Monitoring Table Size Growth in SQL Server

By : Yaniv Etrogi
Dec 01, 2010

General
In this article I introduce a simple process that saves tables size information at points in time which latter allows for tracking down table growth over time.

The process is useful in monitoring data growth over time and lets you see what tables are growing rapidly in what databases and can also help in estimating future growth and disk space requirements. In addition it can help prevent unexpectedly running out of disk space, or if you are already in that situation then quickly find the table(s) that for some reason have grown more than excepted and are the source to the problem.

In addition, I also show a stored procedure called *sp_DataFiles* that returns a report of disk space usage information at the data file level.

Implementation

Run these three scripts:

TableSizeHistory.sql

sp_TableSize.sql

ExecuteDatabasesCommand.sql

The process is built on top of a stored procedure called sp_TableSize which returns table level information such as row count and size.

Periodically, typically by a weekly SQL Server Agent job, the procedure will be executed against all databases using an INSERT...EXEC statement that saves the results into a database table.

Having done that we actually freeze and document size related information at the table level which can now be analyzed and so the next and final step would be to retrieve the data.

Usage

There are two main scenarios in which you can benefit from the process;

One scenario is simply viewing the report knowing what tables grow in what size and that rate at which they grow.

The second scenario would be to execute the DataGrowth stored procedure while passing the @Delta_MB input parameter a value that is not expected to be reached so that it always returns an empty result set. Now, when the day comes and this procedure returns data you know there was a growth in data exceeding your predefined threshold and can trigger a notification email.

Summary of scripts used:

TableSizeHistory.sql – Creates the PerfDB database and the TableSizeHistory table

DataGrowth.sql – Analyzes and retrieves the captured data

ExecuteDatabasesCommand.sql – A wrapper over a database cursor

sp_DataFiles.sql – Returns size information at the data file level for all databases

sp_TableSize.sql – Returns table level information for the current database

Download all the scripts

sp_TableSize

sp_TableSize is a stored procedure created in the master database and marked as a system stored procedure. The procedure works at the table level in the context of the current database, that is the database where it is executed from and returns one row per table with the information of the row count and disk space usage in units of MB sorted by disk space usage in a descending order.

The procedure is handy and easy to use with no input parameters required and the information returned is based on system meta data, meaning you do not actually access any user object so there is no effect (no shared locks and no IO) on user objects as opposed to issuing a COUNT(*) query.

Bare in mind that due to the above fact there may be a *slight* (minor) inaccuracy with the row count figure but that is not common and results from the way SQL Server's Storage Engine keeps track of data that was deleted and typically gets aligned with the accurate figure following a clustered index rebuild.

Part of the code in sp_TableSize is combined in my sp_helpindex2 stored procedure.

DataGrowth

This stored procedure analyzes and retrieves the data that was saved to the *TableSizeHistory*

table using the sp_TableSize stored procedure and returns the disk space growth in units of MB per each table.

The code identifies the first and last occurrences of each table in the requested time period as defined by the @StartTime and @EndTime input parameters and calculates the differences between the two occurrences so that the returned result set is the delta between these two values.

You can control the tables returned by the @Delta_MB input parameter so that you get to see only tables of interest, i.e. tables that grew more than a certain size.

The Delta_reserved_MB column is the sum of the Delta_data_MB + Delta_index_size_MB and this is the actual disk size the table uses. Don't let the name confuse you, I only followed Microsoft's terminology used in their *sp_spaceused* system stored procedure.

The @Database parameter has a default value of 1 and returns a second result set with information at the database level.

ExecuteDatabasesCommand

This procedure is actually a wrapper over a cursor that uses the database name from sys.databases DMV.

I created this procedure in order to reuse my code so that every time I need a cursor over databases I use it instead of recoding that cursor part. i.e. when I run an index rebuild process that uses my sp_Reindex procedure, DBCC commands for integrity checks, attach, detach etc.

sp_DataFiles

This stored procedure is not related to the data growth process but is a very useful disk space related tool.

sp_DataFiles works at the database level and returns one row per each data file on the SQL Server instance.

The procedure is actually built on top of the disk usage report query issued by SSMS (SQL Server Management Studio) which I captured in Profiler a couple of years a go and adjusted a little for my needs. Being able to get that report by executing a procedure is far more convenience and efficient than going through the entire process required via the UI until finally receiving that desired report and also, the SSMS report works on a specific database and when you need to get a more wider picture of how your disk space is consumed by your databases it is not enough.

The procedure makes use of the undocumented DBCC command *SHOWFILESTATS* which returns the total extents and the used extents information per data file and joins that information with the *sys.data_spaces* DMV (Dynamic Management View).

All the disk space figures are calculated and derived from the extent information returned by the DBCC command while the DMV only contributes the file group name to the returned results.

DBCC SHOWFILESTATS returns one row per each data file in the database and *sys.data_spaces* returns one row per each data space (which can actually be a filegroup, partition scheme, or FILESTREAM data filegroup).

The procedure also returns the space reserved and space used information which is very useful. When these two figures are far from each other (i.e. a data file that stores 10GB of data but reserves 40GB) there is space that can be reclaimed to the operating system if needed.

On the other hand when these two figures are close to each other it points out that the file Auto Grow option is kicking in and that of course is something we usually want to avoid.

Execution

1. Save the data to a table

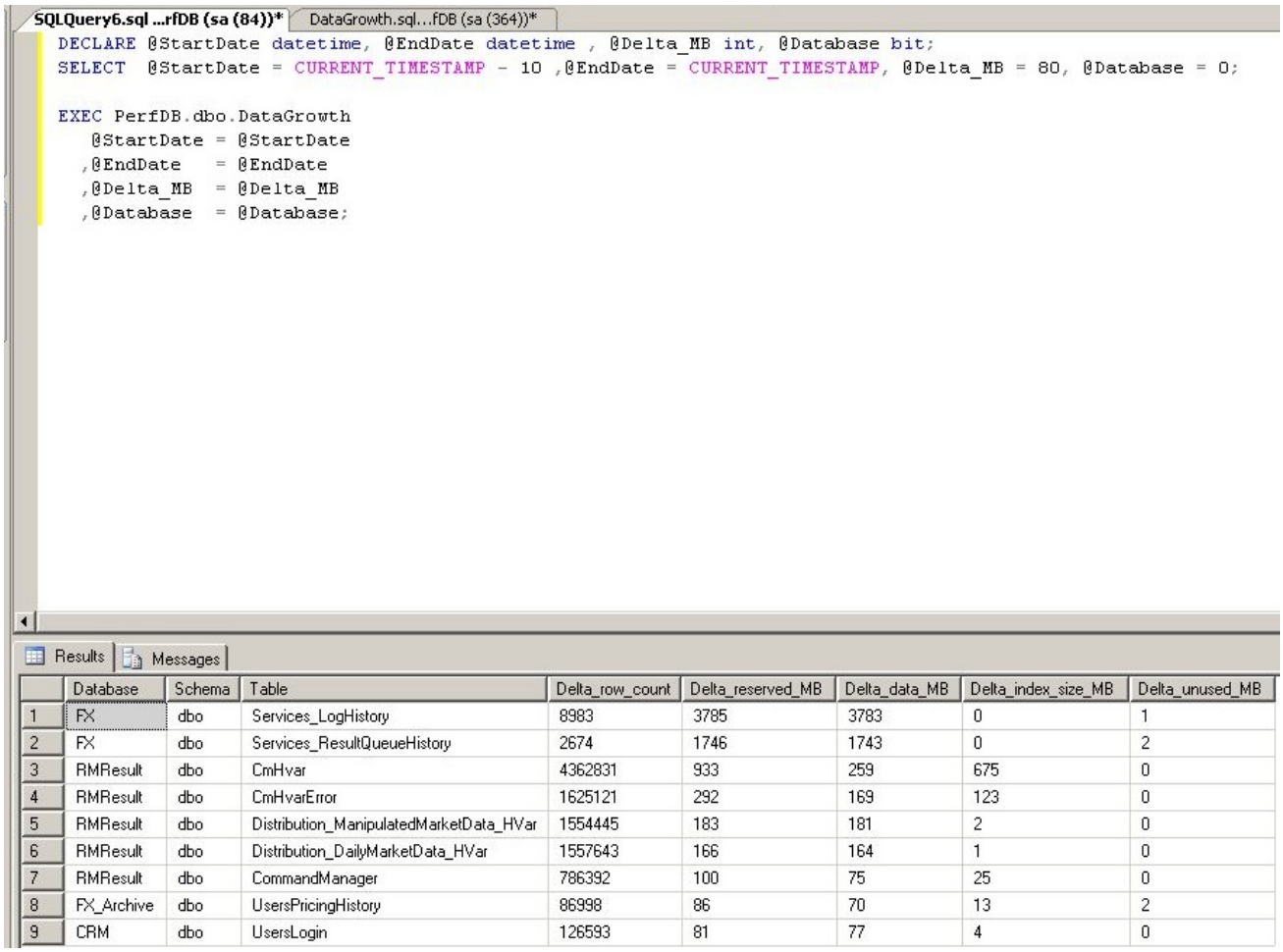
```
-- Execute sp_TableSize against all databases

-- typically by a weekly job

INSERT TableSizeHistory (l1,[Database], [Schema], [Table], row_count, reserved_MB,
data_MB,index_size_MB,unused_MB )

EXEC PerfDB.dbo.ExecuteDatabasesCommand @cmd = N'[sp_TableSize];';
```

2. Retrieve the data from the table

```
-- Retreive data growth table level informtion

EXEC Perfdb.dbo.DataGrowth

,@StartDate     = '20100101'

,@EndDate       = '20100201'

,@Delta_MB = 300; -- look for tables with a data growth greater than this threshold
```

```
-- You can also use the procedure with the default parameter values which work on the last 30 days and look for tables with a data growth greater than 200 MB

EXEC Perfdb.dbo.DataGrowth;

-- You can execute it from a monitoring tool daily with a threshold that is not excpected to be met which should return an empty result set. If the result set is not empty you have crossed the predefined threshold and trigger a notification email

DECLARE @StartDate datetime, @EndDate datetime , @Delta_MB int, @Database bit;

SELECT  @StartDate = CURRENT_TIMESTAMP - 9 ,@EndDate = CURRENT_TIMESTAMP, @Delta_MB = 100, @Database = 0;

 EXEC PerfDB.dbo.DataGrowth

      @StartDate = @StartDate

     ,@EndDate   = @EndDate

     ,@Delta_MB  = @Delta_MB

     ,@Database  = @Database;
```

Below is a sample result set from executing the above:



	Database	Schema	Table	Delta_row_count	Delta_reserved_MB	Delta_data_MB	Delta_index_size_MB	Delta_unused_MB
1	FX	dbo	Services_LogHistory	8983	3785	3783	0	1
2	FX	dbo	Services_ResultQueueHistory	2674	1746	1743	0	2
3	RMResult	dbo	CmHvar	4362831	933	259	675	0
4	RMResult	dbo	CmHvarError	1625121	292	169	123	0
5	RMResult	dbo	Distribution_ManipulatedMarketData_HVar	1554445	183	181	2	0
6	RMResult	dbo	Distribution_DailyMarketData_HVar	1557643	166	164	1	0
7	RMResult	dbo	CommandManager	786392	100	75	25	0
8	FX_Archive	dbo	UsersPricingHistory	86998	86	70	13	2
9	CRM	dbo	UsersLogin	126593	81	77	4	0

Code sp_helpindex2:

```
USE [master];
SET ANSI_NULLS ON; SET QUOTED_IDENTIFIER ON;
GO


IF  EXISTS (SELECT * FROM sys.procedures WHERE object_id =
OBJECT_ID(N'[dbo].[sp_helpindex2]') AND type IN (N'P', N'PC'))
        DROP PROCEDURE [dbo].[sp_helpindex2];
GO


CREATE PROCEDURE dbo.sp_helpindex2
(
         @Table sysname
        ,@IndexExtendedInfo bit = 0
        ,@ColumnsInfo bit = 1
)
/*
        Yaniv Etrogi 20100328
        sp_helpindex2 adds the included columns information that is not
provided by the original sp_helpindex.

        Yaniv Etrogi 20101115
        Modified the bellow line:
        ,ROW_NUMBER() OVER (PARTITION BY sc.is_included_column ORDER BY
sc.key_ordinal) ColPos
        sc.key_ordinal replaces sc.column_id to resolve a bug where in a
composite index the keys were not correctly ordered.


        Send your comments to mailto:yaniv.etrogi@gmail.com or leave your
comments at
        http://blogs.microsoft.co.il/blogs/yaniv_etrogi/
        http://www.sqlserverutilities.com
*/
AS
SET NOCOUNT ON;
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;



-- Index information
SELECT
         SCHEMA_NAME(o.SCHEMA_ID) AS [Schema]
        ,o.name AS [Table]
        ,i.name AS [Index]
        ,i.object_id
        ,i.data_space_id
        ,i.index_id
        ,i.[ignore_dup_key]
        ,i.is_unique
        ,i.is_hypothetical
        ,i.is_primary_key
        ,i.is_unique_constraint
```

```
        ,s.auto_created
        ,s.no_recompute
        ,i.[allow_row_locks]
  ,i.[allow_page_locks]
        ,i.is_disabled
        ,i.fill_factor
        ,i.is_padded
        ,LEFT(list, ISNULL(splitter-1,LEN(list))) AS [Columns]
        ,SUBSTRING(list, indCol.splitter +1, 100) AS Included_Columns
        --,COUNT(*) OVER (PARTITION BY o.[object_id]) AS Num_Indexes
INTO dbo.#Indexes
FROM sys.indexes i
INNER JOIN sys.objects o ON i.[object_id] = o.[object_id]
INNER JOIN  sys.stats s ON i.[object_id] = s.[object_id] AND i.index_id =
s.stats_id
CROSS APPLY (SELECT NULLIF(CHARINDEX('|',indexCols.list),0) splitter , list
               FROM (SELECT CAST((SELECT CASE WHEN sc.is_included_column = 1
AND sc.ColPos = 1 THEN '|' ELSE '' END + CASE WHEN sc.ColPos  > 1 THEN ', '
ELSE '' END + name
                              FROM (SELECT



                                     FROM sys.index_columns  sc
                                     INNER JOIN sys.columns c ON
sc.[object_id] = c.[object_id] AND sc.column_id = c.column_id
                                     WHERE sc.index_id = i.index_id AND
sc.[object_id] = i.[object_id] ) sc
                                              ORDER BY sc.is_included_column,
ColPos
                       FOR XML PATH (''), TYPE) AS VARCHAR(MAX))
list)indexCols ) indCol
WHERE o.name = @Table;



-- Table information
SELECT
        (row_number() OVER(ORDER BY t3.name, t2.name))%2 AS l1
        ,DB_NAME() AS [Database]
        ,t3.name AS [schemaname]
        ,t2.name AS [tablename]
        ,t1.rows AS row_count
        ,((t1.reserved + ISNULL(a4.reserved,0))* 8) / 1024 AS reserved_MB
        ,(t1.data * 8) / 1024 AS data_MB
        ,((CASE WHEN (t1.used + ISNULL(a4.used,0)) > t1.data THEN (t1.used +
ISNULL(a4.used,0)) - t1.data ELSE 0 END) * 8) /1024 AS index_size_MB
        ,((CASE WHEN (t1.reserved + ISNULL(a4.reserved,0)) > t1.used THEN
(t1.reserved + ISNULL(a4.reserved,0)) - t1.used ELSE 0 END) * 8)/1024 AS
unused_MB
INTO dbo.#Table
FROM
 (SELECT
         ps.object_id
        ,SUM (CASE WHEN (ps.index_id < 2) THEN row_count ELSE 0 END) AS [rows]
        ,SUM (ps.reserved_page_count) AS reserved
```

```
        ,SUM (CASE WHEN (ps.index_id < 2) THEN (ps.in_row_data_page_count +
ps.lob_used_page_count + ps.row_overflow_used_page_count) ELSE
(ps.lob_used_page_count + ps.row_overflow_used_page_count) END) AS data
        ,SUM (ps.used_page_count) AS used
  FROM sys.dm_db_partition_stats ps
  GROUP BY ps.[object_id]) AS t1
LEFT OUTER JOIN
 (SELECT
          it.parent_id
         ,SUM(ps.reserved_page_count) AS reserved
         ,SUM(ps.used_page_count) AS used
  FROM sys.dm_db_partition_stats ps
  INNER JOIN sys.internal_tables it ON (it.[object_id] = ps.[object_id])
WHERE it.internal_type IN (202,204)
  GROUP BY it.parent_id) AS a4 ON (a4.parent_id = t1.[object_id])
INNER JOIN sys.all_objects t2  ON ( t1.[object_id] = t2.[object_id])
INNER JOIN sys.schemas t3 ON (t2.[schema_id] = t3.[schema_id])
WHERE t2.[type] <> 'S' AND t2.[type] <> 'IT'
AND t2.name = @Table;



-- Additional index information if requested.
IF (@IndexExtendedInfo = 1)
BEGIN;
        SELECT *
        INTO dbo.#physical_stats
        FROM sys.dm_db_index_physical_stats (DB_ID(), OBJECT_ID(@Table, N'U'),
NULL , NULL, 'LIMITED');

        SELECT
        i.[Index]
        ,CONVERT(varchar(250),
                                  CASE WHEN i.index_id = 1 THEN 'clustered'
ELSE 'nonclustered' END
                                + CASE WHEN i.[ignore_dup_key] <> 0 THEN ',
ignore duplicate keys' ELSE '' END
                                + CASE WHEN i.is_unique <> 0 THEN ', unique'
ELSE '' END
                                + CASE WHEN i.is_hypothetical <> 0 THEN ',
hypothetical' ELSE '' END
                                + CASE WHEN i.is_primary_key <> 0 THEN ',
primary key' ELSE '' END
                                + case when i.is_unique_constraint <> 0 THEN ',
unique key' ELSE '' END
                                + CASE WHEN i.auto_created <> 0 THEN ', auto
create' ELSE '' END
                                + CASE WHEN i.no_recompute <> 0 THEN ', stats
no recompute' ELSE '' END
                                + ' located on ' + d.name ) AS [Description]
        ,i.[Columns]
        ,i.Included_Columns
        ,t.row_count                    AS Table_Row_Count
        ,t.reserved_MB          AS Table_Reserved_MB
        ,t.data_MB                              AS Table_Data_MB
        ,t.index_size_MB        AS Table_Index_Size_MB
        ,s.avg_fragmentation_in_percent
```

```
        ,s.page_count
        ,s.partition_number
        ,i.is_disabled
        ,i.fill_factor
        ,i.is_padded
        ,i.[allow_row_locks]
  ,i.[allow_page_locks]
FROM dbo.#Indexes i
INNER JOIN dbo.#Table t ON i.[Schema] = t.SchemaName AND i.[Table] =
t.[tablename]
INNER JOIN sys.data_spaces d ON d.data_space_id = i.data_space_id
INNER JOIN dbo.#physical_stats s ON i.[object_id] = s.[object_id]  AND
i.index_id = s.index_id
ORDER BY i.[Index];
END


ELSE BEGIN;

SELECT
        i.[Index]
        ,CONVERT(varchar(250),
                                  CASE WHEN i.index_id = 1 THEN 'clustered'
ELSE 'nonclustered' END
                                + CASE WHEN i.[ignore_dup_key] <> 0 THEN ',
ignore duplicate keys' ELSE '' END
                                + CASE WHEN i.is_unique <> 0 THEN ', unique'
ELSE '' END
                                + CASE WHEN i.is_hypothetical <> 0 THEN ',
hypothetical' ELSE '' END
                                + CASE WHEN i.is_primary_key <> 0 THEN ',
primary key' ELSE '' END
                                + case when i.is_unique_constraint <> 0 THEN ',
unique key' ELSE '' END
                                + CASE WHEN i.auto_created <> 0 THEN ', auto
create' ELSE '' END
                                + CASE WHEN i.no_recompute <> 0 THEN ', stats
no recompute' ELSE '' END
                                + ' located on ' + d.name ) AS [Description]
        ,i.[Columns]
        ,i.Included_Columns
        ,t.row_count                    AS Table_Row_Count
        ,t.reserved_MB          AS Table_Reserved_MB
        ,t.data_MB                              AS Table_Data_MB
        ,t.index_size_MB        AS Table_Index_Size_MB
--,t.unused_MB                  AS Table_Unused_MB
        ,i.is_disabled
        ,i.fill_factor
        ,i.is_padded
        ,i.[allow_row_locks]
  ,i.[allow_page_locks]
--,i.Num_Indexes
FROM dbo.#Indexes i
INNER JOIN dbo.#Table t ON i.[Schema] = t.SchemaName AND i.[Table] =
t.[tablename]
INNER JOIN sys.data_spaces d ON d.data_space_id = i.data_space_id
ORDER BY i.[Index];
```

```
END;



-- Columns information
IF (@ColumnsInfo = 1)
BEGIN;
        EXEC sp_columns @table_name = @Table;
END;


RETURN 0;
GO



USE MASTER; EXEC sp_ms_marksystemobject 'sp_helpindex2';
GO
```