- 定义label1标签及其程序体，使用if条件判断来执行GOTO语句的跳转

### CASE语句

case函数允许按列值显示可选项。数据中的更改时临时的，没有对数据进行永久更改。

case函数语法：

第一种：

case 字段名称

when 记录 then 结果值

…

else 结果值

end

第二种：

case

when 布尔表达式 then 结果集

…

else

end

--将某个表达式与一组简单表达式进行比较已确定结果

select [product] =

case [product]

when 'bike1' then N'自行车1'

when 'bike2' then N'自行车2'

else N'自行车'

end

from [dbo].[Table\_Storage]

go

--代码说明：select关键字进行查询操作

-- case-end，要搜索的字段名称为[product]

-- when-then,如果存在该记录则返回then之后的结果值

--

--搜索函数计算一组布尔表达式

select [myTitle] =

case

when [myTitle] is null then N'无标题'

when [myTitle] is not null then N'有标题'

end

from [dbo].[Table\_Test]

go

--代码说明：select关键字进行查询操作

-- case-end代码体中为when 布尔表达式 then 结果集

## 游标

游标相当于其他高级语言中的指针。

游标的五个生命周期：

1. 声明游标：为游标指定获取数据时所使用的select语句，并且在declare关键字定义游标名字时不需要使用“@”符号。
2. 打开游标：检索数据并填充游标。
3. 操作游标：fetch使游标移到下一条记录或之后的N条位置。
4. 关闭游标：释放数据，但保留select语句。
5. 释放游标：释放相关的内存，并删除游标的定义。

**游标选项：**

游标的选项也即游标的附加选项，其用来扩展游标的能力，使其能够管理和跟新数据。其中常用的附加游标包括：动态游标、可滚动游标、将游标作为参数传递的选项和将游标存储于变量中的选项。

**游标的作用域：**

游标的作用域有两种分别是：局部作用和全局作用。

**其中常用的两个全局作用域：**

@@cursorrows全局变量将会返回游标中的行数；

@@fetchstatus全局变量可以返回在最近一次执行fetch命令之后游标的状态：

0---最近一次执行fetch命令成功获取到一行数据；

1---最近一次执行fetch命令到游标结果集的尾部；

2---最近一次执行fetch命令获取的行不可用，改行已经被删除。

### 声明游标

**声明游标的语法：**

declare 游标名称 cursor

相关参数

for select语句

### 打开游标

open 游标名称

### 操作游标

使用语法：

fetch

[ [ next | prior | first | last

| absolute { n | @nvar}

| relative { n | @nvar}

]

from

]

{ { [global] cursor\_name } | @cursor\_variable\_name }

[ into variable\_name [ ,…n ] ]

**next**：返回紧跟当前行之后的结果行，并且当前行递增为结果行。如果fetch next为对游标的第一次提前操作，则返回结果集中的第一行。next为默认的游标提取项。

**prior**：返回紧跟当前行前面的结果行，并且当前行递减为结果行。如果fetch prior为对游标的第一次操作，则没有行返回并且游标置于第一行之前。

**Frist**：返回游标中的第一行，并将其作为当前行。

**last**：返回游标中的最后一行，并将其作为当前行。

**absolute { n | @nvar}**：

如果n或@nvar为正数，返回从游标头开始的第n行，并将返回的行变成新的当前行。如果n或@nvar为负数，返回游标尾之前的第n行，并将返回的行变成新的当前行。

如果n或@nvar为0，则没有行返回。

n必须为整数常量，且@nvar必须为smallint、tinyint或int

**relative { n | @nvar}：**

如果n或@nvar为正数，返回从当前行之后的第n行，并将返回的行变成新的当前行。如果n或@nvar为负数，返回当前行之前的第n行，并将返回的行变成新的当前行。

如果n或@nvar为0，则没有行返回。

如果对游标的第一次提取操作时，将fetch relative的n或@nvar指定为负数或0，则没有行返回

n必须为整数常量，且@nvar必须为smallint、tinyint或int

**global**：指定cursor\_name指的是全局游标。

**cursor\_name**：要从中进行提取的开放游标的名称。如果同时有以cursor\_name作为名称的全局和局部游标存在，若指定为global则cursor\_name对应全局游标，未指定global则对应局部游标。

**@cursor\_variable\_name**：游标变量名，引用要进行提取操作的打开的游标。

**into variable\_name [ ,…n ]**：允许将提取操作的列数据放到局部变量中。列表中的各个变量从左到右与游标结果集中的相应列相关联。各个变量的数据类型必须与相应结果集的数据类型匹配，或是结果集数据类型所支持的隐性转换。变量的数目必须与游标选择立标中的列的数目一致。

### 关闭游标

close 游标名称

### 释放游标

deallocate 游标名称

代码展示：

declare cur cursor --声明一个游标变量

--INSENSITIVE --相关参数：INSENSITIVE将会把所有的结果集临时存放到tempdb中

for select \* from [dbo].[Table\_Test] --为游标变量指定select语句

open cur --打开声明的游标

fetch next from cur --对游标的操作(游标向后以一行)

while @@FETCH\_STATUS = 0

begins

fetch next from cur

end

close cur --关闭游标

deallocate cur --删除游标

go

## 统计函数

不需要得到数据表中的任何一条数据，需要得到数据表中的一条统计数据。

常见的统计数据：

sum()：返回在某一集合上对数值表达式求得的总和；

avg()：返回在某一集合上对数值表达式求得的平均数；

max()：返回在某一集合上数值表达式中的最大值；

min()：返回在某一集合上数值表达式中的最小值；

count()：返回符合条件的记录数。

# 存储过程

存储过程的简介

存储过程是一组为了完成特定功能的SQL语句集合，它经编译后存储在数据库中。

存储过程分为三类：

用户定义存储过程、CLR存储过程、系统存储过程

## 设计与创建存储过程

设计存储过程时一般要遵循的原则：

* 存储过程名称避免使用sp的前缀；
* 尽量不要使用临时连接

存储过程：

1. 使用create procedure 定义SQL语句集合，但不能使用某些关键字；
2. 其他数据库对象均可在存储过程中创建，可以引用在同一存储过程中创建的对象；
3. 可以在存储过程内引用临时表，如果在存储过程中创建临时表，临时表仅在该过程中可用；
4. 如果执行的存储过程将调用另一个存储过程，则被调用的存储过程可以访问由第一个存储过程创建的所有对象，包括临时表在内；(完全访问权限)
5. 存储过程中参数的最大数目为2100，存储过程中局部变量的最大数目仅受可用内存的限制。(存储过程的容量限制)



创建存储过程：

用create procedure语句创建存储过程

if OBJECT\_ID('Test3','P') is not null

drop procedure Test3;

go

create procedure Test3

@name nvarchar(15)

as

select \*

from [TestDB].[dbo].[products]

where [prd\_name] = @name

go--代码说明：带@name参数的存储过程

执行该代码之后，将会生成一个存储过程，右击执行该存储过程。

## 修改存储过程

修改存储过程一般有两种方式：一种方式是利用该存储过程右击的修改选项进行修改，简单便捷但是适用于没有进行加密的存储过程，另一种方式为利用alter procedure语句进行存储过程的修改。

对于加密的存储过程只能用alter procedure语句来进行修改

--使用create procedure语句和with encryption语句创建加密的存储过程

--得到产品的全部数据

if OBJECT\_ID('products\_info','P') is not null

drop procedure products\_info;

go

create procedure products\_info

with encryption

as

select \*

from [TestDB].[dbo].[products]

go

exec products\_info --执行存储过程(无参)

--使用create procedure语句对存储过程进行修改

alter procedure products\_info

@productName nvarchar(15)

as

select [prd\_name],[prd\_date],[prd\_color]

from [TestDB].[dbo].[products]

where [prd\_name] = @productName

go

exec products\_info 产品1 --有一个参数

1. 修改之后的存储过程未加密，若要加密需加上with encryption语句
2. 使用exec关键字执行存储过程时后面跟参数(参数若为字符串也可省略引号)，当有多个参数时按顺序排列用逗号隔开

## 执行存储过程

执行存储过程也有两种方式：一种是通过界面的执行存储过程选项来实行，另一种则是用execute关键字来实现执行存储过程。(execute可以简写为exec)

eg：execute products\_info 1501,产品1 --有一个参数

## 删除存储过程

删除存储过程有两种方式：其一使用删除选项完成删除命令，其二使用drop procedure关键字来实现删除命令。

--使用drop procedure删除存储过程

drop procedure [dbo].[Test1]

go

## 常用系统存储过程

* sp\_help存储过程

获取有关数据库对象、数据类型等结构的信息（可以为存储过程也可以为为某张表）

exec sp\_help [Test2]

go

exec sp\_help [products]

go

* sp\_rename、sp\_renamedb存储过程

sp\_rename存储过程用于更改数据库中创建的对象的名称，(表、索引、列、别名、数据类型)

sp\_renamedb用于更改数据库的名称

exec sp\_rename '[Table\_Test].[myTitle]','tile','column'

# 函数

函数和存储过程类似，允许用户自定义函数，方便实现业务逻辑的可重用性。函数是使用零个、一个或多个输入值，返回一个数值或表格。

## 函数概述

**函数的结构**

函数分为函数头和函数体，函数头一般有如下结构：

* 具有架构、所有者名称的函数名称
* 输入参数名称和数据类型
* 可以用于输入参数的选项
* 可以用于返回参数的选项

函数体，也称为正文，其定义了函数将要执行的操作或逻辑

* 执行函数逻辑的一个或多个T-SQL语句
* .NET程序集的引用

**函数的类型**



函数与存储过程的选取：

当存储过程返回单个结果集，则使用定义表值函数来代替存储过程具有更好的效率；如果存储过程计算标量值，则定义标量函数。

如果存储过程满足如下条件，则表值函数较存储过程更具优势：

* 逻辑上可以用单个SELECT语句表达，推荐使用内联表值函数处理
* 存储过程不执行更新操作，推荐使用函数
* 不需要动态EXECUTE语句，推荐使用函数
* 如果存储过程返回的是一个结果集，推荐使用函数
* 如果存储过程的用途是生成要加载到临时表的中间结果，接着在SELECT语句中查询临时表，则推荐使用函数



内联表值函数语法：

create function [ owner\_name. ] function\_name

( [ { @parameter\_name [ as ] scalar\_parameter\_date\_type

[ =default ] }

[ ,…n ]

]

)

returns TABLE

[ with encryption ] --加密

[ as ]

return [ ( [ select\_stmt ] ) ]

[ ; ]

* TABLE：指定表值函数的返回值为表，只有常量和@local\_variables可以传递到表值函数
* select\_stmt：定义内联表值函数的返回值的单个SELECT语句

多语句表值函数的语法结构

create function [ owner\_name. ] function\_name

( [ { @parameter\_name [ as ] scalar\_parameter\_date\_type

[ =default ] }

[ ,…n ]

]

)

returns @return\_variable TABLE < table\_type\_definition >

[ with encryption ] --加密

[ as ]

begin

function\_body

return

end

* < table\_type\_definition >：定义T-SQL函数的表数据类型，表始终放在主文件组中。

## 函数创建

代码实例：

创建用户定义函数

-------------------------------------------------------------------------

--创建用户定义函数

--功能：用于返回从输入日期中提取的月份信息

--------------------------------------------------------------------------

IF OBJECT\_ID(N'GetMonthDay',N'IF') IS NOT NULL --判断该名称函数是否已经存在，若存在则将其删除

DROP FUNCTION GetMonthDay

GO

CREATE FUNCTION GetMonthDay --创建函数名为GetMonthDay的函数

(@Date DATETIME) --函数的参数名称及其类型

RETURNS INT --函数返回值的类型

AS

BEGIN --函数体

RETURN DATEPART(MONTH,@Date) --函数主体的执行语句

END;

GO

SELECT [dbo].[GetMonthDay](CONVERT(DATETIME,'05/05/2013')) AS DayOfMonth --调用创建的GetMonthDay函数

GO

创建内联表值函数事例

---------------------------------------------------------------------------------

--创建内联表值函数

--功能：用于根据颜色返回数据表的指定信息

---------------------------------------------------------------------------------

IF OBJECT\_ID(N'GetProducts',N'IF') IS NOT NULL --判断该名称函数是否已经存在

DROP FUNCTION GetProducts --若存在则删除

GO

--创建表值函数

CREATE FUNCTION GetProducts(@Color NVARCHAR(48)) --函数名、参数名称、参数类型

RETURNS TABLE --函数返回的数据类型

AS

RETURN --函数的主体（一个select语句）

(

SELECT \* FROM [dbo].[products] WHERE [prd\_color] = @Color

);

GO

--使用函数

SELECT \* FROM [dbo].[GetProducts]('Color1');

GO

## 函数执行

**函数执行的位置：**

* 在T-SQL语句中，

eg：SELECT \* FROM [dbo].[GetProducts]('Color1');

* 在调用该函数的应用程序中
* 在另一个用户定义函数的定义中，

eg：RETURN DATEPART(MONTH,@Date)

* 用于参数化视图或改进索引视图的功能中
* 用于在表中定义列，如使用newid()设置列的默认值
* 用于为列定义CHECK约束
* 用于替换存储过程

## 函数删除

使用DROP FUNCTION语句来删除已经创建的函数

## 内置函数

1. **聚合函数**

聚合函数对一组值执行计算，并返回单个值。

聚合函数只能在一下位置作为表达式使用：

* SELECT语句的选择列表
* COMPUTE或COMPUTE BY子句
* HAVING子句

**表：**常见内置函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| AVG | 返回组中各值的平均值 |
| CHECKSUM | 返回按照表的某一行或某一组表达式计算出来的校验和值 |
| CHECKSUM\_AGG | 返回组中各值的校验和 |
| COUNT | 返回组中的项数 |
| COUNT\_BIG | 返回组中的项数 |
| GROUPING | 包含用于对数据进行分组的表达式 |
| MAX | 返回表达式中的最大值 |
| MIN | 返回表达式中的最小值 |
| SUM | 返回表达式中所有值的和或仅非重复值的和 |
| STDEV | 返回指定表达式中所有值的标准偏差 |
| STDEVP | 返回指定表达式中所有值的总体标准偏差 |
| VAR | 返回指定表达式中所有值的方差 |
| VARP | 返回指定表达式中所有值的总体方差 |

1. **配置函数**

配置函数一般用于返回系统的配置信息。

**表：**常见配置函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| @@DATEFIRS | 指定的每周的第一天 |
| @@DBTS | 返回当前数据库的当前timestamp数据类型的值 |
| @@LANGID | 返回当前使用语言的本地语言标识符(ID) |
| @@LANGUAGE | 返回当前所有语言的名称 |
| @@LOCK\_TIMEOUT | 返回当前会话的当前锁定超时设置(ms) |
| @@MAX\_CONNECTIONS | 返回实例允许同时进行的最大用户连接数 |
| @@MAX\_PRECISION | 返回decimal和numeric类型所用精度级别 |
| @@NESTLEVEL | 返回对本地服务器上执行的当前存储过程的嵌套级别 |
| @@OPTIONS | 返回有关当前SET选项的信息 |
| @@REMSERVER | 返回远程数据库服务器在登录记录中显示的名称 |
| @@SERVERNAME | 返回运行SQL Server的本地服务器的名称 |
| @@SERVICENAME | 返回SQL Server正在其下运行的注册表项的名称 |
| @@SPID | 返回当前用户进程的会话ID |
| @@TEXTSIZE | 返回SET语句中TEXTSIZE选项的当前值 |
| @@VERSION | 返回当前安装的版本、处理器体系结构、生成日期和操作系统 |

1. **游标函数**

游标函数用于返回有关游标的信息，所有游标函数都是非确定性的。

**表：**游标函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| @@CURSOR\_ROWS | 返回连接打开的上一个游标中当前限定行的数目 |
| @@FETCH\_STATUS | 返回连接当前打开的任何游标发出的上一条游标FETCH语句的状态 |
| CURSOR\_STATUS | 它允许存储过程的调用方确定该存储过程是否已为给定的参数返回了游标和结果集 |

1. 日期和时间函数

日期和时间函数是对日期和时间输入值执行操作，并返回一个字符串、数字值或日期和时间值。

**表：**日期和时间函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| DATEADD | 通过向指定日期添加间隔，返回新的datetime值 |
| DATEIFF | 返回两个日期之间的时间间隔 |
| DATENAME | 返回表示指定日期的部分字符串 |
| DATEPART | 返回表示指定日期的部分整数 |
| DAY | 返回一个整数，表示指定日期的“天“部分 |
| GETDATE | 返回当前系统的日期和时间 |
| GETUTCDATE | 使用DT\_DBTIMESTAMP格式返回以UTC时间表示的系统当前时间 |
| MONTH | 返回表示指定日期的“月”部分的整数 |
| YEAR | 返回表示指定日期的“年”的整数 |

1. 数学函数

算术函数、三角函数和其他函数

**表：**数学函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| ABS | 返回指定数值表达式绝对值的数学函数 |
| ACOS | 返回反余弦值（弧度表示） |
| ASIN | 返回反正弦值（弧度表示） |
| ATAN | 返回反正切函数（弧度表示） |
| ATN2 | 返回正切值（弧度表示） |
| CEILING | 返回不带小数部分并且不小于其参数值的最小数字 |
| COS | 返回三角余弦值 |
| COT | 返回三角余切值 |
| DEGREES | 返回以弧度指定的角的相应角度 |
| EXP | 返回指定float表达式的指数值 |
| FLOOR | 返回小于或等于指定数值表达式的最大整数 |
| LOG | 返回指定float表达式的自然对数 |
| LOG10 | 返回指定float表达式的常用对数 |
| PI | 返回PI的常量值 |
| POWER | 返回指定表达式的指定幂的值 |
| RADIANS | 返回弧度值 |
| RAND | 返回0到1之间的随机float值 |
| ROUND | 返回一个数值表达式，舍入指定的长度或精度 |
| SIGN | 返回指定表达式的正号(+1)、零(0)或负号(-1) |
| SIN | 返回正弦值 |
| SQRT | 返回指定表达式的平方根 |
| SQUARE | 返回指定表达式的平方 |
| TAN | 返回输入表达式的正切值 |

1. **元数据函数**

元数据函数用于返回有关数据库和数据库对象的信息，函数获取的信息涉及数据库的底层结构信息。

**表：**元数据函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| @@PROCID | 返回T-SQL当前模块的对象标识符(ID) |
| COL\_LENGTH | 返回列的定义长度（以字节为单位） |
| COL\_NAME | 根据指定的对应表标识号和列标识号返回列的名称 |
| COLUMNPROPERTY | 返回有关列或过程参数的信息 |
| DATABASEPROPERTY | 返回指定数据库和属性名的命名数据库属性值 |
| DATABASEPROPERTYEX | 返回指定数据库的指定数据库选项或属性的当前设置 |
| DB\_ID | 返回数据库标识(ID)号 |
| DB\_NAME | 返回数据库名称 |
| FILE\_ID | 返回当前数据库中给定逻辑文件名的文件标识(ID)号 |
| FILE\_IDEX(T-SQL) | 返回当前数据库中的数据、日志或全文文件的指定逻辑文件名的文件标识(ID)号 |
| FILE\_NAME | 返回给定文件标识(ID)号的逻辑文件名 |
| FILEGROUP\_ID | 返回指定文件组名称的文件组标识(ID)号 |
| FILEGROUP\_NAME | 返回指定文件组标识(ID)号的文件组名 |
| FILEGROUPPROPERTY | 提供文件组和属性名时，返回指定的文件组属性值 |
| FILEPROPERTY | 指定文件名和属性时，返回指定的文件名属性 |
| fn\_listextendedproperty | 返回数据库对象的扩展属性值 |
| FULLTEXTCATALOGPROPERTY | 返回有关全文目录属性的信息 |
| FULLTEXTSERVICEPROPERTY | 返回有关全文服务级别属性的信息 |
| INDEX\_COL | 返回索引列名称 |
| INDEXKEY\_PROPERTY | 返回有关索引键的信息 |
| INDEXPROPERTY | 根据指定的表标识号、索引或统计信息名称以及属性名称，返回已命名的索引或统计信息属性值 |
| OBJECT\_ID | 返回架构范围内对象的数据库对象标识号 |
| OBJECT\_NAME | 返回架构范围内对象的数据库对象名称 |
| OBJECTPROPERTY | 返回当前数据库中架构范围内对象的有关信息 |
| OBJECTPROPERTYEX | 返回当前数据库中架构范围内对象的有关信息 |
| SQL\_VARIANT\_PROPERTY | 返回有关sql\_variant值的基本数据类型和其他信息 |
| TYPE\_ID | 返回指定数据类型名称的ID |
| TYPE\_NAME | 返回指定类型ID未限定的类型名称 |
| TYPEPROPERTY | 返回有关数据类型的信息 |

1. **排名函数**

排名函数为分区中的每一行返回一个排名值，根据所使用的函数，某些行可能与其他行接收相同值

**表：**排名函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| RANK | 返回结果集的分区中每行的排名 |
| DENSE\_RANK | 返回结果集分区中行的排名，在排名中没有任何间断 |
| NTILE | 将有序分区的行分发到指定数目的组中 |
| ROW\_NUMBER | 返回结果集分区中行的序号，每行分区的第一行从1开始 |

1. **行集函数**

行集函数将返回一个可用于代替T-SQL语句中表引用的对象，所有行集函数都具有不确定性。

**表：**行集函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| CONTAINSTABLE | 返回具有零行、一行或多行的表，这些行的列中包含基于字符类型的数据是单个词语和短语的完美匹配或模糊匹配、某个词在一定范围内的近似词或者加权匹配项 |
| FREETEXTTABLE | 为符合下述条件的列返回行数为零或包含一行或多行的表：这些行包含基于字符的数据类型，其中的值符合指定的freetext\_string中文本的含义，但不一定具有完全相同的文本语言 |
| OPENDATASOURCE | 不使用链接服务器的名称，而提供特殊的连接信息，并将其作为四部分对象名的一部分 |
| OPENQUERY | 对给定的链接服务器执行指定的传递查询 |
| OPENROWSET | 包含访问OLEDB数据源中的远程数据所需的全部连接信息 |
| OPENXML | OPENXML通过XML文档提供行集视图 |

1. **安全函数**

安全函数返回对管理安全性有用的信息。

**表：**安全函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| CURRENT\_USER | 返回当前用户的名称，此函数等价于USER\_NAME() |
| Sys.fn\_builtin\_permisions | 返回对服务器内置权限层次结构的说明 |
| Has\_Perms\_By\_Name | 评估当前用户对安全对象的有效权限 |
| IS\_MEMBER | 指示当前用户是否为指定Windows组或SQL Server数据库角色的成员 |
| IS\_SRVROLEMEMBER | 指示SQL Server登录名是否为指定固定服务器角色的成员 |
| PERMISSIONS | 返回一个包含位图的值，该值指示当前用户的语句、对象或列权限 |
| SCHEMA\_ID | 返回与结构名称关联的架构ID |
| SCHEMA\_NAME | 返回与架构名称关联的架构名称 |
| SESSION\_USER | 返回当前数据库中当前上下文的用户名 |
| SETUSER | 允许sysadmin固定服务器角色的成员或db\_owner固定数据库角色的成员模拟另一用户 |
| SUSER\_ID | 返回用户的登录标识号 |
| SUSER\_SID | 返回指定登录名的安全标识号(SID) |
| SUSER\_SNAME | 返回与安全标识号(SID)关联的登录名 |
| SYSTEM\_USER | 当位指定默认值时，允许将系统为当前登录提供的值插入表中 |
| SUSER\_NAME | 返回用户的登录表示名 |
| USER\_ID | 返回数据库用户的标识号 |
| USER\_NAME | 基于指定的标识号返回数据库用户名 |

1. **字符串函数**

字符串函数对字符串输入值执行操作，并返回字符串或数值。

**表：**字符串函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| ASCII | 返回字符串表达式中最左侧字符的ASCII代码值 |
| CHAR | 将intASCII代码转换为字符 |
| CHARINDEX | 返回字符串中指定表达式的开始位置 |
| DIFFERENCE | 返回一个整数值，指示两个字符表达式的SOUNDEX值之间的差异 |
| LEFT | 返回字符串中从左边开始指定个数的字符 |
| LEN | 返回指定字符串表达式的字符数，其中不包含尾随空格 |
| LOWER | 将大写字符数据转换为小写字符数据后返回字符串表达式 |
| LTRIM | 返回删除了前导空格之后的字符串表达式 |
| NCHAR | 返回具有指定整数代码的Unicode字符 |
| PATINDEX | 返回指定表达式中某模式第一次出现的起始位置；如果在全部有效的文本和字符数据类型中没有找到该模式，则返回零 |
| QUOTENAME | 返回带有分隔符的Unicode字符串，分隔符的加入可使输入的字符串成为有效的SQL Server分割标识符 |
| REPLACE | 用第三个表达式替换第一个表达式中出现的所有第二个指定字符串表达式的匹配项 |
| REPLICATE | 以指定的次数重复字符表达式 |
| REVERSE | 返回字符表达式的逆向表达式 |
| RIGHT | 返回字符串中从右边开始指定个数的字符 |
| RTRIM | 截断所有的尾随空格后，返回其字符串结果 |
| SOUNDEX | 返回一个由四个字符组成的代码，用户评估两个字符串的相似性 |
| SPACE | 返回由重复的空格组成的字符串 |
| STR | 返回由数字数据转换来的字符数据 |
| STUFF | 删除指定长度的字符，并在指定的起点处插入另一组字符 |
| SUBSTRING | 返回字符表达式、二进制表达式、文本表达式或图像表达式的一部分 |
| UNICODE | 按照Unicode标准的定义，返回输入表达式的第一个字符的整数值 |
| UPPER | 返回小写字符数据转换为大写的字符表达式 |

1. **系统函数**

数据库提供了系统级的消息函数，对数据库中的值、对象和设置进行操作并返回有关信息。

**表：**系统函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| APP\_NAME | 返回当前会话的应用程序名称 |
| CASE | 计算条件列表并返回多个可能表达式之一 |
| CAST和CONVERT | 将一种数据类型的表达式显式转换为另一种数据类型的表达式 |
| COALESCE | 返回其参数中第一个非空表达式 |
| COLLATIONPROPERTY | 返回指定序列规则的属性 |
| COLUMNS\_UPDATED | 返回varbinary位模式，它指示表或视图中插入或更新了哪些列 |
| CURRENT\_TIMESTAMP | 返回当前日期的和时间 |
| CURRENT\_USER | 返回当前用户的名称，(等价于USER\_NAME()) |
| DATALENGTH | 返回用于表示任何表达式的字节数 |
| @@ERROR | 返回执行上一个T-SQL语句的错误号 |
| ERROR\_LINE | 返回发生错误的行号，该错误导致运行TRY…CATCH构造的CATCH |
| ERROR\_MESSAGE | 返回导致TRY…CATCH构造的CATCH块运行错误的消息文本 |
| ERROR\_NUMBER | 返回错误的错误号，该错误导致运行TRY…CATCH构造的CATCH |
| ERROR\_PROCEDURE | 返回在其中出现了导致TRY…CATCH构造的CATCH块运行的错误存储过程或触发器的名称 |
| ERROR\_SEVERITY | 返回导致TRY…CATCH构造的CATCH块运行的错误严重级别 |
| ERROR\_STATE | 返回导致TRY…CATCH构造的CATCH块运行的错误状态号 |
| fn\_helpcollations | 返回SQL Server所支持的所有排列规则的列表 |
| fn\_servershareddrives | 返回群集服务器使用的共享驱动器的名称 |
| fn\_virtualfilestats | 返回数据库文件(包括日志文件)的I/O统计信息 |
| FORMATMESSAGE | 根据sys.message中现有的信息构造一条信息 |
| GETANSINULL | 返回此会话的数据库的默认为空 |
| HOST\_ID | 返回工作站标识号 |
| HOST\_NAME | 返回工作站名 |
| IDENT\_CURRENT | 返回为某个会话和作用域中指定的表或视图生成的最新标识值 |
| IDENT\_INCR | 返回增量值，该值是在带有标识列的表或视图中创建标识列时指定的 |
| IDENT\_SEED | 返回种子值，该值是在带有标识列的表或视图中创建标识列时指定的 |
| @@IDENTITY | 返回最后插入的标识值的系统函数 |
| IDENTITY | 用于在带有INTOtable子句的SELECT语句中将标识列插入到新表中 |
| ISDATE | 确定输入表达式是否为有效时间 |
| ISNULL | 使用指定的替换值替换NULL |
| ISNUMERIC | 确定表达式是否为有效的数值类型 |
| NEWID | 创建uniqueidentifier类型的唯一值 |
| NULLIF | 如果两个指定的表达式等价，则返回空值 |
| PARSENAME | 返回对象名称的指定部分 |
| @@ROWCOUNT | 返回受上一行影响的行数 |
| ROWCOUNT\_BIG | 返回已执行的上一语句影响的行数 |
| SCOPE\_IDENTITY | 返回插入到同一作用域中标识列内的最后一个标识值 |
| SERVERPROPERTY | 返回有关服务器实例的属性信息 |
| SESSIONPROPERTY | 返回会话的SET选项设置 |
| SESSION\_USER | 返回当前数据库中当前上下文的用户名 |
| STATS\_DATE | 返回上次更新指定索引的统计信息的日期 |
| dm\_db\_index\_physical\_stats | 返回指定表或视图的数据和索引的大小和碎片信息 |
| SYSTEM\_USER | 当未指定默认值时，允许将系统为当前登录提供的值插入表中 |
| @@TRANCOUNT | 返回当前连接的活动事务数 |
| UPDATE() | 指示是否对表或视图的指定列进行INSERT或UPDATE尝试 |
| USER\_NAME | 基于指定的标识号返回数据库用户名 |

1. **系统统计函数**

系统统计函数返回系统的统计信息，所有的系统统计函数都具有不确定性。

**表：**系统统计函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| @@CONNECTIONS | 返回自上次启动以来尝试的连接次数，无论连接是成功还是失败 |
| @@CPU\_BUSY | 返回SQL Server自上次启动后的工作时间 |
| fn\_virtualfilestats | 返回数据库文件的I/O统计信息 |
| @@IDLE | 返回SQL Server自上次启动后的空闲时间 |
| @@IO\_BUSY | 返回自从SQL Server最近一次启动以来，SQL Server已经用于执行输入和输出的时间 |
| @@PACKET\_ERRORS | 返回自上次启动SQL Server后，在SQL Server连接上发生的网络数据包错误数 |
| @@PACK\_RECEIVED |  |
| @@PACK\_SENT | 返回SQL Server自上次启动后，写入网络的输出数据包个数 |
| @@TIMETICKS | 返回每个时钟周期的微秒数 |
| @@TOTAL\_ERRORS | 返回SQL Server自上次启动之后所遇到的磁盘写入错误数 |
| @@TOTAL\_READ | 返回SQL Server自上次启动读取磁盘的次数 |
| @@TOTAL\_WRITE | 返回SQL Server自上次启动以来所执行磁盘写入次数 |

1. **文本和图像函数**

文本和图像函数可对文本或图像输入值或列执行操作，并返回有关该值的信息。

**表：**文本和图像函数

|  |  |
| --- | --- |
| 函数名 | 描述 |
| PATINDEX | 对于所有有效的文本和字符数据类型，返回指定表达式中模式第一次出现的起始位置，如果未找到模式则返回0 |
| TEXTPTR | 返回对应于varbinary格式的text、ntext或image列的文本指针值 |
| TEXTVALID | 检查特定文本指针是否有效的text、ntext或image函数 |

# 约束

约束保证数据库中数据的完整性。

**约束分类表**

|  |  |  |
| --- | --- | --- |
| 完整性类型 | 约束分类 | 描述 |
| 域完整性 | DEFAULT | 指定列的默认值 |
| CHECK | 指定允许值 |
| 外键约束 | 指定必须存在的值 |
| NULL | 是否允许为空 |
| 实体完整性 | 主键约束 | 每行的唯一标识 |
| UNIQUE | 不允许有重复key |
| 引用完整性 | FOREIGN KEY | 定义列的值必须与某列的主键值一致 |
| CHECK | 基于另一列的值 |

定义约束的语法结构：

[ CONSTRAINT constraint\_name ]

{

[ NULL | NOT NULL ]

{ PRIMARY KEY | UNIQUE }

[ CLUSTERED | NONCLUSTERED ]

[ WITH FILLFACTOR = fillfactor ]

[ WITH (index\_option [,…,n] ) ]

[ ON { filegroup | “default”} ]

| [ FOREIGN KEY ]

REFERENCES [ schema\_name . ] referenced\_table\_name [ ( ref\_column ) ]

[ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]

[ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]

[ NOT FOR REPLICATION ]

| CHECK [ NOT FOR REPLICATION ]

| DEFAULT constant\_expression [ WITH VALUES ] ( logical\_expression )

}

代码说明：

* CONSTRAINT项，定义约束的开始关键字。
* constraint\_name项，约束的名称。
* NULL | NOT NULL项，指定该列是否可接受空值
* PRIMARY KEY项，通过唯一索引对指定的一列或多列强制进行实体完整性的约束
* UNIQUE项，通过唯一索引为指定的一列或多列提供实体完整性的约束
* CLUSTERED | NONCLUSTERED项，指定为PRIMARY KEY或UNIQUE约束创建聚集或非聚集索引，PRIMARY KEY约束默认为CLUSTERED；UNIQUE约束默认为NONCLUSTERED模式。
* WITH FILLFACTOR = fillfactor项，指定在存储索引数据时对每个索引页的填充程度。
* ON项，指定为约束创建的索引的存储位置。如果指定了filegroup，则将在命名文件组内创建索引；如果指定了“default”或者根本没有指定ON，将在创建表的同一个文件组中创建索引。
* FOREIGN KEY REFERENCES项，为列中数据提供引用完整性的约束
* schema\_name项，FOREIGN KEY约束引用的表所属架构的名称
* referenced\_table\_name项，FOREIGN KEY约束引用的表
* ref\_column项，新FOREIGN KEY约束引用的带括号的列
* ON DELETE项，如果行具有引用关系且该行引用的行已从父表中删除，则对这写行采取的操作，默认值为NO ACTION
* NOT FOR REPLICATION项，可以为FOREIGN KEY约束和CHECK约束指定该参数。如果为约束指定了该子句，则当复制代理执行插入、更新或删除操作时，将不会强制执行此约束。
* CHECK项，该约束通过限制可输入列中的可能值，强制实现域完整性。
* logical\_expression项，用于CHECK约束的逻辑表达式，返回TRUE或FALSE
* DEFAULT项，指定列的默认值
* constant\_expression项，用作默认列值的文字值，NULL或系统函数
* WITH VALUES项，只有在ADD列子句中指定了DEFAULT的情况下，才能使用WITH VALUES。

注意：

每个表只能创建一个PRIMARY KEY约束

### 主键约束

主键约束需要满足列不能有空值且不能有重复的值。

CREATE TABLE syslog

(

myID INT IDENTITY(1,1) NOT NULL,

sysTime DATETIME NOT NULL,

evevtInfo NVARCHAR(100),

memo NVARCHAR(100),

--------------------设置主键约束--------------

CONSTRAINT PK\_syslog PRIMARY KEY NONCLUSTERED

(

myID ASC

)

)

### 外键约束

外键是用于建立和加强两个表数据之间关系的约束，它链接两表的一列或多列。通过将数据表中主键值的列添加到另一个数据表中，可创建两个表中的关系。这个主键列就成第二个表的外键。一般表现为两张数据表中，一张数据表的某一列的所有值，全部取自另外一张表的主键值。外键关系特性是关系型数据库二维表间关系的最重要的组成部分之一。

--------------外键约束-------------------------------

ALTER TABLE [dbo].[new\_phones]

WITH CHECK ADD CONSTRAINT [FK\_new\_phones\_new\_employees]

FOREIGN KEY([employeeid])

REFERENCES [dbo].[new\_employees] ([id\_num])

GO

CREATE TABLE new\_employees

(

[id\_num] INT IDENTITY(1,1) NOT NULL,

[name] VARCHAR(20) NULL,

[sex] CHAR(1) NULL,

[memo] VARCHAR(30) null,

CONSTRAINT PK\_NEW\_EMPLOYEES PRIMARY KEY CLUSTERED

(

id\_num ASC

)

)

CREATE TABLE new\_phones

(

id INT NOT NULL,

employeeid INT NULL,

phone NCHAR(10) NULL,

memo NCHAR(10) NULL,

CONSTRAINT PK\_NEW\_PHONES PRIMARY KEY CLUSTERED

(

id ASC

)

)

### UNIQUE约束

UNIQUE约束是指表中的任何两行都不能有相同的列值。主键也强制实施唯一性，但主键不允许NULL的出现。一般情况下UNIQUE约束用于确保在非主键列中不输入重复的值。一个表可含有多个UNIQUE约束。

CREATE TABLE new\_phone\_with\_unique

(

id INT NOT NULL,

employeeid INT NULL UNIQUE NONCLUSTERED,

phone NCHAR(10),

memo NCHAR(10),

CONSTRAINT PK\_new\_phone\_uni PRIMARY KEY CLUSTERED

(

id ASC

)

)

--代码说明：

--使用UNIQUE NONCLUSTERED为employeeid字段时设为UNIQUE约束

--建立的约束为非聚集的

### CHECK约束

CHECK约束是限制列可接受的值，它可以强制域的完整性。在数据库中一般会经常使用CHECK约束。CHECK约束的使用需要遵循的限制：

* 列可以有多个CHECK约束，且约束条件中可以包含用AND和OR的组合，列的多个CHECK约束按创建的顺序进行验证。
* 列上的CHECK约束只能引用被约束的列，当执行INSERT和DELETE语句时，CHECK约束和规则具有相同的数据验证功能。
* 不能在text、ntext或image数据类型的列上定义CHECK约束。

CREATE TABLE new\_employee\_check

(

[id\_num] INT IDENTITY(1,1) NOT NULL,

[name] VARCHAR(20),

[sex] CHAR(1),

[age] INT,

[memo] VARCHAR,

CONSTRAINT PK\_new\_employee\_check PRIMARY KEY CLUSTERED

(

[id\_num] ASC

)

)

ALTER TABLE new\_employee\_check

ADD CONSTRAINT CK\_age\_2

CHECK (age < 100)

### DEFAULT约束

默认值是一种常用的约束。在数据表中插入列时没有为列指定值，默认值则指定列中使用什么值。默认值可以是计算结果为常数的任何值，如常量、内置函数或数学表达式

------------------------DEFAULT约束----------------------

ALTER TABLE [TestDB].[dbo].[syslog]

ADD CONSTRAINT DF\_syslog\_memo\_df DEFAULT N'无' FOR [memo]

### 禁用约束

禁用约束的使用情形：在大容量的导入\导出数据、临时性的数据维护或者其他操作。

1. 对复制禁用外键约束

打开“外键关系”对话框—>对“强制用于复制”项选择“否“值

1. 对复制禁用CHECK约束

打开“CHECK约束”对话框🡪对“强制用于复制”项选择“否“值

1. 对INSERT和UPDATE语句禁用外键约束

打开“外键关系”对话框🡪选定关系🡪设置INSERT和UPDATE规范

1. 对INSERT和UPDATE语句禁用CHECK约束

打开“CHECK约束”对话框🡪对“强制用于INSERT和UPDATE”项选择“否”值

# 触发器

触发器的描述：

触发器是一种特殊类型的存储过程，其主要是通过事件进行触发而被执行的，而存储过程是通过存储过程的名称进行直接调用的。

触发器的作用：

触发器的作用主要是实现由主键和外键所不能实现的、复杂的参照完整性和数据的一致性。即主要用于表中间的完整性约束，它还可以用于解决高级高级形式的业务规则、复杂行为限制，以及实现定制记录等问题，主要功能：

* 用于强化约束。触发器能够实现比CHECK语句更为复杂的约束。
* 用于跟踪变化。触发器可以检测数据库内的操作，可以实现禁用未经许可的更新和变化。
* 用于级联运行。触发器可以检测数据库内的操作，并自动的级联影响整个数据库得到各项内容。eg：某个表的触发器中包含对另外一个表的数据操作，而该操作导致该表的触发器被触发。
* 用于存储过程的调用。为了影响数据库更新触发器，系统调用一个或多个存储过程，甚至可以通过外部过程的调用而在DBMS系统之外的操作。

注意：触发器的性能通常比较低。原因是系统处理的大部分时间花费在参考其他表的这种处理上。

触发器的分类：

DML触发器（AFTER触发和INSTEAD OF触发器）、DDL触发器和登录触发器

**DML触发器**：是当数据库服务器中发生数据操作语言(DML)事件时要执行的操作。DML触发器用于在数据被修改时，强制执行业务规则，以及扩展SQL Server数据库约束、默认值和规则的完整性检查逻辑。

* AFTER触发器：在INSERT、UPDATE、DELETE语句操作之后执行
* INSTEAD OF触发器：在INSERT、UPDATE、DELETE语句执行时可替代执行

**DDL触发器**：是一种特殊的触发器，在相应数据定义语言(DDL)时触发。一般用于在数据库中执行管理任务，如审核以及规范数据库操作。

登录触发器：是为了响应LOGON事件而激发存储过程。与SQL Server实例建立用户会话时将引发此事件。登录触发器将在登录的身份验证之后、用户会话实际建立之前激发。如果身份验证失败，将不激发登录触发器。

触发器的工作原理：

INSERT触发器：实在将数据插入到表或视图时执行的特殊的存储过程。

DELETE触发器：是将数据从数据表或视图删除时执行的特殊的存储过程。

UPDATE触发器：是在数据表或视图中修改数据时执行的触发器。

INSTEAD OF触发器：是用于代替通常触发器操作的特殊的存储过程。

## 设计DML触发器

AFTER触发器和INSTEAD OF触发器的使用理解：

* AFTER触发器适用于数据表。每个触发操作只包含一个触发器。AFTER触发器允许级联引用，它的执行晚于约束处理、声明性引用操作、创建插入的和删除的表等操作。
* INSTEAD OF触发器适用于数据表和视图。每个触发操作中包含一个触发器。不允许在作为级联引用完整性约束目标的数据表上，使用INSTEAD OF UPDATE和DELETE触发器。INSTEAD OF触发器的执行早于约束处理、晚于创建插入的表和删除的表。

创建触发器所要考虑的问题：

* 使用CREATE TRIGGER语句创建触发器，必须是批处理中的第一个语句，该语句后面所有其他的语句被解释为CREATE TRIGGER语句定义的一部分。
* 创建MDL触发器的权限默认分配给表的所有者，且不能将该权限转给其他用户。
* DML触发器为数据库对象，其名称必须遵循标识符的命名规则。
* 虽然DML触发器可以引用当前数据库以外的对象，但只能在当前数据库中创建DML触发器。
* 不能对临时表或系统表创建DML触发器。
* 不应引用系统表，而应使用信息结构视图。
* 对于含有用DELETE或UPDATE操作定的外键表，不能定义INSTEAD OF和INSTEAD OF UPDATE触发器.
* 使用TRUNCATE TABLE语句不会触发DELETE触发器。
* WRITETEXT语句不会触发INSERT或UPDATE触发器。

## 创建触发器

**CREATE TRIGGER创建DML触发器语法：**

CREATE TRIGGER [ schema\_name . ]trigger\_name

ON { table | view }

[ WITH <dml\_trigger\_option> [,…,n] ]

{ FOR | AFTER | INSTEAD OF }

{ [ INSERT ][,] [ UPDATE ] [,] [ DELETE ] }

[ WITH APPEND ]

[ NOT FOR REPLICATION ]

AS { sql\_statement [;] [,…,n] | EXTERNAL NAME <method specifier > [;] }

<dml\_trigger\_option> ：： =

[ ENCRYPTION ]

[ EXECUTE AS Clause ]

语法说明：

**schema\_name**项，DML触发器所属架构的名称。

**trigger\_name**项，触发器的名称。

**table | view**项，对其执行DML触发器的表或视图，有时称为触发器表或触发器视图。

**WITH ENCRYPTION**项，对CREATE TRIGGER语句的文本进行加密。

**EXECUTE AS**项，指定用于执行该触发器的安全上下文。

**AFTER**项，指定的所有操作都已经成功执行后DML触发器才被激活。

**INSTEAD OF**指定DML触发器是替代SQL语句执行的，其优先级高于触发语句的操作。

**{ [ INSERT ][,] [ UPDATE ] [,] [ DELETE ] }**项，指定数据修改语句，这些语句可在DML触发器对此表或视图进行尝试时激活该触发器。

**NOT FOR REPLICATION**项，指示当复制代理修改涉及触发器的表时，不应该执行触发器。

**sql\_statement**项，指定触发条件和操作。

IF OBJECT\_ID('Table\_Test\_Info','TR') IS NOT NULL

DROP TRIGGER Table\_Test\_Info

GO

CREATE TRIGGER Table\_Test\_Info

ON [dbo].[Table\_Storage] --触发器所在位置

AFTER INSERT,UPDATE

AS RAISERROR('Notity log',16,10) --不执行插入与修改操作

GO

## 修改触发器

使用ALTER TRIGGER语句修改触发器

ALTER TRIGGER [ schema\_name . ]trigger\_name

ON { table | view }

[ WITH <dml\_trigger\_option> [,…,n] ]

{ FOR | AFTER | INSTEAD OF }

{ [ INSERT ][,] [ UPDATE ] [,] [ DELETE ] }

[ NOT FOR REPLICATION ]

AS { sql\_statement [;] [,…,n] | EXTERNAL NAME <method specifier [;]> }

<dml\_trigger\_option> ：： =

[ ENCRYPTION ]

[ EXECUTE AS Clause ]

代码说明与创建时的语法说明一样

代码示例：

IF OBJECT\_ID('TestTrigger','TR') IS NOT NULL

DROP TRIGGER TestTrigger

GO

CREATE TRIGGER TestTrigger --触发器阻止Table\_Storage表进行修改和插入操作

ON [dbo].[Table\_Storage]

WITH ENCRYPTION

AFTER INSERT,UPDATE

AS RAISERROR('Notify Information',16,10)

GO

ALTER TRIGGER TestTrigger --对之前定义的触发器进行修改操作，使之仅限制其插入操作

ON [dbo].[Table\_Storage]

AFTER INSERT

AS RAISERROR('Notify Information',16,10)

GO

## 删除触发器

DROP TRIGGER语法：

DROP TRIGGER TestTrigger --删除触发器

GO

## 禁用/启用触发器

当有大量的数据进行导入或插入时，启用触发器会降低效率。

**禁用触发器**

DISABLE TRIGGER语法：

DISABLE TRIGGER { [ schema\_name ] trigeer\_name [,…,n] | ALL }

ON { object\_name | DATABASE | ALL SERVER } [;]

代码说明：

schema\_name项，触发器所属架构的名称。

trigeer\_name项，要禁用的触发器的名称。

ALL项，指示禁用在ON子句作用域中定义的所有触发器。

object\_name项，对其创建要执行的DML触发器trigeer\_name的表或视图的名称。

DATABASE项，对于DDL触发器,所有创建或修改的trigeer\_name将在数据库操作域内执行

ALL SERVER项，对于DDL触发器，所创建或修改的trigeer\_name将在服务器作用域内执行

代码示例：

DISABLE TRIGGER TestTrigger --禁用指定名称的触发器

ON [dbo].[Table\_Storage] --触发器所应用的表

GO

**启用触发器**

ENABLE TRIGGER语法：

ENABLE TRIGGER { [ schema\_name ] trigeer\_name [,…,n] | ALL }

ON { object\_name | DATABASE | ALL SERVER } [;]

代码说明：

schema\_name项，触发器所属架构的名称。

trigeer\_name项，要启用的触发器的名称。

ALL项，指示启用在ON子句作用域中定义的所有触发器。

object\_name项，对其创建要执行的DML触发器trigeer\_name的表或视图的名称。

DATABASE项，对于DDL触发器,所有创建或修改的trigeer\_name将在数据库操作域内执行

ALL SERVER项，对于DDL触发器，所创建或修改的trigeer\_name将在服务器作用域内执行

## 设计DDL触发器

DDL触发器用于响应各种数据定义语言，主要包括CREATE、ALTER、DROP语句。

创建DDL触发器语法：

CREATE TRIGGER trigger\_name

ON { ALL SERVER | DATABASE }

[ WITH <dml\_trigger\_option> [,…,n] ]

{ FOR | AFTER }

{ event\_type | even\_group } [,…,n]

AS { sql\_statement [;] [,…,n] | EXTERNAL NAME <method specifier > [;] }

<dml\_trigger\_option> ：： =

[ ENCRYPTION ]

[ EXECUTE AS Clause ]

代码说明：

DATABASE项，将DDL触发器的作用域用于当前数据库

ALL SERVER项，将DLL触发器的作用域用于当前服务器

event\_type项，执行之后将导致激活DDL触发器的T-SQL语言事件的名称

even\_group项，预定义的T-SQL语言事件分组的名称

# 事务

事务可以看作单个的处理单元。如果某一事务执行，则在该事务中所有的操作均会执行，并记录在数据库中，成为持久的组成部分。如果遇到问题，则必须全部取消或回滚，所有处理均被清除。

SQL Server数据库包括自动提交事务、显示事务、隐式事务和批处理级事务四种形式。

**自动提交事务**：是指每条单独的语句都是一个事务，执行语句即提交事务；

**显示事务**：是指以BEGIN TRANSACTION语句开始，以COMMIT或ROLLBACK语句结束事务

**隐式事务**：在前一个事务完成时，新事务隐式启动，但每个事务仍以COMMIT或ROLLBACK语句显示完成。

**批处理级事务**：只能应用于多个活动结果集。

事务的特点：

* **原子性**：每个事务是一个工作单元，它不能被分割成更小的部分，这个属性表示在事务中定义的一切数据更改要么都完成，要么都不完成。
* **一致性**：一个事务不能违背定义在数据库中任何完整性检查。为了维护一致性，所有的规则、约束、检查和触发都会应用在事务中。由于所有的数据更改在事务期间进行，这些数据在事务开始和事务结束前会被确保一致。
* **隔离性**：事务必须与其他事务进行的数据更改相隔离。着表示没有其他操作可以改变中间态的数据。事务必须等待其他事务的更改被提交，或者只能查看到处于上一个提交状态的数据。
* **持久性**：在一个事务完成，并且应用程序已经提示这个事务已经成功完成后，无论发生任何系统错误，这些更改的数据将永久存在数据库。

## 启动事务

BEGIN TRANSACTION表示事务的起点。

BEGIN TRANSACTION语句的语法：

BEGIN { TRAN | TRANSACTION }

[ { transaction\_name | @tran\_name\_variable }

[ WITH MARY [‘description’] ]

]

[;]

语法说明：

transaction\_name项，指分配给事务的名称。

@tran\_name\_variable项，是用户定的、含有有效事务名称的变量的名称。

[ WITH MARY [‘description’]：指定在日志中标记事务。

注意：

BEGIN TRANSACTION使TRANCOUNT按1递增

代码示例：

BEGIN TRANSACTION LogDelete0 --指定事务的开始语句

WITH MARK 'Deleting Database Log Info' --为事务进行标记

GO

USE TestDB

GO

DELETE FROM [TestDB].[dbo].[Table\_Test]

GO

COMMIT TRANSACTION MyTrans --指定事务的结束

## 保存事务

SAVE TRANSACTION语法： 设置保存点

SAVE { TRAN | TRANSACTION } { savepoint\_name | @savepoint\_variable }

代码说明：

savepoint\_name项，分配给保存点的名称。

@savepoint\_variable项，包含有效保存点名称的用户定义的名称的变量。

代码示例：

USE TestDB

GO

CREATE TABLE DEMO\_EMPLOYEE --建立测试环境

(

[id\_num] INT IDENTITY(1,1) NOT NULL,

[name] VARCHAR(20) NULL,

[sex] CHAR(1) NULL,

[memo] VARCHAR(30) NULL,

CONSTRAINT PK\_DEMO\_EMPLOYEE PRIMARY KEY CLUSTERED

(

[id\_num] ASC

)

)

USE TestDB

GO

BEGIN TRANSACTION --启动事务

INSERT INTO DEMO\_EMPLOYEE ([name],[sex],[memo]) --执行插入操作

VALUES('LISI','1','This is test!')

SAVE TRANSACTION A

INSERT INTO DEMO\_EMPLOYEE ([name],[sex],[memo])

VALUES('ZHANGSAN','1','This is test')

SAVE TRANSACTION B

INSERT INTO DEMO\_EMPLOYEE ([name],[sex],[memo])

VALUES('WANFMEI','1','This is test!')

ROLLBACK TRANSACTION B --回退至第二条数据插入之后

INSERT INTO DEMO\_EMPLOYEE ([name],[sex],[memo])

VALUES('ZHANGTAO','1','This is test!')

ROLLBACK TRANSACTION A --回退至第一条数据插入之后

COMMIT TRANSACTION

SELECT \* FROM DEMO\_EMPLOYEE

GO

## 提交事务

在提交事务时，如果@@TRANCOUNT函数的值为1，COMMIT TRANSACTION使得自从事务开始以来，所执行的所有数据提交到数据库全部生效，同时释放事务所占用的资源，将@@TRANCOUNT函数值减到0.如果@@TRANCOUNT大于1，则COMMIT TRANSACTION使@@TRANCOUNT按1递减，并且事务将保持活动状态。

COMMIT TRANSACTION语法：

COMMIT { TRAN | TRANSACTION } { transaction\_name | @tran\_name\_variable }

## 回滚事务

用户可以使用ROLLBACK TRANSACTION清除自事务的起点或某个保存点所做的所有数据的操作，回滚后，系统将释放由事务控制的资源。

ROLLBACK TRANSACTION语法：

ROLLBACK { TRAN | TRANSACTION }

{ transaction\_name | @tran\_name\_variable |

savepoint\_name | }

[;]

语法说明：

transaction\_name项，是为BEGIN TRANSACTION上的事务分配的名称。

@tran\_name\_variable项，是用户定义的、包含有效事务名称的变量的名称。

savepoint\_name项，是SAVE TRANSACTION语句中的savepoint\_name。

@savepoint\_variable项，是用户定义的、包含有效保存点名称的变量的名称。

注意：

默认情况下，不带savepoint\_name和transaction\_name的ROLLBACK TRANSACTION回滚到事务的起点。嵌套事务时，该语句将所有内层事务回滚到最外面的BEGIN TRANSACTION语句。无论哪种情况，ROLLBACK TRANSACTION都将@@TRANCOUNT系统函数减小为0. ROLLBACK TRANSACTION savepoint\_name不减小@@TRANCOUNT。

代码示例：

USE TestDB

GO

IF EXISTS(SELECT \* FROM SYSOBJECTS WHERE NAME = 'Account') --创建示例环境

DROP TABLE Account

CREATE TABLE Account --创建示例表

(

customerName NVARCHAR(10),

currentMoney MONEY

)

GO

ALTER TABLE Account --为该表添加CHECK约束

ADD CONSTRAINT CK\_currentMoney CHECK (currentMoney >= 1)

GO

INSERT INTO Account VALUES(N'张三','1000') --使用INSERT INTO向表中添加数据

INSERT INTO Account VALUES(N'李四','1')

SELECT \* FROM Account --第一次SELECT，，显示转账前的表值项

BEGIN TRANSACTION --事务的起点

DECLARE @errorSum INT --创建记录错误号的变量

SET @errorSum = 0 --设置归零

UPDATE Account SET currentMoney -= 1000 --转出

WHERE customerName = N'张三'

SET @errorSum += @@ERROR --若发生错误则记录该错误

UPDATE Account SET currentMoney += 1000 --转入

WHERE customerName = N'李四'

SET @errorSum += @@ERROR --若发生错误则记录该错误

SELECT 错误 = @errorSum

PRINT N'查看转账事务过程中的余额'

SELECT \* FROM Account --第二次SELECT，，显示转账事务中的余额信息

IF (@errorSum <> 0) --条件判断 1、为真回滚 2、否则执行确认

BEGIN

PRINT N'交易失败，回滚事务'

ROLLBACK TRANSACTION

END

ELSE

BEGIN

PRINT N'交易成功，提交事务，写入硬盘，永久保存'

COMMIT TRANSACTION

END

PRINT N'查看转账后余额'

SELECT \* FROM Account --第三次SELECT，，显示执行转账事务之后的余额信息

GO

## 自动提交事务

实际上，SQL Server数据库将一切数据处理都以事务的形式运行如果没有显式的定义事务，SQL Server会自动为该处理定义事务。由SQL Server定义的事务被称为自动定义事务。

代码示例：

USE TestDB

GO

CREATE TABLE Trans

(

id INT NOT NULL PRIMARY KEY,

coll NVARCHAR(20) NOT NULL,

memo NVARCHAR(20) NULL

)

GO

INSERT INTO Trans(id,coll,memo)

VALUES(1,'TEST1','TEST1')

INSERT INTO Trans(id,coll,memo)

VALUES(2,NULL,'TEST2')

INSERT INTO Trans(id,coll,memo)

VALUES(3,'TEST3','TEST3')

GO

SELECT \* FROM Trans

GO

第二次的插入操作将被作废。

## 显示事务

显式事务就是显式地在代码中定义事务的开始和结束的事务。

代码示例：

USE TestDB

GO

CREATE TABLE Trans\_1 --创建测试环境

(

id INT NOT NULL PRIMARY KEY,

coll NVARCHAR(20) NOT NULL,

memo NVARCHAR(20) NULL

)

GO

BEGIN TRY --捕捉事务错误信息

BEGIN TRAN --事务起点

INSERT INTO Trans(id,coll,memo)

VALUES(1,'TEST1','TEST1')

INSERT INTO Trans(id,coll,memo)

VALUES(2,NULL,'TEST2')

INSERT INTO Trans(id,coll,memo)

VALUES(3,'TEST3','TEST3')

COMMIT TRAN --事务结束

END TRY

BEGIN CATCH --捕捉错误后回滚事务

SELECT 错误 = @@ERROR

ROLLBACK TRAN --回滚事务

END CATCH

SELECT \* FROM Trans\_1

GO

## 隐式事务

隐式事务是SQL Server的另一种事务处理模式。在使用这种模式时，SQL Server在没有事务存在的条件下会开始一个事务，但不会像在自动模式中那样自动执行COMMIT或ROLLBACK语句，事务必须显示结束。即在发出COMMIT或ROLLBACK语句之前，该事务将一直保持有效。（十分危险一定要显示结束）

开启隐式事务：

SET IMPLICIT\_TRANSACTIONS ON

关闭隐式事务：

SET IMPLICIT\_TRANSACTIONS OFF

**表：**隐式事务模式打开后自动启动事务的语句

|  |  |  |
| --- | --- | --- |
| ALTER TABLE | CREATE | DROP |
| DELECT | FEICH | GRANT |
| INSERT | OPEN | REVOKE |
| SELECT | TRUNCATE TABLE | UPDATE |

## 事务嵌套

**嵌套事务**

事务可以进行嵌套，但是只有在显示事务模式下，事务才可以嵌套。主要是为了支付存储过程业务处理的需要。

代码示例：

SET QUOTED\_IDENTIFIER OFF --禁止标识符内的引号

GO

SET NOCOUNT OFF --返回受影响的行数

GO

USE TestDB --跳转到指定的数据库中

GO

CREATE TABLE Trans\_3 --创建测试的环境表

(

Col1 INT PRIMARY KEY,

Col2 NVARCHAR(50) NOT NULL

)

GO

CREATE PROCEDURE Trans3Proc @pri INT,@Col2 NVARCHAR(50) --创建一个存储过程

AS

BEGIN TRANSACTION Pro3 --存储过程中定义的嵌套事务

INSERT INTO Trans\_3 VALUES(@pri,@Col2)

INSERT INTO Trans\_3 VALUES(@pri+1,@Col2)

COMMIT TRANSACTION Pro3

GO

BEGIN TRAN outerProc; --开始第一个事务，执行存储过程

EXEC Trans3Proc 1,'TEST1';

ROLLBACK TRAN outerProc; --回滚外层事务，同时回滚Trans3Proc的嵌套事务

GO

EXEC Trans3Proc 2,'TEST2';

GO

SELECT \* FROM Trans\_3;

GO

注意：整个嵌套事务是提交还是回滚主要取决于外部事务。若外外部事务回滚则整个事务将进行回滚操作，若外部事务提交也将提交内部事务对数据的操作。

**@@TRANCOUNT函数**

提交嵌套事务时可以使用@@TRANCOUNT函数查看嵌套的层级。

BEGIN TRAN使@@TRANCOUNT增加1；

COMMIT TRAN或COMMIT WORK使@@TRANCOUNT减去1；

ROLLBACK WORK或ROLLBACK TRAN将回滚所有的嵌套事务，并使@@TRANCOUNT减小到0.

## 管理并发数据访问

**并发的影响**

并发情况下，修改数据的用户会影响同时读取或修改相同数据的其他用户，这些用户可以并发访问数据。如果数据存储系统没有并发访问机制，将会出现许多问题：

* 更新丢失
* 出现未提交的依赖性（脏读）
* 出现不一致的分析（不可重复读）
* 幻读

**并发控制类型：**

悲观并发控制

乐观并发控制

事务的隔离级别

**表：**事务的隔离级别

|  |  |  |  |
| --- | --- | --- | --- |
| 隔离级别 | 脏读 | 不可重复读取 | 幻读 |
| 未提交读 | 是 | 是 | 是 |
| 已提交读 | 否 | 是 | 是 |
| 可重复读 | 5426 | 否 | 是 |
| 快 | 否 | 否 | 否 |
| 可序列化 | 否 | 否 | 否 |

# 锁定

锁定是SQL Server数据库引擎用来同步多个用户时对同一个数据块访问的一种机制。每个事务对所依赖的资源(行、页或表)可以请求不同类型的锁。

锁可以阻止其他事务造成的负面影响，当事务不再依赖锁定的资源时，可以释放锁。

**表：**数据库引擎可以锁定的资源

|  |  |
| --- | --- |
| 资源 | 说明 |
| RID | 用于锁定堆中的单个行的行标识符 |
| KEY | 索引中用于保护可序列化事务中的键盘范围的行锁 |
| PAGE | 数据库中的8KB，例如数据页或索引页 |
| EXTENT | 一组连续的八页，如数据页或索引页 |
| HOBT | 堆或B树，保护索引或没有聚集索引的表中数据页堆的锁 |
| TABLE | 包括所有数据和索引的整个表 |
| FILE | 数据库文件 |
| APPLICATION | 应用程序专用的资源 |
| METADATA | 元数据锁 |
| ALLOCATION\_UNIT | 分配单元 |
| DATABASE | 整个数据库 |

模式锁

SQL Server数据库引擎使用不同的锁模式锁定资源，这些锁模式确定了并发事务访问资源的方式。

**表：**锁模式

|  |  |
| --- | --- |
| 锁模式 | 说明 |
| 共享(S) | 用于不更新或不更改数据的读取操作，如SELECT语句 |
| 更新(U) | 用于防止当多个会话在读取、锁定、以及随后可能进行的资源更新时发生常见形式的死锁 |
| 排他(X) | 用于数据修改操作，如INSERT、UPDATE等，确保不会同时对同一个资源进行多重修改 |
| 意向 | 用于建立锁的层次结构，其类型有：意向共享(IS)、意向排他(IX)、以及意向排他共享(SIX) |
| 架构 | 在执行依赖于表架构的操作时使用，其类型有：架构修改(Sch-M)和架构稳定性(Sch-S) |
| 大容量更新(BU) | 在向表进行大容量数据复制且指定了TABLOCK提示时，使用该锁 |
| 键范围 | 当使用可序列化事务隔离级别时，保护查询读取的行的范围，确保再次运行查询时，其他事务无法插入符合可序列化事务的查询的行 |