**An Automated Process to Archive Big Tables**

**By** [**Leo Peysakhovich**](http://www.sqlservercentral.com/Authors/Articles/Leo_Peysakhovich/86759/)**, 2013/07/10**

**FROM:** [**http://www.sqlservercentral.com/articles/archiving/100196/**](http://www.sqlservercentral.com/articles/archiving/100196/)

Many databases have large tables with hundreds of millions of rows. However, many of these tables are simply keeping log, or history, data that can be archived and kept outside the user database in a special archive database. Many DBAs move data during a regular maintenance window that can happen quarterly, every half a year, or even once a year. There are plenty of articles on the Internet discussing this scenario and correctly recommending archiving these tables into chunks of data. In this article I am not going to reinvent the wheel, but rather look at a real-life scenario and provide step-by-step coding to explain how to establish the scheduled automated archive process for a large table when the table has not been archived for years.

The company for which I am working has such tables, and in some databases those tables’ sizes are about 40% of the whole database. We have multiple clients and every client has such tables in the client specific database. When I started to analyze the client’s historical data, I realized that there were no existing archive processes and the data was not archived, neither automatically nor manually. Some data goes back up to 5 years, resulting in a substantial increase of the database size. We worked with clients and discovered that they do not need this all this data for their immediate needs; however, they would like to keep the last week’s data in the online database. On the other side, my company would also like to keep log/historical data for compliance and auditing purposes. The DBA group came up with a set of requirements for log/historical data to combine the needs of our clients and our company. Those are:

1. Reduce the size of the backups  and the time to run backup processes.
2. Log/historical tables are populated by means of executing certain stored procedures. This means that table locking should be minimized during data archiving.
3. Internal users occasionally need to be able to verify data during regular working hours and rollback some data up to the previous day.  They do not need to see the current day’s data.
4. Data archiving processes should cause appreciable growth of the database log file.
5. Clients should have the full week of historical data. The average daily historical data for one client grows to about 100,000 rows, and we have 40+ clients with 5 to 10 such tables.

In reality, there are essentially 2 separate tasks: the initial data archiving and the process to archive data regularly by keeping only X number of days in the client database. Let’s talk about both tasks separately.

**Initial data archiving.**

This is a manual task which should be done only once to archive and remove data from client databases. We are not going to discuss the option to insert data into an archive table with one select statement and one delete statement. This option, while possible, will lock the table for a long time and grow the log file with a delete statement.

However, there are some potential problems. I have outlined a few of these problems below, but the reader should be aware that these problems are not limited to the following:

* No matter what the database recovery mode, all data modifications are always logged. There is no way in SQL Server to make any form of data change without it being logged. This means that inserting and deleting millions of rows through one transaction will grow the transaction log.
* In our case, application(s) access to the table will be blocked by the table lock placed by the archiving process.
* If, for any reason, the query stops or dies, then this can delay the process even more as the server has to rollback a huge transaction. Depending on how far along the operation is, this could add on even more time to what was originally planned.

With these setbacks in mind, a different strategy for the initial data archiving has been developed. This strategy includes 7 general steps and was used for all huge tables in the client’s database before standard daily or weekly processes of archiving data are able to run on a regular basis.

After researching the table’s structure, I realized that historical tables have Date\_Inserted field as the first column in the index. The index itself consists of 5-6 columns. In reality, only Date\_Inserted and one more field are being used by the client when they need to verify data. As I mentioned before, verification is seldom required.

I decided to develop the following logic to offload the initial data. At the same time, I would make preparations for the data archive process, which will be created later because of the set of requirements described in this article above. The list of steps for the task 1 called “***Initial Data Archiving”*** is below.

Task 1 is to create the archive structures with this process

1. Create an archive database (this is a onetime task). Let’s call the database [Archive].
2. Create the archive table(s) with the same structure as in the client’s database, adding some extra fields: DBname, ArchiveID bigint identity column, ArchiveDate as the date when the record was archived.
3. Create a control table to monitor the progress of the process and the ability to easily verify that the number of archived and original records is the same.
4. Rename original table with a new name: <tablename>\_archived. For example, [UserObjectLog] table will be renamed to [UserObjectLog\_Archived].
5. Create a new table with all the columns being the same as in the original table with the additional identity column as the primary key.
6. Load X days of data from the original table.
7. Archive data from the original table to the [Archive] database by running a script in chunks of rows. Verify the result. Below is the code for one table in the database.
8. Drop the renamed original table

Task 2 is to create a job and apply the process to archive data regularly by keeping only X number of days in the client database table.

Each task has a set of scripts for one client’s database, which exemplifies the basics for clarity and simplicity. Then these scripts are modified, building off of the basics of the single client’s database, to establish something more complex that can run for multiple clients as generic code. **This code represents the real process.**

Let’s also assume that the [Archive] database is in simple mode. This database was placed in a daily differential and weekly full backup schedule. Usually, the database backup is scheduled to run after the daily archive process is finished.

Let's examine each preparation step for the tasks and the archiving process technique.

**Create Archive Tables:** Create archive table(s) in archive database with the same structure as in the client database with the few additional fields:

* DBname – the database the record came from
* ArchiveID bigint identity column
* ArchiveDate – date when record was archived.

The DBname field helps distingush the record's parent database and gives the ability to separate records for each client. The ArchiveID will be a primary key and a clustered index. In this case it will prevent page splits caused by INSERT statement because, the new row is always appended to the end of the page and when page is filled it will automatically shift to another page. Let’s assume that the original table structure presented below.

CREATE TABLE dbo.Log\_Table1

( user\_id int NOT NULL,

date\_changed datetime NULL,

column1 varchar(255) NULL,

column2 bit NULL,

column3 smalldatetime NOT NULL,

Date\_Created datetime

);

The archive table structure in the Archive database is created based on the modified client’s log table. We are going to modify the original table and add an identity column as the primary key. With archive process explanations later in this article and by analyzing the coding technique, you will see the proof that an identity column is suitable to act as a control mechanism. This provides the best performance for the row deletion during the archive process. The structure of the archive table is below:

CREATE TABLE dbo.ARCH\_Log\_Table1

( ArchiveID bigint identity(1,1),

OriginalLog\_TableID int ,

user\_id int NOT NULL,

date\_changed datetime NULL,

column1 varchar(255) NULL,

column2 bit NULL,

column3 smalldatetime NOT NULL,

Date\_Created datetime,

ArchiveDate datetime,

DBname varchar(20) not NULL,

constraint PK\_ARCH\_Log\_Table1 primary key (ArchiveID)

) ;

**Create the Control Table:** Create the control table in the Archive database to monitor the progress of the initial step of the process. This also gives the ability to easily verify the number of records in the archived and original tables. The number of archived records should match between those two tables. Remember that the number of rows in a log table is huge. The control table will be used only to help with archiving process the very first time.

create table TEMP\_ArchiveFirstTime

( dbname varchar(20) ,

tablename varchar(100),

rowsintable int,

min\_date\_changed datetime,

max\_date\_changed datetime,

status char(1) not null,

StartDate datetime,

EndDate datetime,

constraint PK\_TEMP\_ArchiveFirstTime primary key (dbname,tablename)

);

The Column descriptions for the control table:

* Dbname – database where the log tables reside
* Tablename – the actual table name for the archiving process;
* Rowsintable – number of rows in a log table
* Min\_date\_changed and max\_date\_changed – oldest and latest change date from the log table. Those fields in control table allows easily control the initial process boundaries.
* Status – allows control of  the initial process flow.

Here is the code to insert new data:

insert into TEMP\_ArchiveFirstTime

( dbname,

tablename,

rowsintable,

min\_date\_changed,

max\_date\_changed ,status

)

select name ,'Log\_Table1',null, null, null,'I'

from master..sysdatabases sd

where sd.name like 'Client%'

order by name;

Status 'I' means that table is “Initialized” and ready to be archived.

**Rename the original table**

We first rename original log table with new name as old table name plus “\_archive” (e.g.  <tablename>\_archived). This allows archive data without affecting users and deleting archived rows.  At the end, this table can be simply dropped. At the same time we avoid using an alter table to add new column(s). To add a column with a default value can be costly and time consuming for a table with a lot of rows. An example call:

EXEC sp\_rename 'dbo.Log\_Table1', 'Log\_Table1\_Archived';

**Create the new table**

We create the new table with the same set of columns as in the original table but with an additional identity column as a primary key.

CREATE TABLE dbo.Log\_Table1

( Table1ID bigint identity(1,1) primary key,

user\_id int NOT NULL,

date\_changed datetime NULL,

column1 varchar(255) NULL,

column2 bit NULL,

column3 smalldatetime NOT NULL,

Date\_Created datetime

);

**Load X days of data**

The next step is to load X days of data from the original table. The table archive.dbo.ARCH\_Control is a metadata table defining the set of tables for the archiving process. The code to do this is:

declare @daysback smallint;

select @daysback = KeepDaysBack

from dbo.ARCH\_Control

where tablename = @Log\_Table1

and tableschema = 'dbo';

set @daysback = 0-@daysback;

INSERT INTO dbo.Log\_Table1

(user\_id ,date\_changed,column1 ,column2 ,column3 ,Date\_Created )

SELECT user\_id ,date\_changed,column1 ,column2 ,column3 ,Date\_Created

FROM dbo.Log\_Table1\_Archived

where Date\_Created >= DATEDIFF(DD,@daysback,GETDATE());

**Archive the data from the original table**

We next need to archive data from the original table by running the script in chunks of rows. Verify the result of the execution. Below is an example of the code for one table in one database.

declare @dbnm varchar(20), @tablename varchar(100), @cnt int;

declare @mindt1 datetime, @maxdt1 datetime, @mindate datetime, @maxdate datetime;

select @dbnm = 'Client1', @tablename = 'Log\_Table1';

select @mindt1 = min(date\_changed) , @maxdate = max(date\_changed) , @cnt = COUNT(\*)

from client1.dbo.Log\_Table1;

set @maxdt1 = Dateadd(dd,20, @mindt1);

update TEMP\_ArchiveFirstTime

set rowsintable = @cnt, min\_date\_changed = @mindt1,

max\_date\_changed = @maxdate,

status = 'R',StartDate = GETDATE()

where dbname = @dbnm and tablename = @tablename;

while @mindt1 <= @maxdate

begin

insert into Archive.dbo.ARCH\_Log\_Table1(

OriginalLog\_Table1ID, user\_id, date\_changed,

column1, column2,column3,Date\_Created,DBname )

select

0, user\_id, date\_changed,

column1, column2, column3,GETDATE(),@dbnm

from client1.dbo.Log\_Table1\_Archived

where Log\_Table1ID between @mindt1 and @maxdt1;

set @mindt1 = Dateadd(MILLISECOND,3, @maxdt1);

set @maxdt1 = Dateadd(dd,20,@maxdt1);

checkpoint;

end;

update Archive.dbo.ARCH\_Log\_Table

set OriginalLog\_Table1ID = (0 - ARCH\_Log\_Table1ID)

update TEMP\_ArchiveFirstTime

set status = 'S',EndDate = GETDATE()

where dbname = @dbnm and tablename = @tablename;

go

create unique index IN\_ARCH\_Log\_Table1 on ARCH\_Log\_Table1(DBname,OriginalLog\_Table1ID);

go

**Drop the renamed original table**.

Next we need to drop the table we renamed.

drop table dbo.Log\_Table1\_Archived;

To make these scripts work for the multiple clients, we have to wrap them up with more complicated code. This code will be working for all clients’ databases. Below is the code for this step, which is the most complicated part of the initial data archiving. Remember, there are multiple ways to achieve the task and as example, I provided only one option. Similar code was used in a real live archiving process. In our case it produced a great result on the tables with 150,000,000+ rows.

USE Archive

GO

IF EXISTS (SELECT \* FROM sys.indexes WHERE object\_id = OBJECT\_ID(N'dbo.ARCH\_Log\_Table1') AND name = N'IN\_ARCH\_Log\_Table1\_20')

DROP INDEX IN\_ARCH\_Log\_Table1\_20 ON dbo.ARCH\_Log\_Table1 WITH ( ONLINE = OFF )

GO

declare @tablename varchar(100) = 'Log\_Table1';

declare @cmd varchar(8000), @minid int, @maxid int, @dbnm varchar(20), @rows int;

declare @dbs table (tid int identity(1,1) primary key, name varchar(20) );

declare @mindt1 datetime, @maxdt1 datetime, @mindate datetime, @maxdate datetime;

declare @maxrows table

(rowsintable int, dbname varchar(20) primary key, min\_date\_changed datetime, max\_date\_changed datetime);

set @minid = 0;

set @maxid = -1;

insert into @dbs (name)

select name

from master..sysdatabases sd

inner join Archive.dbo.TEMP\_ArchiveFirstTime a

on a.dbname = sd.name

where sd.name like 'Client%'

and a.tablename = @tablename

and a.status ='I' -- insert

order by name;

select @minid = min(tid), @maxid = MAX(tid) from @dbs;

while (@minid <= @maxid)

begin

select @dbnm = name

from @dbs

where tid = @minid;

update Archive.dbo.TEMP\_ArchiveFirstTime

set StartDate = GETDATE(),

Status = 'R'

where dbname = @dbnm and tablename = @tablename;

set @cmd = 'select count(\*), ''' + @dbnm + ''', min(date\_changed) , max(date\_changed) from '+ @dbnm + '.dbo.Log\_Table1\_Archive ';

insert into @maxrows (rowsintable, dbname, min\_date\_changed, max\_date\_changed)

exec (@cmd);

set @mindate = '1/1/1950';

set @maxdate = '1/1/1900';

select @rows = rowsintable,

@mindate = min\_date\_changed,

@maxdate = max\_date\_changed

from @maxrows

where dbname = @dbnm;

set @mindt1 = @mindate;

set @maxdt1 = Dateadd(dd,20, @mindate);

while @mindt1 <= @maxdate

begin

set @cmd = '

select

0,

user\_id,

date\_changed,

column1,

column2,

column3,

GETDATE(),

''' + @dbnm + '''

from ' + @dbnm + '.dbo.Log\_Table1\_Archive

where date\_changed between ''' + Convert( varchar(30),@mindt1,109 ) + ''' and ''' + Convert( varchar(30),@maxdt1,109 ) + ''';';

insert into Archive.dbo.ARCH\_Log\_Table1(

OriginalLog\_Table1ID,

user\_id,

date\_changed,

column1,

column2,

column3,

Date\_Created,

DBname )

exec (@cmd);

set @mindt1 = Dateadd(MILLISECOND,3, @maxdt1);

set @maxdt1 = Dateadd(dd,20,@maxdt1);

checkpoint;

end;

update ta

set rowsintable = mr.rowsintable,

min\_date\_changed = mr.min\_date\_changed,

max\_date\_changed = mr.max\_date\_changed,

status = 'S',

EndDate = getdate()

from Archive.dbo.TEMP\_ArchiveFirstTime ta

inner join @maxrows mr

on mr.dbname = ta.dbname and ta.tablename = @tablename

where ta.dbname = @dbnm ;

select @minid = @minid + 1;

end;

update Archive.dbo.ARCH\_Log\_Table1

set OriginalLog\_Table1ID = (0 - ARCH\_Log\_Table1ID);

go

use Archive

go

create unique index IN\_ARCH\_Log\_Table1\_20 on ARCH\_Log\_Table1(DBName,OriginalLog\_Table1ID);

go

The initial data is loaded from the old structured tables. Those tables do not have an identity column. I updated column the OriginalLog\_Table1ID with negative numbers from OriginalLog\_Table1ID = (0 - ARCH\_Log\_Table1ID). This allows us to mark the originally loaded records. Another option could be to update the records with positive numbers and set the identity column seed  for the table in client database, starting with the maximum number from the archived table.

This update is very important because it allows to avoid loading the same records the second time during the regular archiving process.  The records in the client’s database will be deleted if record is older than  X days. This step has a “checkpoint” command to run after each chunk of processed data. Based on the Microsoft’s definition:

*“For performance reasons, the Database Engine performs modifications to database pages in memory—in the buffer cache—and does not write these pages to disk after every change. Rather, the Database Engine periodically issues a checkpoint on each database. A checkpoint writes the current in-memory modified pages (known as dirty pages) and transaction log information from memory to disk and, also, records information about the transaction log.”*

When the script is finished for a one table we should verify that the number of archived rows is matching with the number of rows in the original table. Once it is done the original table can be dropped.  The script to compare the number of rows between tables is below:

;with period (dbname, cnt)

as (

select dbname, COUNT(\*)

from dbo.ARCH\_Log\_Table1 with (nolock)

group by dbname

)

select t.dbname, t.tablename,t.rowsintable - s.cnt

from dbo.TEMP\_ArchiveFirstTime t

inner join period s

on t.dbname = s.dbname and t.tablename = 'Log\_Table1'

where t.rowsintable - s.cnt <> 0

order by t.dbname, t.tablename;

And finally, the script to produce the “drop table” statements for all clients.

select 'drop table ' + name + '.dbo.Log\_Table1\_Archived;'

from master..sysdatabases

where name like 'client%'

order by name;

***Task 2. Automated Process of data archiving***

Process code is developed based on the fact that a client is getting only about 500,000 rows in one log table per day. It should work fine even if the number of produced rows per day is larger. The limit for this code will be reached when this code has to be modified to run in a loop and to archive the data by broking it down into chunks. Even then, with some adjustments, the process methodology can be utilized as well.

Let’s assume that an automated archiving process is going to be scheduled to run daily. We do not need a complex mechanism for the regular daily archive processes. However, the process does need a control mechanism to make sure that it is not loading the same data rows multiple times. The identity column is a very convenient technique to act as a control mechanism. Each client’s database has a built in archive stored procedure with 3 step logic:

1. Define the maximum ID from the archive database for the log table
2. Insert all records with IDs higher than the maximum archived ID in the archive table
3. Delete all records from the log table in the client database with data older than X days. Since archives and deletions are done daily, we are going to archive/delete only a daily loaded number of rows. In our case it is close to 500,000 rows per table/transaction. The Deletion/insertion by the clustered one column primary key is the fastest statement in most cases and typically it takes only a few seconds to insert/delete a number of rows. Even if this job is going to be disabled for a day or two for some reasons, the established control mechanism is going to pick up all non-archived records by the last identity id. Column Date\_Inserted is going to help to define the IDs for deletion from the client’s database table. The process’s stored procedure is not required any transactions because the control mechanism allows for repetition of the archiving processes in case of failure.

In the process, the Delete and Insert phases are acting independently. If the archive statement is failed then the deletion statement is not executed. In case of failure for the Insert statement the control mechanism is picking up the same rows during the next run. For simplicity, the stored procedures below do not have an error handler mechanism built-in. In production runs, an error handler should always be added not only to all stored procedures but also to the offload data script presented in this article above.

The code for the one client’s table is posted below.  The next step is to create the wrapper. This wrapper is going to have a loop to serve multiple clients’ databases.  Control table ARCH\_Control was created to define the tables in clients’ databases for the archiving process. It controls the clients’ tables based on the status. Wrapper is a stored procedure from which a set of stored procedures are invoked. Each called stored procedure is developed for one table. To simplify the wrapper the name of the stored procedure can be hard coded as “uspArchive\_<tablename>”.

In our case, the process is more flexible. It keeps the name of the stored procedure in an additional column called StoredProcedureName in table ARCH\_Control  where the name and the schema of the stored procedure are kept. The stored procedure is going to be used by the process to archive a specific table.

CREATE TABLE archive.dbo.ARCH\_Control

( ControlID int NOT NULL primary key,

DatabaseName varchar(20) NOT NULL,

TableName varchar(100) not null,

TableSchema varchar(10),

KeepDaysBack smallint not null,

Status char(1) not null,

StoredProcedureName varchar(100)

);

INSERT INTO archive.dbo.ARCH\_Control

(ControlID ,DatabaseName ,TableName ,TableSchema ,KeepDaysBack,Status ,StoredProcedureName)

VALUES (1 ,'Client1' ,'LOG\_Table1' ,'dbo' ,7 ,'A', ‘uspArchive\_Log\_Table1’);

The stored procedure, uspArchive\_Log\_Table1, is compiled in each client database. The parameter, @keepdaysback, defines for how many days the records are kept in the log table.

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CREATE procedure dbo.uspArchive\_Log\_Table1

@keepdaysback smallint

as

begin

SET NOCOUNT ON

declare @id int;

create table #tid (id int primary key);

-- check just in case negative number is passed to protect data

IF (@keepdaysback > 0 )

set @keepdaysback = 0 - @keepdaysback;

-- get max processed id from archive table

select @id = MAX(OriginalLog\_Table1ID)

from Archive.dbo.ARCH\_Log\_Table1 with (nolock)

where DBname = DB\_NAME();

set @id = ISNULL(@id,0);

insert into Archive.dbo.ARCH\_Log\_Table1(

OriginalLog\_Table1ID,

user\_id, date\_changed,

column1,

column2,

column3,

Date\_Created,

CustomerCode)

select

Log\_Table1ID ,

user\_id,

date\_changed,

column1,

column2,

column3,

GETDATE(),

DB\_NAME()

from dbo.Log\_Table1

where Log\_Table1ID > @id;

-- because deletion will be done daily or weekly it should not be many records in the table

insert into #tid (id)

select Log\_Table1ID

from dbo.Log\_Table1 with (nolock)

where date\_changed <= DATEADD(dd,@keepdaysback,getdate() );

delete lt

from dbo.Log\_Table1 lt

join #tid t

on t.id = lt.Log\_Table1ID;

return;

end;

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Finally, the wrapper procedure is going to be used in the archiving job step. This procedure is based on the control table [Archive].dbo.ARCH\_Control.

use archive

go

CREATE procedure dbo.uspArchiveLogTablesWrapper

as

begin

SET NOCOUNT ON

declare @tablename varchar(100), @TableSchema varchar(10), @keepdaysback smallint, @DatabaseName varchar(60);

declare @minid smallint, @maxid smallint, @execSQL varchar(1000), @StoredProcedureName varchar(100);

declare @tbl table

( DatabaseName varchar(60)

, TableName varchar(100)

, TableSchema varchar(10)

, KeepDaysBack smallint

, StoredProcedureName varchar(100)

, tid int identity(1,1) primary key

);

set @minid = 0;

set @maxid = -1;

-- for all DBs all tables to cleanup

insert into @tbl ( DatabaseName, TableName , TableSchema , KeepDaysBack, StoredProcedureName)

select DatabaseName, TableName ,TableSchema

, KeepDaysBack, StoredProcedureName

from dbo.ARCH\_Control

where Status = 'A';

select @minid = MIN(tid), @maxid = MAX(tid)

from @tbl;

while (@minid <= @maxid)

begin

select

@tablename = TableName,

@DatabaseName = DatabaseName,

@KeepDaysBack = KeepDaysBack,

@StoredProcedureName = StoredProcedureName from @tbl

where tid = @minid;

set @execSQL = 'exec ' + @DatabaseName + '.dbo.' + @StoredProcedureName + ' @keepdaysback = ' + CAST (@KeepDaysBack as varchar) + ';';

exec ( @execSQL );

set @minid = @minid + 1;

end;

return;

end;

**Conclusion**

This article presents only one of the many ideas on how to establish a regular daily or weekly archiving processes for the tables with large number of rows located across the multiple databases and based on the fact that tables were not archived before. It explains a multistep process and some initial data archiving caveats. For clarity, some pieces of code are simplified and the error handlers were removed. Similar code base was used for multiple big tables to establish automated archiving processes.