[SQL Server 2005 分区表实践——建立分区表（partition table)](http://www.cnblogs.com/hellen-li/archive/2009/11/10/1599587.html)

【鹏城万里】 发表于 [**http://www.sqlstudy.com/**](http://www.sqlstudy.com/)

**SQL Server 2005 分区表实践——建立分区表（partition table）**

问题：有一个订单表 Orders，要转换成分区表，以订单日期 OrderDate 为分区列，目前含有订单日期为 1996-07-04 ~ 1998-05-06 的数据。可以在 SQL Server 2000 Northwind 数据库中找到 Orders 表，下面是简化了的表结构：

create table dbo.Orders

(

OrderID int not null

,CustomerID varchar(10) not null

,EmployeeID int not null

,OrderDate datetime not null

,constraint PK\_Orders primary key noclustered (OrderID, CustomerID)

)

go

create clustered index IXC\_Orders\_OrderDate on dbo.Orders(OrderDate)

go

**1. 创建分区函数（partition function）**

在创建分区函数的时候，我一般这样命名分区函数：'PF' + Table Name + 分区字段名 + 'Range'，例如：'PF\_Orders\_OrderDateRange'，如果字段名较长的话，则可以省略去，可以这样命名： 'PF\_Orders\_DateRange'

create partition function PF\_Orders\_OrderDateRange(datetime)

as

range right for values (

'1997-01-01',

'1998-01-01',

'1999-01-01'

)

go

分区函数 'PF\_Orders\_OrderDateRange' 有 3 个边界值（'1997-01-01', '1998-01-01', '1999-01-01'），这三个边界值（boundary value）组成了 4 个分区（partition），并且 “range right” 表明边界值属于右边的分区。下面使用存储过程 dbo.sp\_show\_partition\_range 来查看分区范围：

exec dbo.sp\_show\_partition\_range @partition\_function = 'PF\_Orders\_OrderDateRange'

partition\_function partition minval value maxval

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

PF\_Orders\_OrderDateRange 1 NULL <= val < 1997-01-01 00:00:00.000

PF\_Orders\_OrderDateRange 2 1997-01-01 00:00:00.000 <= val < 1998-01-01 00:00:00.000

PF\_Orders\_OrderDateRange 3 1998-01-01 00:00:00.000 <= val < 1999-01-01 00:00:00.000

PF\_Orders\_OrderDateRange 4 1999-01-01 00:00:00.000 <= val < NULL

每个分区的最大值和最小值，一清二楚。获得 [dbo.sp\_show\_partition\_range](http://www.sqlstudy.com/sql_article.php?id=2008070801" \o "dbo.sp_show_partition_range" \t "_blank) 代码。

**2. 创建分区方案（partition scheme）**

分区方案定义了，分区表或者分区索引的每个分区的数据存放在哪个文件组上。试想，如果没有分区方案的话，如何实现把分区表中不同的分区存放在不同的文件组上？我们可以看下创建普通数据表的语法：

create table [table name] ... on [filegroup]

显然，普通表整个表的数据，只能存放在同一个文件组上。为了实现分区机制，才引入了分区方案这个概念。每个分区表只属于一个方案（scheme），因此分区方案可以这样命名：'PS' + Table Name

create partition scheme PS\_Orders

as

partition PF\_Orders\_OrderDateRange

to ([primary], [primary], [primary], [primary])

go

分区方案 PS\_Orders 规定了，把分区表的 4 个分区分别存放在主文件组 primary 上。分区方案中指定的文件组数目，不能少于分区函数中划定的分区数目；但可以多于分区函数中划定的分区数目。多出的第 1 个文件组用来指定当分区表增加分区时，下一个分区所使用的文件组；多出的其他文件组将被忽略。

下面的例子，分区方案指定了 5 个文件组（多出了 1 个文件组）。

drop partition scheme PS\_Orders

go

create partition scheme PS\_Orders

as

partition PF\_Orders\_OrderDateRange

to ([primary], [primary],[primary], [primary],[primary])

go

Partition scheme 'PS\_Orders' has been created successfully.

'PRIMARY' is marked as the next used filegroup in partition scheme 'PS\_Orders'.

下面的例子，分区方案指定了 7 个文件组（多出了 3 个文件组）。

drop partition scheme PS\_Orders

go

create partition scheme PS\_Orders

as

partition PF\_Orders\_OrderDateRange

to ([primary], [primary],[primary], [primary],[primary],[primary],[primary])

go

Partition scheme 'PS\_Orders' has been created successfully.

'PRIMARY' is marked as the next used filegroup in partition scheme 'PS\_Orders'.

2 filegroup specified after the next used filegroup are ignored.

如果分区表所有的分区都将分配在同一个文件组 [primary] 上，那么可以使用下面更简洁的方法：

drop partition scheme PS\_Orders

go

create partition scheme PS\_Orders

as

partition PF\_Orders\_OrderDateRange

all to ([primary])

go

**3. 创建分区表（partition table）**

创建了分区函数和分区方案，准备工作做完了，现在终于可以开始创建分区表了。创建分区表和普通表的语法大致相同，不同之处：普通表需要指定所存放的文件组，分区表需要指定分区方案。

create table dbo.Orders

(

OrderID int not null

,CustomerID varchar(10) not null

,EmployeeID int not null

,OrderDate datetime not null

)

on PS\_Orders(OrderDate)

go

根据订单表 Orders 查询时经常使用 OrderDate 范围条件来查询的特点，我们最好在 Orders.OrderDate 列上建立聚集索引（clustered index）。为了便于进行分区切换（partition swtich），大多数情况下，建议在分区表上建立分区索引。下面建立聚集分区索引：

create clustered index IXC\_Orders\_OrderDate on dbo.Orders(OrderDate)

on PS\_Orders(OrderDate)

如果没有指定 “on PS\_Orders(OrderDate)”，默认建立的聚集索引和分区表的分区方案相同。

另外 Orders 分区表需要在（OrderID, CustomerID）上建立主键。我们知道主键实际上是个唯一索引，但分区表在建立唯一索引（分区索引）的时候，分区列必须是唯一索引的一部分。为什么要这样子呢？因为 SQL Server 不但要保证索引在各个分区是唯一的，还要保证在整个表中是唯一的。

alter table dbo.Orders add constraint PK\_Orders primary key (OrderID, CustomerID, OrderDate)

如果不指定 OrderDate 的话，则会出现错误：1908

Msg 1908, Level 16, State 1, Line 1

Column 'OrderDate' is partitioning column of the index 'PK\_Orders'.

Partition columns for a unique index must be a subset of the index key.

Msg 1750, Level 16, State 0, Line 1

Could not create constraint. See previous errors.

查看分区表 Orders 上的索引：

exec sp\_helpindex 'dbo.Orders'

index\_name index\_description index\_keys

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

IXC\_Orders\_OrderDate clustered located on PS\_Orders OrderDate

PK\_Orders nonclustered, unique, primary key located on PS\_Orders OrderID, CustomerID, OrderDate

**4. 向分区表中填充数据**

insert into dbo.Orders

select OrderID, CustomerID, EmployeeID, OrderDate

from dbo.Orders\_From\_SQL2000\_Northwind

数据表 dbo.Orders\_From\_SQL2000\_Northwind，是从 SQL Server 2000 中 Northwind.Orders 迁移过来的。

**5. 查看分区表各分区数据情况（数据行数，最大最小 OrderDate 值）**

select partition = $partition.PF\_Orders\_OrderDateRange(OrderDate)

,rows = count(\*)

,minval = min(OrderDate)

,maxval = max(OrderDate)

from dbo.Orders

group by $partition.PF\_Orders\_OrderDateRange(OrderDate)

order by partition

partition rows minval maxval

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

1 152 1996-07-04 00:00:00.000 1996-12-31 00:00:00.000

2 408 1997-01-01 00:00:00.000 1997-12-31 00:00:00.000

3 270 1998-01-01 00:00:00.000 1998-05-06 00:00:00.000

(3 row(s) affected)

从以上结果集中可以看出：分区表 Orders 的 3 个分区中已经填入了数据。这里要注意 $partition.partition\_function\_name(expression) 这个函数，或许像我一样：一辈子没见过这样的函数：）

本文由 **www.sqlstudy.com** 原创，版权所有，转载请注明作者和出处！

本文链接：<http://www.sqlstudy.com/sql_article.php?id=2008071101>

SQL Server数据库表分区操作过程由三个步骤组成：

　　1. 创建分区函数

　　2. 创建分区架构

　　3. 对表进行分区

　　下面将对每个步骤进行详细介绍。

　　步骤一：创建一个分区函数

　　此分区函数用于定义你希望SQL Server如何对数据进行分区的参数值([u]how[/u])。这个操作并不涉及任何表格，只是单纯的定义了一项技术来分割数据。

　　我们可以通过指定每个分区的边界条件来定义分区。例如，假定我们有一份Customers表，其中包含了关于所有客户的信息，以一一对应的客户编号(从1到1,000,000)来区分。我们将通过以下的分区函数把这个表分为四个大小相同的分区：

CREATEPARTITIONFUNCTIONcustomer\_partfunc(int)  
　　ASRANGERIGHT  
　　FORVALUES(250000,500000,750000)

　　这些边界值定义了四个分区。第一个分区包括所有值小于250,000的数据，第二个分区包括值在250,000到49,999之间的数据。第三个分区包括值在500,000到7499,999之间的数据。所有值大于或等于750,000的数据被归入第四个分区。

　　请注意，这里调用的"RANGE RIGHT"语句表明每个分区边界值是右界。类似的，如果使用"RANGE LEFT"语句，则上述第一个分区应该包括所有值小于或等于250,000的数据，第二个分区的数据值在250,001到500,000之间，以此类推。

　　步骤二：创建一个分区架构

　　一旦给出描述如何分割数据的分区函数，接着就要创建一个分区架构，用来定义分区位置([u]where[/u])。创建过程非常直截了当，只要将分区连接到指定的文件组就行了。例如，如果有四个文件组，组名从"fg1"到"fg4"，那么以下的分区架构就能达到想要的效果：

CREATEPARTITIONSCHEMEcustomer\_partscheme  
　　ASPARTITIONcustomer\_partfunc  
　　TO(fg1,fg2,fg3,fg4)

　　注意，这里将一个分区函数连接到了该分区架构，但并没有将分区架构连接到任何数据表。这就是可复用性起作用的地方了。无论有多少数据库表，我们都可以使用该分区架构(或仅仅是分区函数)。

　　步骤三：对一个表进行分区

　　定义好一个分区架构后，就可以着手创建一个分区表了。这是整个分区操作过程中最简单的一个步骤。只需要在表创建指令中添加一个"ON"语句，用来指定分区架构以及应用该架构的表列。因为分区架构已经识别了分区函数，所以不需要再指定分区函数了。

　　例如，使用以上的分区架构创建一个客户表，可以调用以下的Transact-SQL指令：

CREATETABLEcustomers(FirstNamenvarchar(40),LastNamenvarchar(40),CustomerNumberint)  
　　ONcustomer\_partscheme(CustomerNumber)