How to partition a non-partitioned table in SQL Server 2008

<http://www.kodyaz.com/articles/how-to-partition-table-non-partitioned-table-sql-server-2008.aspx>

"Can we partition existing table in SQL Server ?", or  
"How can we add partitioning to a non partitioned table ?" are frequently asked questions about sql partitioning and partitioned tables in MS SQL Server among t-sql developers and SQL Server professionals.

Especially partitioning non-partitioned table may become a difficult task to partition an already existing table if it contains a lot of data and has many constraints and relations with other sql tables in the database.

In this sql tutorial, I'll show sql developers a SQL Server 2008 partition file method.  
Existing tables can be partitioned by creating a **Clustered Index ON PartitionScheme** sql syntax.

CREATE CLUSTERED INDEX ClusteredIndexName  
  ON Production.Product(FieldName) ON PartitionSchemeName (FieldName)

After the clustered index is created on sql table for related field, our sql table is being partitioned by **partition scheme** and **partition function**.

If cluster index is not required, sql administrators can drop clustered index using DROP INDEX syntax.  
Dropping index will not delete partition and will not affect the partitioning of sql table.  
Later, SQL Server database administrators and sql developers can create cluster index on a different field which will not affect partitionning of the sql table.

*Backup SQL Database using SQL Backup Script*

I'm going to try to give an sql example with t-sql code required for partitioning a non-partitioned table using AdventureWorks sample database tables.

On my development computer, I have Microsoft SQL Server 2008 R2 installed.

I have also download SQL Server 2008 R2 sample databases.

Let's start taking backup of AdventureWorks2008R2 sql sample database using the below sql backup script.

BACKUP DATABASE AdventureWorks2008R2 TO DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL10\_50.MSSQLSERVER\MSSQL\Backup\AdventureWorks2008R2.bak'

*Create a New SQL Server Database*

After sql backup database AdventureWorks2008R2, now create a new database.

CREATE DATABASE SQLPartitionedDB

*Restore SQL Backup over a SQL Server Database using T-SQL*

Following creation of new sql database, restore AdventureWorks2008R2.bak database backup file over this new sql database.

RESTORE DATABASE [SQLPartitionedDB] FROM

DISK = N'C:\Program Files\Microsoft SQL Server\MSSQL10\_50.MSSQLSERVER\MSSQL\Backup\AdventureWorks2008R2.bak' WITH FILE = 1,MOVE N'AdventureWorks2008R2\_Data' TO N'C:\Program Files\Microsoft SQL Server\MSSQL10\_50.MSSQLSERVER\MSSQL\DATA\SQLPartitionedDB\_Data.mdf' MOVE N'AdventureWorks2008R2\_Log' TO N'C:\Program Files\Microsoft SQL Server\MSSQL10\_50.MSSQLSERVER\MSSQL\DATA\SQLPartitionedDB\_Log.ldf',MOVE N'FileStreamDocuments2008R2' TO N'C:\Program Files\Microsoft SQL Server\MSSQL10\_50.MSSQLSERVER\MSSQL\DATA\SQLPartitionedDB', NOUNLOAD, REPLACE, STATS = 10

GO

Now we have a sample database that we can work on it.

How to Create FileGroups and Database Files on SQL Server using T-SQL

Let's assume we want to create partitioning on Production.Product database table.

When I checked the database table I see that I can create a partition on "name" column which is defined in nvarchar sql type with 50 characters long.

After sql administrator has defined which column he should create partition based on, sql administrator is ready to build CREATE PARTITION FUNCTION code.

Using partition function database administrators can continue partitioning process with CREATE PARTITION SCHEME statement.

But at this point database filegroups and database files must already have been defined and selected for partitioning.

You can check existing database files and file groups using following t-sql select scripts.

SELECT \* FROM sys.filegroups

SELECT \* FROM sys.database\_files

Since in our sample database file, we do not have additional data files and file groups, let's continue partition example by creating filegroups and assigning database files to this filegroups.

USE [master]

GO

ALTER DATABASE [SQLPartitionedDB] ADD FILEGROUP [filegroup1]

ALTER DATABASE [SQLPartitionedDB]ADD FILE (NAME = N'File1', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL10\_50.MSSQLSERVER\MSSQL\DATA\SQLFile1.ndf', SIZE = 3072KB , FILEGROWTH = 1024KB ) TO FILEGROUP [filegroup1]

ALTER DATABASE [SQLPartitionedDB] ADD FILEGROUP [filegroup2]

ALTER DATABASE [SQLPartitionedDB] ADD FILE (NAME = N'File2',FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL10\_50.MSSQLSERVER\MSSQL\DATA\SQLFile2.ndf',SIZE = 3072KB , FILEGROWTH = 1024KB ) TO FILEGROUP [filegroup2]

ALTER DATABASE [SQLPartitionedDB] ADD FILEGROUP [filegroup3]

ALTER DATABASE [SQLPartitionedDB] ADD FILE (NAME = N'File3',FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL10\_50.MSSQLSERVER\MSSQL\DATA\SQLFile3.ndf',SIZE = 3072KB , FILEGROWTH = 1024KB ) TO FILEGROUP [filegroup3]

ALTER DATABASE [SQLPartitionedDB] ADD FILEGROUP [filegroup4]

ALTER DATABASE [SQLPartitionedDB] ADD FILE (NAME = N'File4',FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL10\_50.MSSQLSERVER\MSSQL\DATA\SQLFile4.ndf',SIZE = 3072KB , FILEGROWTH = 1024KB ) TO FILEGROUP [filegroup4]

*How to Create Partition Function on SQL Server 2008*

We are now ready to create our sql create partition function command.

Since we create 4 filegroups, we can define 3 points to divide sql data into 4 partitions.

CREATE PARTITION FUNCTION pf\_PartitionFunction ( nvarchar(50) )

AS RANGE RIGHT FOR VALUES ('E', 'N', 'S');

*How to Create Partition Scheme on SQL Server 2008*

Using the partition function pf\_PartitionFunction, let's create partition scheme using below t-sql sample code by defining the approtiate filegroup names as argument to the sql command.

CREATE PARTITION SCHEME ps\_PartitionScheme

AS PARTITION pf\_PartitionFunction

TO (filegroup1, filegroup2, filegroup3, filegroup4)

*Check Existing Clustered Index on Partition Table Candidate*

Before create clustered index on partition scheme ps\_PartitionScheme in order to alter non-partitioned table to partitioned table, let's check if there is already a clustered index on target sql database table.

SELECT \* FROM sys.indexes

WHERE

OBJECT\_NAME(object\_id) = 'Product'

AND type = 1

I see that there is already a clustered index PK\_Product\_ProductID on sql table Product.

When I try to drop clustered index PK\_Product\_ProductID on Product table using DROP INDEX command,

DROP INDEX Production.Product.PK\_Product\_ProductID -- or

DROP INDEX PK\_Product\_ProductID ON Production.Product

I get the following exception from sql engine.

Msg 3723, Level 16, State 4, Line 1

An explicit DROP INDEX is not allowed on index 'Production.Product.PK\_Product\_ProductID'. It is being used for PRIMARY KEY constraint enforcement.

*Drop Foreign Key Constraint Referencing To Partition Table Candidate*

Let's check foreign keys referencing our example table for partition.

SELECT \* FROM sys.foreign\_keys

WHERE

referenced\_object\_id = OBJECT\_ID('Production.Product')

Now drop those foreign keys.

Note that it is a better decision to create all those foreign key constraints defined on different sql tables referencing our partition table later after partitioning is completed.

So please take create scripts of those foreign keys for later use.

DECLARE @SQL nvarchar(max) = ''

SELECT @SQL = @SQL + '

ALTER TABLE ' +

SCHEMA\_NAME(SCHEMA\_ID) + '.' +

OBJECT\_NAME(parent\_object\_id) +

' DROP CONSTRAINT ' +

name + ';'

FROM sys.foreign\_keys

WHERE

referenced\_object\_id = OBJECT\_ID('Production.Product')

PRINT @SQL

EXEC SP\_EXECUTESQL @SQL

Here is the DROP Constraint sql statements that the above dynamic sql script generates and executes.

ALTER TABLE Production.BillOfMaterials DROP CONSTRAINT FK\_BillOfMaterials\_Product\_ProductAssemblyID;

ALTER TABLE Production.BillOfMaterials DROP CONSTRAINT FK\_BillOfMaterials\_Product\_ComponentID;

ALTER TABLE Production.ProductCostHistory DROP CONSTRAINT FK\_ProductCostHistory\_Product\_ProductID;

ALTER TABLE Production.ProductDocument DROP CONSTRAINT FK\_ProductDocument\_Product\_ProductID;

ALTER TABLE Production.ProductInventory DROP CONSTRAINT FK\_ProductInventory\_Product\_ProductID;

ALTER TABLE Production.ProductListPriceHistory DROP CONSTRAINT FK\_ProductListPriceHistory\_Product\_ProductID;

ALTER TABLE Production.ProductProductPhoto DROP CONSTRAINT FK\_ProductProductPhoto\_Product\_ProductID;

ALTER TABLE Production.ProductReview DROP CONSTRAINT FK\_ProductReview\_Product\_ProductID;

ALTER TABLE Purchasing.ProductVendor DROP CONSTRAINT FK\_ProductVendor\_Product\_ProductID;

ALTER TABLE Purchasing.PurchaseOrderDetail DROP CONSTRAINT FK\_PurchaseOrderDetail\_Product\_ProductID;

ALTER TABLE Sales.ShoppingCartItem DROP CONSTRAINT FK\_ShoppingCartItem\_Product\_ProductID;

ALTER TABLE Sales.SpecialOfferProduct DROP CONSTRAINT FK\_SpecialOfferProduct\_Product\_ProductID;

ALTER TABLE Production.TransactionHistory DROP CONSTRAINT FK\_TransactionHistory\_Product\_ProductID;

ALTER TABLE Production.WorkOrder DROP CONSTRAINT FK\_WorkOrder\_Product\_ProductID;

*Drop Primary Key Constraint From Partition Table Candidate*

Now we can drop primary key constraint which prevents sql developers drop existing clustered index on partitioned table.

ALTER TABLE Production.Product DROP CONSTRAINT PK\_Product\_ProductID

*ALTER Non-Partitioned Table to a SQL Partition Table*

Let's execute CREATE CLUSTERED INDEX ON partition scheme we have created in order to partition existing non partitioned table Product in SQL Server to partition it according to partition function pf\_PartitionFunction.

CREATE CLUSTERED INDEX PK\_Product\_Name

ON Production.Product(Name) ON ps\_PartitionScheme (Name)

*SELECT From Partition Table*

Let's see if we really partitioned a non partitioned table and modified it as a partition table with new cluster index.

SELECT \* FROM Production.Product WHERE $PARTITION.pf\_PartitionFunction(Name) = 1

SELECT \* FROM Production.Product WHERE $PARTITION.pf\_PartitionFunction(Name) = 2

SELECT \* FROM Production.Product WHERE $PARTITION.pf\_PartitionFunction(Name) = 3

SELECT \* FROM Production.Product WHERE $PARTITION.pf\_PartitionFunction(Name) = 4

Or better use the following sql query

SELECT MIN(Name), MAX(Name), COUNT(\*) FROM Production.Product

WHERE $PARTITION.pf\_PartitionFunction(Name) = 1

UNION ALL

SELECT MIN(Name), MAX(Name), COUNT(\*) FROM Production.Product

WHERE $PARTITION.pf\_PartitionFunction(Name) = 2

UNION ALL

SELECT MIN(Name), MAX(Name), COUNT(\*) FROM Production.Product

WHERE $PARTITION.pf\_PartitionFunction(Name) = 3

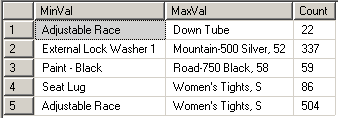
UNION ALL

SELECT MIN(Name), MAX(Name), COUNT(\*) FROM Production.Product

WHERE $PARTITION.pf\_PartitionFunction(Name) = 4

UNION ALL

SELECT MIN(Name), MAX(Name), COUNT(\*) FROM Production.Product



I hope to partition table SQL Server 2008, or on SQL Server 2008 R2 version is not a great deal after reading this sql tutorial for SQL Server administrators and t-sql developers.