[<< Sliding Window Table Partitioning](http://weblogs.sqlteam.com/dang/archive/2008/08/30/Sliding-Window-Table-Partitioning.aspx) | [Home](http://weblogs.sqlteam.com/dang/default.aspx) | [Automating RANGE RIGHT Sliding Window Maintenance >>](http://weblogs.sqlteam.com/dang/archive/2008/09/17/Automating-RANGE-RIGHT-Sliding-Window-Maintenance.aspx)

[Automating Sliding Window Maintenance](http://weblogs.sqlteam.com/dang/archive/2008/09/14/Automating-Sliding-Window-Maintenance.aspx)

I mentioned in my last post that partitioning provides a great way to manage a time-based sliding window. Unfortunately, there are no out-of-the-box tools to facilitate adjusting the sliding window so I decided to share scripts I’ve developed to automate the task.

Details of a time-based sliding window strategy vary slightly depending on whether the partition function is RANGE RIGHT or RANGE LEFT. The default RANGE LEFT is pervasive so I’ll post a RANGE LEFT example here. I’ll follow up with a RANGE RIGHT version in my next post.

The stored procedure below shows how you can automate a RANGE LEFT daily sliding window for a datetime data type partition function. This sample procedure can be adjusted for other time-including data types simply by changing the data type declarations and adjusting the boundary calculation to return the latest possible time. See my [Sliding Window Table Partitioning](http://weblogs.sqlteam.com/dang/archive/2008/08/30/Sliding-Window-Table-Partitioning.aspx) post for a list of the appropriate time value boundary values by data type.

The @RetentionDays parameter specifies the desired data retention in days. This value is used to purge old data using SWITCH and also adjust the partition function so that historical, current and future data are in separate partitions. See my [Sliding Window Table Partitioning](http://weblogs.sqlteam.com/dang/archive/2008/08/30/Sliding-Window-Table-Partitioning.aspx) post for more information on how this approach avoids costly data movement during SPLIT and MERGE.

This sample stored proc can be tweaked in order to address environment-specific requirements (object names, filegroups, multiple partition schemes) other time intervals (e.g. month, year). I’ve found the verbose messages (SELECT and RAISERROR) are useful for troubleshooting but these can be removed if desired.

Note that this sample uses a staging table as a SWITCH target that is partitioned identically to the primary table. I’ve found that this technique makes it easy to ensure the staging table target partition is properly aligned for SWITCH.

**Example of RANGE LEFT Sliding Window Automation**

Below are the demo objects used by the range LEFT sliding window procedure.

|  |
| --- |
| --no boundaries initially - proc will create as needed  CREATE PARTITION FUNCTION PF\_MyPartitionFunction(datetime)  AS RANGE LEFT FOR VALUES();  GO    CREATE PARTITION SCHEME PS\_MyPartitionScheme  AS PARTITION PF\_MyPartitionFunction ALL TO ([PRIMARY]);  GO    CREATE TABLE dbo.MyPartitionedTable  (  PartitionColumnDateTime datetime  ) ON PS\_MyPartitionScheme(PartitionColumnDateTime);  GO    --note staging table uses same partition scheme as primary table  CREATE TABLE dbo.MyPartitionedTable\_Staging  (  PartitionColumnDateTime datetime  ) ON PS\_MyPartitionScheme(PartitionColumnDateTime);  GO |

The sliding window proc:

|  |
| --- |
| CREATE PROC dbo.SlideRangeLeftWindow\_datetime  @RetentionDays int,  @RunDate datetime = NULL  /\*  This proc maintains a RANGE LEFT daily sliding window  based on the specified @RetentionDays. In addition to  purging old data, the partition function is adjusted to  account for scheduling issues or changes in @RetentionDays.    Partitions are split and merged so that the first partition  contains the oldest data and the last two partitions (future  data) are empty. The other petitions will contain historical  and current data for the specified number of @RetentionDays.    After successful execution, (at least) the following  partitions will exist:  - partition 1 = oldest data period  - third from last partition = current data (@RunDate)  - second from last partition = @RunDate + 1 (normally empty)  - last partition = @RunDate + 2 (normally empty)  \*/    AS    SET NOCOUNT, XACT\_ABORT ON;    DECLARE  @Error int,  @RowCount bigint,  @ErrorLine int,  @Message varchar(255),  @ExpirationDate datetime,  @PartitionBoundaryDate datetime;    SET @Error = 0;    BEGIN TRY    IF @RunDate IS NULL  BEGIN  --use current date (midnight) if no date specified  SET @RunDate = DATEADD(day, 0, DATEDIFF(day, '', GETDATE()));  END  ELSE  BEGIN  --set time to midnight of specified date  SET @RunDate = DATEADD(day, 0, DATEDIFF(day, '', @RunDate));  END    SET @Message =  'Run date = ' +  CONVERT(varchar(23), @RunDate, 121) +  ', Retention days = ' +  CAST(@RetentionDays AS varchar(10));    RAISERROR (@Message, 0, 1) WITH NOWAIT;    --set datetime boundary to latest posible datetime of previous day  SET @RunDate = DATEADD(millisecond, -3, @RunDate);    --calculate expiration date based on @RetentionDays and @RunDate  SET @ExpirationDate = DATEADD(day, @RetentionDays \* -1, @RunDate);    RAISERROR ('Partition boundaries before maintenance:', 0, 1) WITH NOWAIT;  SELECT CAST(prv.value AS datetime) AS PartitionBoundary  FROM sys.partition\_functions AS pf  JOIN sys.partition\_range\_values AS prv ON  prv.function\_id = pf.function\_id  WHERE pf.name = 'PF\_MyPartitionFunction';    --get oldest existing boundary  SELECT  @PartitionBoundaryDate =  CAST(MIN(prv.value) AS datetime)  FROM sys.partition\_functions AS pf  JOIN sys.partition\_range\_values AS prv ON  prv.function\_id = pf.function\_id  WHERE pf.name = 'PF\_MyPartitionFunction';    BEGIN TRAN;    --acquire exclusive table lock to prevent deadlocking  --with concurrent activity.  SELECT TOP 1 @error = 0  FROM dbo.MyPartitionedTable WITH (TABLOCKX, HOLDLOCK);    --If first partition boundary is later than expiration date,  --split first partition so that it contains only expired data.  IF @PartitionBoundaryDate > @ExpirationDate  OR @PartitionBoundaryDate IS NULL  BEGIN  SET @PartitionBoundaryDate = @ExpirationDate;  ALTER PARTITION SCHEME PS\_MyPartitionScheme  NEXT USED [PRIMARY];  ALTER PARTITION FUNCTION PF\_MyPartitionFunction()  SPLIT RANGE(@PartitionBoundaryDate);  SET @Message =  'Created boundary ' +  CONVERT(varchar(30), @PartitionBoundaryDate, 121);  RAISERROR(@Message, 0, 1) WITH NOWAIT;  END;    --Switch and merge partitions older that retention period  WHILE @PartitionBoundaryDate <= @ExpirationDate  BEGIN    --Ensure target staging partition is empty  TRUNCATE TABLE dbo.MyPartitionedTable\_Staging;  ALTER TABLE dbo.MyPartitionedTable SWITCH PARTITION 1  TO dbo.MyPartitionedTable\_Staging PARTITION 1;  --get count of purged rows for message  SELECT @RowCount = rows  FROM sys.partitions  WHERE  object\_id = OBJECT\_ID(N'MyPartitionedTable\_Staging')  AND partition\_number = 1;  --purge data permanently  TRUNCATE TABLE dbo.MyPartitionedTable\_Staging;  SET @Message =  'Moved data older than ' +  CONVERT(varchar(23), @PartitionBoundaryDate, 121) +  ' (' + CAST(@RowCount as varchar(20)) + ' rows) to staging table';  RAISERROR(@Message, 0, 1) WITH NOWAIT;    --merge first and second partitions  ALTER PARTITION FUNCTION PF\_MyPartitionFunction()  MERGE RANGE(@PartitionBoundaryDate);  SET @Message =  'Removed boundary ' +  CONVERT(varchar(30), @PartitionBoundaryDate, 121);    RAISERROR(@Message, 0, 1) WITH NOWAIT;    --get oldest partition boundary for next iteration  SET @PartitionBoundaryDate = NULL;  SELECT  @PartitionBoundaryDate =  CAST(MIN(prv.value) AS datetime)  FROM sys.partition\_functions AS pf  JOIN sys.partition\_range\_values AS prv ON  prv.function\_id = pf.function\_id  WHERE pf.name = 'PF\_MyPartitionFunction';  END;    --Make sure we have individual partitons for tomorrow  --and future days.  SET @PartitionBoundaryDate = DATEADD(day, 1, @RunDate);  WHILE @PartitionBoundaryDate <= DATEADD(day, 2, @RunDate)  BEGIN  IF NOT EXISTS  (  SELECT prv.value  FROM sys.partition\_functions AS pf  JOIN sys.partition\_range\_values AS prv ON  prv.function\_id = pf.function\_id  WHERE  pf.name = 'PF\_MyPartitionFunction'  AND CAST(prv.value AS datetime) = @PartitionBoundaryDate  )  BEGIN  ALTER PARTITION SCHEME PS\_MyPartitionScheme  NEXT USED [PRIMARY];  ALTER PARTITION FUNCTION PF\_MyPartitionFunction()  SPLIT RANGE(@PartitionBoundaryDate);  SET @Message =  'Created boundary ' +  CONVERT(varchar(30), @PartitionBoundaryDate, 121);    RAISERROR(@Message, 0, 1) WITH NOWAIT;  END;  ELSE  BEGIN  SET @Message =  'Partition already exists for boundary ' +  CONVERT(varchar(30), @PartitionBoundaryDate, 121);  RAISERROR(@Message, 0, 1) WITH NOWAIT;  END;    --calc boundary date for next iteration  SET @PartitionBoundaryDate =  DATEADD(day, 1, @PartitionBoundaryDate);  END    COMMIT;    END TRY  BEGIN CATCH    SELECT  @Error = ERROR\_NUMBER(),  @Message = ERROR\_MESSAGE(),  @ErrorLine = ERROR\_LINE();    SET @Message =  'Partition maintenenace failed with error %d at line %d: ' +  @Message;  RAISERROR(@Message, 16, 1, @Error, @ErrorLine) WITH NOWAIT;    IF @@TRANCOUNT > 0  BEGIN  ROLLBACK;  END;    END CATCH;    RAISERROR ('Partition boundaries after maintenance', 0, 1) WITH NOWAIT;  SELECT CAST(prv.value AS datetime) AS PartitionBoundary  FROM sys.partition\_functions AS pf  JOIN sys.partition\_range\_values AS prv ON  prv.function\_id = pf.function\_id  WHERE pf.name = 'PF\_MyPartitionFunction';    Done:    RETURN @error;  GO |

Demo scripts:

|  |
| --- |
| --day 1: create initial 3 partitions  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-01T01:00:00');  EXEC dbo.SlideRangeLeftWindow\_datetime @RetentionDays = 1, @RunDate = '2008-09-01T00:00:00';  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-01T02:00:00');    --day 2: create partition for future data (4 partitions)  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-02T01:00:00');  EXEC dbo.SlideRangeLeftWindow\_datetime @RetentionDays = 1, @RunDate = '2008-09-02T00:00:00';  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-02T02:00:00');  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-02T03:00:00');    --day 3: purge data and create partition for future data (rolling 4 partitions)  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-03T01:00:00');  EXEC dbo.SlideRangeLeftWindow\_datetime @RetentionDays = 1, @RunDate = '2008-09-03T00:00:00';  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-03T02:00:00');  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-03T03:00:00');  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-03T04:00:00');    --day 5: catch-up after missed day 4  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-04T01:00:00');  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-04T02:00:00');  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-04T03:00:00');  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-04T04:00:00');  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-04T05:00:00');  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-05T01:00:00');  EXEC dbo.SlideRangeLeftWindow\_datetime @RetentionDays = 1, @RunDate = '2008-09-05T00:00:00'; |

Below is the output from the demo script:

|  |
| --- |
| Run date = 2008-09-01 00:00:00.000, Retention days = 1  Partition boundaries before maintenance:  PartitionBoundary  -----------------------    Created boundary 2008-08-30 23:59:59.997  Moved data older than 2008-08-30 23:59:59.997 (0 rows) to staging table  Removed boundary 2008-08-30 23:59:59.997  Created boundary 2008-09-01 23:59:59.997  Created boundary 2008-09-02 23:59:59.997  Partition boundaries after maintenance  PartitionBoundary  -----------------------  2008-09-01 23:59:59.997  2008-09-02 23:59:59.997 |
| Run date = 2008-09-02 00:00:00.000, Retention days = 1  Partition boundaries before maintenance:  PartitionBoundary  -----------------------  2008-09-01 23:59:59.997  2008-09-02 23:59:59.997    Created boundary 2008-08-31 23:59:59.997  Moved data older than 2008-08-31 23:59:59.997 (0 rows) to staging table  Removed boundary 2008-08-31 23:59:59.997  Partition already exists for boundary 2008-09-02 23:59:59.997  Created boundary 2008-09-03 23:59:59.997  Partition boundaries after maintenance  PartitionBoundary  -----------------------  2008-09-01 23:59:59.997  2008-09-02 23:59:59.997  2008-09-03 23:59:59.997 |
| Run date = 2008-09-03 00:00:00.000, Retention days = 1  Partition boundaries before maintenance:  PartitionBoundary  -----------------------  2008-09-01 23:59:59.997  2008-09-02 23:59:59.997  2008-09-03 23:59:59.997    Moved data older than 2008-09-01 23:59:59.997 (2 rows) to staging table  Removed boundary 2008-09-01 23:59:59.997  Partition already exists for boundary 2008-09-03 23:59:59.997  Created boundary 2008-09-04 23:59:59.997  Partition boundaries after maintenance  PartitionBoundary  -----------------------  2008-09-02 23:59:59.997  2008-09-03 23:59:59.997  2008-09-04 23:59:59.997 |
| Run date = 2008-09-05 00:00:00.000, Retention days = 1  Partition boundaries before maintenance:  PartitionBoundary  -----------------------  2008-09-02 23:59:59.997  2008-09-03 23:59:59.997  2008-09-04 23:59:59.997    Moved data older than 2008-09-02 23:59:59.997 (3 rows) to staging table  Removed boundary 2008-09-02 23:59:59.997  Moved data older than 2008-09-03 23:59:59.997 (4 rows) to staging table  Removed boundary 2008-09-03 23:59:59.997  Created boundary 2008-09-05 23:59:59.997  Created boundary 2008-09-06 23:59:59.997  Partition boundaries after maintenance  PartitionBoundary  -----------------------  2008-09-04 23:59:59.997  2008-09-05 23:59:59.997  2008-09-06 23:59:59.997 |

posted @ Sunday, September 14, 2008 9:23 AM

[Print](javascript:window.print();)

**Comments on this entry:**

[**#**](http://weblogs.sqlteam.com/dang/archive/2008/09/14/Automating-Sliding-Window-Maintenance.aspx#40170)**re: Automating Sliding Window Maintenance**

Left by peleg at 12/24/2008 9:51 AM

Gravatar

i have an un paratiotioned index, and when i try to make a switch i get : "Partition maintenenace failed with error 7733 at line 106: 'ALTER TABLE SWITCH' statement failed. The table 'NReports5.dbo.TargetTable2' is partitioned while index 'IX\_TargetTable1\_RecipientSender' is not partitioned."  
what can i do?do i need to fix it on both original table and staging table?  
  
thnaks in advance  
peleg

[**#**](http://weblogs.sqlteam.com/dang/archive/2008/09/14/Automating-Sliding-Window-Maintenance.aspx#40210)**re: Automating Sliding Window Maintenance**

Left by Dan Guzman at 1/3/2009 11:14 AM

Gravatar

This error means that the table is partitioned but the index listed in the error message is not. One of the requirements in order to use SWITCH  
is that a table and it's indexes must be aligned (partitioned similarly). This implies that the tables cannot contain a mix of partitioned and non-partitioned indexes. See the Books Online for details.  
  
The script below (based on DDL from your separate email) shows how you can change the existing non-partitioned index to a partitioned one so that up can use SWITCH:  
  
CREATE TABLE [dbo].[TargetTable2\_Staging](  
[StartDate] [varchar](15) NOT NULL,  
[Column1] [varchar](25) NOT NULL,  
[Column2] [varchar](25) NOT NULL,  
[id] [bigint] IDENTITY(1,1) NOT NULL,  
CONSTRAINT [PK\_TargetTable2\_Staging] PRIMARY KEY CLUSTERED   
(  
[StartDate] ASC,  
[id] ASC  
) ON [PRIMARY]  
) ON [PRIMARY]  
GO  
  
CREATE NONCLUSTERED INDEX [IX\_TargetTable2\_Staging] ON  
[dbo].[TargetTable2\_Staging]   
(  
Column1,  
Column2  
) ON [PRIMARY]  
GO  
  
CREATE PARTITION FUNCTION [PF\_TargetTable2](varchar(15))   
AS RANGE LEFT FOR VALUES (N'20081210000000', N'20081211000000',   
N'20081212000000', N'20081213000000', N'20081214000000', N'20081215000000',  
N'20081216000000', N'20081217000000', N'20081218000000', N'20081230000000')  
GO  
  
CREATE PARTITION SCHEME [PS\_TargetTable2] AS PARTITION  
[PF\_TargetTable2] TO ([PRIMARY], [PRIMARY], [PRIMARY], [PRIMARY],   
[PRIMARY], [PRIMARY], [PRIMARY], [PRIMARY], [PRIMARY], [PRIMARY], [PRIMARY])  
GO  
  
CREATE TABLE [dbo].[TargetTable2](  
[StartDate] [varchar](15) NOT NULL,  
[Column1] [varchar](25) NOT NULL,  
[Column2] [varchar](25) NOT NULL,  
[id] [bigint] IDENTITY(1,1) NOT NULL,  
CONSTRAINT [PK\_TargetTable2] PRIMARY KEY CLUSTERED   
(  
[StartDate] ASC,  
[id] ASC  
) ON [PS\_TargetTable2]([StartDate])  
) ON [PS\_TargetTable2]([StartDate])  
GO  
  
CREATE NONCLUSTERED INDEX [IX\_TargetTable2] ON [dbo].[TargetTable2]   
(  
Column1,  
Column2  
) ON [PRIMARY]  
GO  
  
--this will fail  
ALTER TABLE dbo.TargetTable2  
SWITCH PARTITION 1 TO TargetTable2\_Staging  
/\*  
Msg 7733, Level 16, State 4, Line 1  
'ALTER TABLE SWITCH' statement failed. The table 'TechnologyStack.dbo.TargetTable2' is partitioned while index 'IX\_TargetTable2' is not partitioned.  
\*/  
  
--recreate source table index as partitioned  
CREATE NONCLUSTERED INDEX [IX\_TargetTable2] ON  
[dbo].[TargetTable2]   
(  
Column1,  
Column2  
)  
WITH (DROP\_EXISTING = ON)  
ON [PS\_TargetTable2]([StartDate])   
  
--this will succeed  
ALTER TABLE dbo.TargetTable2  
SWITCH PARTITION 1 TO TargetTable2\_Staging