**More Results searching for “*range right*”**

It looks like you found this post via a search engine result.  
Here are a few other posts you might find interesting:

* [Sliding Window Table Partitioning](http://weblogs.sqlteam.com/dang/archive/2008/08/30/Sliding-Window-Table-Partitioning.aspx)
* [Partition Details and Row Counts](http://weblogs.sqlteam.com/dang/archive/2008/12/11/Partition-Details-and-Row-Counts.aspx)
* [Automating Sliding Window Maintenance](http://weblogs.sqlteam.com/dang/archive/2008/09/14/Automating-Sliding-Window-Maintenance.aspx)
* [Move a Partition to a Different File Group Efficiently](http://weblogs.sqlteam.com/dang/archive/2011/04/17/move-a-partition-to-a-different-file-group-efficiently.aspx)

[***Click for all Search Results for range right***](http://weblogs.sqlteam.com/dang/search.aspx?q=range%20right)

[<< Automating Sliding Window Maintenance](http://weblogs.sqlteam.com/dang/archive/2008/09/14/Automating-Sliding-Window-Maintenance.aspx) | [Home](http://weblogs.sqlteam.com/dang/default.aspx) | [Partition Details and Row Counts >>](http://weblogs.sqlteam.com/dang/archive/2008/12/11/Partition-Details-and-Row-Counts.aspx)

[Automating RANGE RIGHT Sliding Window Maintenance](http://weblogs.sqlteam.com/dang/archive/2008/09/17/Automating-RANGE-RIGHT-Sliding-Window-Maintenance.aspx)

I posted example scripts to automate RANGE LEFT sliding window maintenance in my last post. As promised, I am sharing a RANGE RIGHT version in this post.

I personally prefer a RANGE RIGHT partition function when partitioning on a data type that includes time. RANGE RIGHT allows specification of an exact date boundary instead of the maximum date/time value needed for RANGE LEFT to keep all data for a given date in the same partition. Another nicety with RANGE RIGHT is that same boundaries can be used in a RANGE RIGHT partition function of any date/time data type. In contrast, the time component of RANGE LEFT boundary values must be customized for the specific data type as I described in [Sliding Window Table Partitioning](http://weblogs.sqlteam.com/dang/archive/2008/08/30/Sliding-Window-Table-Partitioning.aspx).

The downside with RANGE RIGHT is that maintaining the sliding window isn’t quite as intuitive as with RANGE LEFT. Instead of switching out and merging the first partition during purge/archive, one needs to switch out and merge the second partition. This practice avoids the costly movement of retained data from the removed second partition into the retained first partition. Both the first and second partitions are empty during the merge so no data need to be move moved. The first partition is normally empty at all times.

The stored procedure below shows how you can automate a RANGE RIGHT daily sliding window. The main differences between this version and the RANGE LEFT version I posted in Automating Sliding Window Maintenance are

1)      Removed boundary calculation of maximum time (DATEADD(millisecond, -3, @RunDate)).

2)      Added a conditional create of upper boundary of second partition (oldest period retained in third partition)

3)      Added $PARTITION for SWITCH instead of hard-coding the partition number

**Example of RANGE RIGHT Sliding Window Automation**

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

|  |
| --- |
| --no boundaries initially - proc will create as needed  CREATE PARTITION FUNCTION PF\_MyPartitionFunction(datetime)  AS RANGE RIGHT 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.SlideRangeRightWindow\_datetime  @RetentionDays int,  @RunDate datetime = NULL  /\*  This proc maintains a RANGE RIGHT daily sliding window  based on the specified @RetentionDays. It is intended to  be scheduled daily shortly after midnight. 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  boundary is the oldest retained data date and the last  boundary is the next day. Other partitions contain current  and historical data for the specifiednumber of @RetentionDays.    After successful execution, (at least) the following  partitions will exist:  - partition 1 = data older than retained date (empty)  - other partitions = hitorical data (@RunDate - 1 and earlier)  - second from last partition = current data (@RunDate)  - last partition = future data (@RunDate + 1) (empty)    \*/    AS    SET NOCOUNT, XACT\_ABORT ON;    DECLARE  @Error int,  @RowCount bigint,  @ErrorLine int,  @Message varchar(255),  @OldestRetainedDate 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    --calculate oldest retention date based on @RetentionDays and @RunDate  SET @OldestRetainedDate = DATEADD(day, @RetentionDays \* -1, @RunDate);    SET @Message =  'Run date = ' +  + CONVERT(varchar(23), @RunDate, 121)  + ', Retention days = '  + CAST(@RetentionDays AS varchar(10))  + ', Oldest retained data date = '  + CONVERT(varchar(23), @OldestRetainedDate, 121);    RAISERROR (@Message, 0, 1) WITH NOWAIT;    BEGIN TRAN;    --acquire exclusive table lock to prevent deadlocking  --with concurrent activity.  SELECT TOP 1 @error = 0  FROM dbo.MyPartitionedTable WITH (TABLOCKX, HOLDLOCK);    --make sure we have a boundary for oldest retained period  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) = @OldestRetainedDate  )  BEGIN  ALTER PARTITION SCHEME PS\_MyPartitionScheme  NEXT USED [PRIMARY];  ALTER PARTITION FUNCTION PF\_MyPartitionFunction()  SPLIT RANGE(@OldestRetainedDate);  SET @Message =  'Created boundary for oldest retained data ('  + CONVERT(varchar(30), @OldestRetainedDate, 121) + ')';    RAISERROR(@Message, 0, 1) WITH NOWAIT;  END  ELSE  BEGIN  SET @Message =  'Oldest retained data boundary already exists ('  + CONVERT(varchar(30), @OldestRetainedDate, 121) + ')';    RAISERROR(@Message, 0, 1) WITH NOWAIT;  END    --get earliest expired boundary  SET @PartitionBoundaryDate = NULL;  SELECT  @PartitionBoundaryDate =  MIN(CAST(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'  AND CAST(prv.value AS datetime) < @OldestRetainedDate;    --get rowcount of first partition  SELECT @RowCount = rows  FROM sys.partitions  WHERE  object\_id = OBJECT\_ID(N'dbo.MyPartitionedTable')  AND partition\_number = 1  AND index\_id IN(0, 1);    --purge data from first partition if not empty  IF @RowCount > 0  BEGIN  TRUNCATE TABLE dbo.MyPartitionedTable\_Staging;  ALTER TABLE dbo.MyPartitionedTable SWITCH PARTITION 1  TO dbo.MyPartitionedTable\_Staging PARTITION 1;  TRUNCATE TABLE dbo.MyPartitionedTable\_Staging;  SET @Message =  'Purged data older than '  + CONVERT(varchar(23), @PartitionBoundaryDate, 121)  + ' (' + CAST(@RowCount as varchar(20)) + ' rows)';  RAISERROR(@Message, 0, 1) WITH NOWAIT;  END  ELSE  BEGIN  SET @Message =  'First partition is empty. No data older than '  + CONVERT(varchar(23), @OldestRetainedDate, 121);  RAISERROR(@Message, 0, 1) WITH NOWAIT;  END    --Switch and merge expired data partitions, starting with the earliest  WHILE @PartitionBoundaryDate < @OldestRetainedDate  BEGIN    --get count of rows to be purged  SELECT @RowCount = rows  FROM sys.partitions  WHERE  object\_id = OBJECT\_ID(N'MyPartitionedTable')  AND partition\_number = $PARTITION.PF\_MyPartitionFunction(@PartitionBoundaryDate)  AND index\_id IN(0, 1);    --purge data, if needed  IF @RowCount > 0  BEGIN  --move data to staging table  TRUNCATE TABLE dbo.MyPartitionedTable\_Staging;  ALTER TABLE dbo.MyPartitionedTable SWITCH PARTITION $PARTITION.PF\_MyPartitionFunction(@PartitionBoundaryDate)  TO dbo.MyPartitionedTable\_Staging PARTITION $PARTITION.PF\_MyPartitionFunction(@PartitionBoundaryDate);      --purge data from staging table  TRUNCATE TABLE dbo.MyPartitionedTable\_Staging;  SET @Message =  'Purged data for boundary '  + CONVERT(varchar(23), @PartitionBoundaryDate, 121)  + ' (' + CAST(@RowCount as varchar(20)) + ' rows)';  RAISERROR(@Message, 0, 1) WITH NOWAIT;  END  ELSE  BEGIN  SET @Message =  'Partition for boundary '  + CONVERT(varchar(23), @PartitionBoundaryDate, 121)  + ' is empty';  RAISERROR(@Message, 0, 1) WITH NOWAIT;  END    --remove purged partition  ALTER PARTITION FUNCTION PF\_MyPartitionFunction()  MERGE RANGE(@PartitionBoundaryDate);  SET @Message =  'Removed boundary '  + CONVERT(varchar(30), @PartitionBoundaryDate, 121);    RAISERROR(@Message, 0, 1) WITH NOWAIT;    --get earliest boundary before retention date for next iteration  SET @PartitionBoundaryDate = NULL;  SELECT  @PartitionBoundaryDate =  MIN(CAST(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'  AND CAST(prv.value AS datetime) < @OldestRetainedDate;  END;    --Make sure we have an empty partition for tomorrow  SET @PartitionBoundaryDate = DATEADD(day, 1, @RunDate);  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 future data '  + CONVERT(varchar(30), @PartitionBoundaryDate, 121);    RAISERROR(@Message, 0, 1) WITH NOWAIT;  END  ELSE  BEGIN  SET @Message =  'Partition already exists for future boundary '  + CONVERT(varchar(30), @PartitionBoundaryDate, 121);  RAISERROR(@Message, 0, 1) WITH NOWAIT;  END;    COMMIT;    END TRY  BEGIN CATCH    SELECT  @Error = ERROR\_NUMBER(),  @Message = ERROR\_MESSAGE(),  @ErrorLine = ERROR\_LINE();    RAISERROR('Partition maintenenace failed with error %d at line %d: %s', 16, 1, @Error, @ErrorLine, @Message) WITH NOWAIT;    IF @@TRANCOUNT > 0  BEGIN  ROLLBACK;  END;    END CATCH;    RETURN @Error;  GO |

Demo scripts:

|  |
| --- |
| --day 1: create initial 4 partitions  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-01T01:00:00');  EXEC dbo.SlideRangeRightWindow\_datetime @RetentionDays = 1, @RunDate = '2008-09-01T00:00:00';  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-01T02:00:00');    --day 2: purge data and create partition for future data (rolling 4 partitions)  INSERT INTO dbo.MyPartitionedTable VALUES('2008-09-02T01:00:00');  EXEC dbo.SlideRangeRightWindow\_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.SlideRangeRightWindow\_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.SlideRangeRightWindow\_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, Oldest retained data date = 2008-08-31 00:00:00.000 Created boundary for oldest retained data (2008-08-31 00:00:00.000)  Created boundary future data 2008-09-02 00:00:00.000 Run date = 2008-09-02 00:00:00.000, Retention days = 1, Oldest retained data date = 2008-09-01 00:00:00.000 Created boundary for oldest retained data (2008-09-01 00:00:00.000) First partition is empty.  No data older than 2008-08-31 00:00:00.000 Partition for boundary 2008-08-31 00:00:00.000 is empty Removed boundary 2008-08-31 00:00:00.000 Created boundary future data 2008-09-03 00:00:00.000 Run date = 2008-09-03 00:00:00.000, Retention days = 1, Oldest retained data date = 2008-09-02 00:00:00.000 Oldest retained data boundary already exists (2008-09-02 00:00:00.000) First partition is empty.  No data older than 2008-09-01 00:00:00.000 Purged data for boundary 2008-09-01 00:00:00.000 (2 rows) Removed boundary 2008-09-01 00:00:00.000 Created boundary future data 2008-09-04 00:00:00.000 Run date = 2008-09-05 00:00:00.000, Retention days = 1, Oldest retained data date = 2008-09-04 00:00:00.000 Oldest retained data boundary already exists (2008-09-04 00:00:00.000) First partition is empty.  No data older than 2008-09-02 00:00:00.000 Purged data for boundary 2008-09-02 00:00:00.000 (3 rows) Removed boundary 2008-09-02 00:00:00.000 Purged data for boundary 2008-09-03 00:00:00.000 (4 rows) Removed boundary 2008-09-03 00:00:00.000 Created boundary future data 2008-09-06 00:00:00.000 |

posted @ Wednesday, September 17, 2008 7:03 AM

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

**Comments on this entry:**

[**#**](http://weblogs.sqlteam.com/dang/archive/2008/09/17/Automating-RANGE-RIGHT-Sliding-Window-Maintenance.aspx#41283)**re: Automating RANGE RIGHT Sliding Window Maintenance**

Left by Jay at 7/8/2009 11:55 AM

Gravatar

this one is really a very good article but i have doubt that when there is PK-FK with this partition table then i don't think this sliding window will work as there will be violation of FK.

[**#**](http://weblogs.sqlteam.com/dang/archive/2008/09/17/Automating-RANGE-RIGHT-Sliding-Window-Maintenance.aspx#41300)**re: Automating RANGE RIGHT Sliding Window Maintenance**

Left by Dan Guzman at 7/10/2009 8:28 AM

Gravatar

SWITCH is not allowed when the table is referenced (PK side) in a foreign key constraint. This is the same restriction as the TRUNCATE DDL statement. Note that there is no restriction when the partitioned table is a referencing table (FK side) of the relationship.

[**#**](http://weblogs.sqlteam.com/dang/archive/2008/09/17/Automating-RANGE-RIGHT-Sliding-Window-Maintenance.aspx#41445)**re: Automating RANGE RIGHT Sliding Window Maintenance**

Left by Rachel at 8/6/2009 1:44 PM

Gravatar

I'm running into an interesting problem with partitioning: we want to move older partitions in a table off to slower storage on the same server, but I haven't seen any "recipes" on how to do that. This is an intermediate step before moving partitions off to an archive table - we want to keep the "middle-aged" data alive, but we don't have room to keep it all on our fastest drives.  
  
Is this a reasonable thing to do?

[**#**](http://weblogs.sqlteam.com/dang/archive/2008/09/17/Automating-RANGE-RIGHT-Sliding-Window-Maintenance.aspx#41513)**re: Automating RANGE RIGHT Sliding Window Maintenance**

Left by Dan Guzman at 8/18/2009 10:22 PM

Gravatar

It's certainly reasonable to move less frequently accessed data to shower drives. One way to accomplish this is to setup the partition scheme with one filegroup/file per partition. Prior to the normal periodic partition maintenance, move the file(s) to the slower drives as you would any planned file relocation (set database offline, move file, alter database...modify file and set database online).  
  
The process is a bit more involved if you need to keep the database online because you'll need to copy data to the slower drives. This can be done with a simple merge, alter..next used and split but can take awhile for large partitions.   
  
To maximize availability, you might consider switching the data out of the adjacent affected partitions (latest one on slow disks, earliest one on fast disks), move the staging table to the slow filegroup merge/split the adjacent partitions so that both are on the slow filegroup and then switch both back in. Data from the affected partitions will be temporarily unavailable.

[**#**](http://weblogs.sqlteam.com/dang/archive/2008/09/17/Automating-RANGE-RIGHT-Sliding-Window-Maintenance.aspx#45462)**re: Automating RANGE RIGHT Sliding Window Maintenance**

Left by MattA at 1/18/2011 12:24 PM

Gravatar

I'm running SQL Server 2008 R2 and get the following error when running the code above:  
  
(1 row(s) affected)  
Run date = 2008-09-01 00:00:00.000, Retention days = 1  
Partition boundaries before maintenance:  
Created boundary 2008-08-31 00:00:00.000  
Created boundary 2008-09-01 00:00:00.000  
Msg 50000, Level 16, State 1, Procedure SlideRangeRightWindow\_datetime, Line 494  
Partition maintenenace failed with error 3609 at line 312: The transaction ended in the trigger. The batch has been aborted.  
Partition boundaries after maintenance  
  
(1 row(s) affected)

[**#**](http://weblogs.sqlteam.com/dang/archive/2008/09/17/Automating-RANGE-RIGHT-Sliding-Window-Maintenance.aspx#45496)**re: Automating RANGE RIGHT Sliding Window Maintenance**

Left by [guzmanda](http://weblogs.sqlteam.com/dang/default.aspx) at 1/26/2011 6:45 AM

Gravatar

Do you perhaps have a DDL trigger that is rolling back the operation?

[**#**](http://weblogs.sqlteam.com/dang/archive/2008/09/17/Automating-RANGE-RIGHT-Sliding-Window-Maintenance.aspx#45605)**re: Automating RANGE RIGHT Sliding Window Maintenance**

Left by Yash Ganthe at 2/16/2011 7:38 AM

Gravatar

The article says:  
Instead of switching out and merging the first partition during purge/archive, one needs to switch out and merge the second partition.  
  
If we find the minimum boundary and start with it and do  
ALTER TABLE dbo.MyPartitionedTable SWITCH PARTITION $PARTITION.PF\_MyPartitionFunction(@PartitionBoundaryDate) - 1  
TO dbo.MyPartitionedTable\_Staging PARTITION $PARTITION.PF\_MyPartitionFunction(@PartitionBoundaryDate) - 1;  
We are actually switching out the first partition. Further we are also merging the first partition.  
  
So I do not see why this is considered different than RANGE LEFT.  
  
Also with RANGE RIGHT, if we merge partition 2, does it cause data from partition 2 to go into the filegroup that holds partition 3? Or will it cause data from partition 3 to come into the filegroup that holds partition 2?  
  
Thanks,  
Yash

[**#**](http://weblogs.sqlteam.com/dang/archive/2008/09/17/Automating-RANGE-RIGHT-Sliding-Window-Maintenance.aspx#45632)**re: Automating RANGE RIGHT Sliding Window Maintenance**

Left by [guzmanda](http://weblogs.sqlteam.com/dang/default.aspx) at 2/20/2011 11:06 AM

Gravatar

Hi, Yash.  
  
You are right that the maintenace proc I posted SWITCHed out partition 1.   
  
MERGE terminology is confusing because 2 partitions involved. The partition that contains the boundary value is dropped and data moved from the dropped partition into the remaining one. MERGE behavior is exactly opposite with RANGE LEFT and RANGE RIGHT. With RANGE LEFT, a MERGE of the first partition boundary will drop the first partition and move data into the remaining second partition(s) and corresponding filegroup(s). All partition numbers are reduced by 1 so that partition 2 becomes partition 1, etc.  
  
With RANGE RIGHT, the same merge will drop the second partition and move data into the remaining first partition(s) and corresponding filegroup(s). The partition numbers of all but the first are reduced by 1. This is what the proc I posted did.  
  
The goal of efficient merging is to drop empty partitions to avoid data movement. As you noted, the original maintenace proc I posted was suboptimal with RANGE RIGHT because it switched out the first partition and then merged the second whereas my intent was to switch out the second before the merge. This resulted in unnecessary data movement.  
  
I revised the code to switch out the earliest expired partition (normally the second partition) and then merge the first 2 partitions. Since the dropped partition is empty after the switch, no data movement is needed during the merge. This will perform much better, especially in cases where a catch-up after missed days is needed. I also added code to conditionally switch out the first partition beforehand in case it is not empty. The first partition always contains expired data because the proc creates a boundary for the oldest retained data if one does not already exists.   
  
Thanks for the feedback.

[**#**](http://weblogs.sqlteam.com/dang/archive/2008/09/17/Automating-RANGE-RIGHT-Sliding-Window-Maintenance.aspx#45797)**re: Automating RANGE RIGHT Sliding Window Maintenance**

Left by [guzmanda](http://weblogs.sqlteam.com/dang/default.aspx) at 3/13/2011 12:02 PM

Gravatar

Matt,  
  
In addition to a user DDL trigger, you might get the trigger rollback error if CDC is enabled in the database. CDC uses DDL triggers under the covers to track DDL changes.