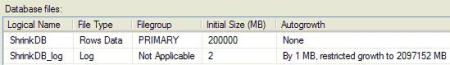
**Testing Transaction log auto growth behavior in SQL Server**

[shrinking database files](http://searchsqlserver.techtarget.com/tip/0,289483,sid87_gci1330922,00.html), we tested and proved the performance implications when a database file grows during a transaction. In the case of transaction logs, there are also potential performance issues.

Transaction logs tend to grow more often than database files, and as a result, database administrators shrink them more often. In this tip, I will test the impact when a transaction log file grows during a transaction. In this article's example, I use the [same environment and configuration](http://searchsqlserver.techtarget.com/tip/0,289483,sid87_gci1330922,00.html#CONFIG) as that described in my previous tip; the only difference is that I have installed SP3 on the SQL Server.

**The database**

My test database, ShrinkDB, is configured not to allow the database file to grow during the transaction. The transaction log file size, however, is only 2 MB, and the goal of these tests is to let it grow with each transaction to measure its implications:



The recovery model of the ShrinkDB database is set to FULL.

select size fromsysfiles where fileid = 2

This above query, in which fileid equals the transaction log file, returns 256. This indicates that the size of the transaction log file, in 8 KB pages, is 2 MB.

As in my previous article, the [ExpandDB table](http://searchsqlserver.techtarget.com/tip/0,289483,sid87_gci1330922,00.html#TABLE) is used yet again:

create table ExpandDB (a varchar(8000))

**The tests**

My goal was to learn how the INSERT, UPDATE and DELETE commands affect the growth of the transaction log, along with how the growth of the transaction log affects the INSERT, UPDATE and DELETE transactions.

I divided the tests into two phases:

http://media.techtarget.com/searchSQLServer/images/homepageFloater_arrow.gif In the first phase, I monitored the general behavior of the transaction log's growth relative to the          running modifications.

http://media.techtarget.com/searchSQLServer/images/homepageFloater_arrow.gif In the second phase, I used only those commands that in the first phase caused the transaction            log to grow. In this phase, I've tested autogrowth scenarios similar to those used in part one.

Since updates, inserts and deletes can behave differently, each phase includes three sections, one per operation: UPDATES, INSERTS and DELETE.

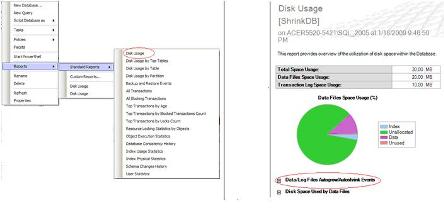
**Phase 1: Transaction log growth behavior**

**UPDATE commands and transaction log growth**  
In this test, my code inserts only one row to the table, and it will update this row until the transaction log file grows to 8 MB (1,024 pages):

 -- Insert one row into the table  
insert into ExpandDB select replicate ( 'a',8000)  
--================================================================  
 -- Update the table until T-Log reaches 8MB  
while (select size from sysfiles where fileid = 2) <= 1024  
   update ExpandDB set a = replicate ( 'a',8000)  
go

**The results were surprising!**

My loop ran indefinitely, the transaction log file never grew and the percentage of free space in the transaction log, which was captured using dbcc sqlperf(logspace), was consistently greater than 30%. During the transaction, the transaction log would fill up until it reached 70% and then decrease to about 50%. To make sure I got the right data from dbcc sqlperf, I also ran the report to look for autogrowth in database files:

[](http://media.techtarget.com/digitalguide/images/Misc/sql_shrinkDB_figure_2.JPG)

It reported no autogrowth for any database at that time!

I then decided to check a large transaction with short updates of one row at a time, so I added a begin transaction:

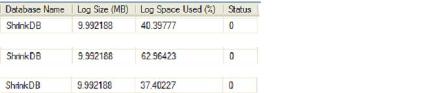
 -- Big transaction  
begin tran  
while (select size from sysfiles where fileid = 2) <= 1024  
    update ExpandDB set a = replicate ('a',8000)  
go  
commit tran

This time, the results were equally confusing. My loop ran indefinitely, the transaction log file never grew beyond the initial 2 MB and the percentage of free space in the transaction log was constantly greater than 60%. I canceled the loop and committed the transaction. After the commit, the used space in the transaction log file even decreased a bit in size.

In another test of the UPDATE command, I inserted 10,000 rows into the table and ran the following code:

  --  Truncate the T-Log  
backup transaction ShrinkDB with truncate\_only  
go  
  
  --   Check the size of the T-Log file (RESULT = 2MB)  
select size from sysfiles where fileid = 2  
go  
  
  --  Check % free space in T-Log  
dbcc sqlperf(logspace)  
go  
  
  --  Update 10000 rows at a time  
update ExpandDB set a = a + 'a'  
go  
update ExpandDB set a = a + 'a'  
go  
update ExpandDB set a = a + 'a'  
go  
update ExpandDB set a = a + 'a'  
go  
update ExpandDB set a = a + 'a'  
go  
update ExpandDB set a = a + 'a'  
go  
update ExpandDB set a = a + 'a'  
go  
-- Check % free space in T-Log  
dbcc sqlperf(logspace)  
go

The results were similar. The transaction log file did grow by 1 MB until it reached 10 MB, but after reaching this size, it stayed put. No matter how many times I executed the same update, the space filled up and reduced again:



I searched for answers regarding this issue and found none in [SQL Server 2005](http://msdn.microsoft.com/en-us/sqlserver/bb671246.aspx) documentation. I also performed this test on another machine with SQL Server 2005 SP3, and I encountered the same behavior.

I analyzed the contents of the transaction log with the [ApexSQL Log](http://www.apexsql.com/downloads_p.asp#log) tool and the results confirmed that every time I looked at the content of the transaction log, I got a different number of updates. This is the same behavior I encountered with the transaction log sizes; it grew and shrank, but the file size did not change.

In another attempt to understand the behavior of the transaction log, I executed a transaction where the UPDATE command modified the values in the row:

  --  Truncate the table  
truncate table ExpandDB  
go  
  --  Truncate the T-Log  
backup transaction ShrinkDB with truncate\_only  
go  
  --  Shrink T-Log back to 2MB:  
DBCC SHRINKFILE (N'ShribkDB\_Log' , 0, TRUNCATEONLY)  
Go  
  --  Insert one row into the table  
insert into ExpandDB select replicate ('a',3)  
--================================================================  
  --  Update the table until T-Log reaches 8MB  
  --  Big transaction  
begin tran  
while (select size from sysfiles

**Show Me More**

Top of Form

where fileid = 2) <= 1024  
      update ExpandDB set a = case len(a)  
        when 7999 then a else a + 'b' end  
commit tran  
go  
  --  Check status of T-log  
dbcc sqlperf(logspace)  
go

In this case, the transaction log grew to 4 MB, and the amount of space used remained around 94% for a long time. I did not, however, get an error indicating string truncation or overflow. As monitored by the ApexSQL Log tool, the transaction log contents stayed constant at around 13,380 rows when the transaction log stopped growing. Furthermore, the [monitoring statement in SQL Profiler](http://msdn.microsoft.com/en-us/library/aa173918(SQL.80).aspx) also showed that no command was being executed after the transaction log stopped growing, and no errors appeared. After I canceled the transaction, no special message appeared in the error log to indicate that an autogrowth operation was canceled or aborted, but the transaction appeared to be stuck.

In similar instances, I sometimes see the following message in the error log:

*"Autogrow of file '<DB logical file name' in database '<Database name>' was canceled by user or timed out after 469 milliseconds. Use ALTER DATABASE to set a smaller FILEGROWTH value for this file or to explicitly set a new file size."*

But no matter how many times I executed the above code, this did not happen during the test.

After canceling the transaction, the space used in the transaction log was reduced to around 45%.

I decided to run the above UPDATE command without a transaction on a larger amount of rows:

  --  Truncate the T-Log  
backup transaction ShrinkDB with truncate\_only  
go  
  
  --  Check the size of the T-Log file (RESULT = 2MB)  
select size from sysfiles where fileid = 2  
go  
  
  --  Check % free space in T-Log  
dbcc sqlperf(logspace)  
go  
  --  Update 10000 rows at a time  
update ExpandDB set a case len(a)  
        when 7999 then a else a 'b' end  
go  
update ExpandDB set a case len(a)  
        when 7999 then a else a 'b' end  
go  
update ExpandDB set a case len(a)  
        when 7999 then a else a 'b' end  
go  
update ExpandDB seta caselen(a)  
        when 7999 then a else a 'b' end  
go  
update ExpandDB set a case len(a)  
        when 7999 then a else a 'b' end  
go  
update ExpandDB set a case len(a)  
        when 7999 then a else a 'b' end  
go  
update ExpandDB set a case len(a)  
        when 7999 then a else a 'b' end  
go  
  --  Check % free space in T-Log  
dbcc sqlperf(logspace)  
go

The results showed that if I executed the updates one by one, waiting about one or two seconds in between, the transaction log did not grow and space filled up and emptied as before. When I executed two updates in a row, however, it seemed that the space did not fill up or empty before the second transaction ran. In this scenario, the transaction log grew.

***INSERT commands and transaction log growth***  
In this test, my goal was to make the transaction log grow to 8 MB and then compare the data file growth rate with the transaction log growth rate. The data file was set to 200 GB with no autogrowth, and the transaction log was set at 2 MB with 1 MB of autogrowth. I executed the following code:

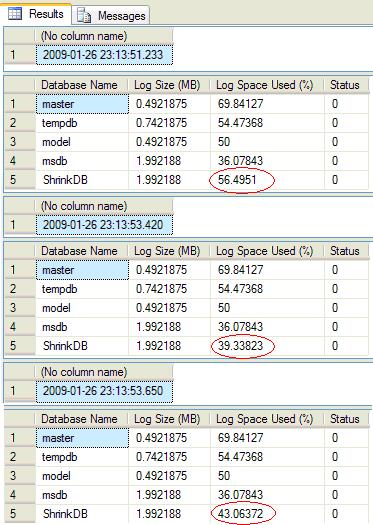
  --  Truncate the table  
truncate table ExpandDB  
go  
 --  Truncate the T-Log  
backup transaction ShrinkDB with truncate\_only  
go  
  --  Shrink T-Log back to 2MB:  
DBCC SHRINKFILE (N'ShrinkDB\_Log' , 0, TRUNCATEONLY)  
Go  
  --  Check the size of the T-Log file  
select size from sysfiles where fileid go  
  --  Insert narrow rows into the table  
while (select size from sysfiles where fileid = 2) insert into ExpandDB select replicate ('a',3)  
go

At the same time, I also executed the following code:

select getdatedbcc sqlperf (logspace)  
go  
waitfor delay '00:00:02'  
go  
select getdatedbcc sqlperf (logspace)  
go  
select getdatego  
dbcc sqlperf (logspace)  
go  
waitfor delay '00:00:02'

***Go***

My loop ran for a long time, the data file filled up slowly and emptied, but the transaction log file stayed at 2 MB. The above code returned these results:

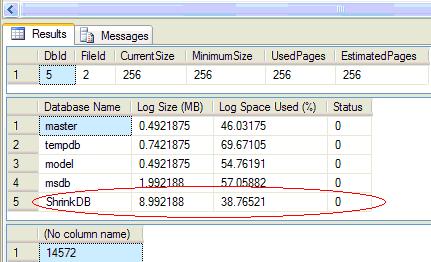


Apparently, the amount of free space decreased. Right before canceling the inserts 45 minutes into the test, I had around 1.8 million rows in the table and the transaction log was still at 2 MB. After the cancellation, the transaction log was still at 2 MB and only 39.88% of the space had been used. The data file was at 31% used.

I then attempted a test using large transactions with one insert at a time:

 -- Truncate the table  
truncate table ExpandDB  
go  
 -- Truncate the T-Log  
backup transaction ShrinkDB with truncate\_only  
go  
 -- Shrink T-Log back to 2MB:  
DBCC SHRINKFILE (N'ShrinkDB\_Log' , 0, TRUNCATEONLY)  
Go  
 -- Insert 100000 rows  
 -- Big transaction  
begin tran  
while (select size from sysfiles where fileid = 2)    insert into ExpandDB select replicate ('a',3)  
go  
commit tran  
go  
dbcc sqlperf(logspace)  
go  
select count(\*) from ExpandDB  
go

The transaction log grew up to 9 MB, and at the end, there were 14,572 rows in the table:



To test large updates, I inserted 10,000 rows in the table and ran the following code:

 -- Truncate the T-Log  
backup transaction ShrinkDB with truncate\_only  
go  
 -- Check the size of the T-Log file (RESULT = 2MB)  
select size from sysfiles where fileid = 2 go  
  
 -- Check % free space in T-Log  
dbcc sqlperf(logspace)  
go  
 -- Insert 10000 rows at a time  
Insert into ExpandDB select \* from ExpandDB  
Go  
 -- Check % free space in T-Log  
dbcc sqlperf(logspace)  
go  
 -- Insert 10000 rows at a time  
Insert into ExpandDB select \* from ExpandDB Go  -- Check % free space in T-Log  
dbcc sqlperf(logspace)  
go  
 -- Insert 10000 rows at a time  
Insert into ExpandDB select \* from ExpandDB  
Go  
 -- Check % free space in T-Log  
dbcc sqlperf(logspace)  
go  
 -- Insert 10000 rows at a time  
Insert into ExpandDB select \* from ExpandDB  
Go  
 -- Check % free space in T-Log  
dbcc sqlperf(logspace)  
go

This time, the transaction log multiplied by 2 after each insert.

***DELETE commands and transaction log growth***  
  
My goal was the same as with the INSERT and UPDATE commands. I inserted 10,000 rows into the table and ran the following code to delete one row at a time:

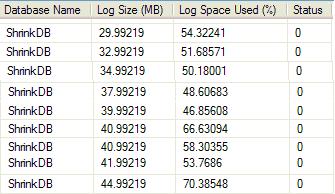
 -- Truncate the table  
truncate table ExpandDB  
go  
 -- Truncate the T-Log  
backup transaction ShrinkDB with truncate\_only  
go  
 -- Shrink T-Log back to 2MB:  
DBCC SHRINKFILE (N'ShrinkDB\_Log' , 0, TRUNCATEONLY)  
Go  
 -- Insert 100000 rows  
declare @i int  
set @i = 1  
while @i <= 10000  
begin  
   insert into ExpandDB select replicate ('a',1000)  
   set @i = @i + 1  
end  
go  
-- Delete rows one by one:  
set rowcount 1  
while (select size from sysfiles where fileid = 2) delete from ExpandDB  
set rowcount 0  
go

The same behavior emerged: The transaction log did not grow, and the space filled up and emptied in a cycle.

I then attempted a large delete:

 -- Truncate the table  
truncate table ExpandDB  
go  
 -- Truncate the T-Log  
backup transaction ShrinkDB with truncate\_only  
go  
 -- Shrink T-Log back to 2MB:  
DBCC SHRINKFILE (N'ShrinkDB\_Log' , 0, TRUNCATEONLY)  
Go  
 -- Insert 100000 rows  
declare @i int  
set @i = 1  
while @i <= 10000  
begin  
   insert into ExpandDB select replicate ('a',1000)  
   set @i = @i + 1  
end  
go  
 -- Delete 10,000 at a time and monitor the T-Log size and free space:  
begin tran  
set rowcount 10000  
while begin  
   delete from ExpandDB  
   dbcc sqlperf(logspace)  
end set rowcount 0 commit tran go

The results indicated that the transaction log grew:



Finally, I ran a large transaction with small operations:

 -- Truncate the table  
truncate table ExpandDB  
go  
 -- Truncate the T-Log  
backup transaction ShrinkDB with truncate\_only  
go  
 -- Shrink T-Log back to 2MB:  
DBCC SHRINKFILE (N'ShrinkDB\_Log' , 0, TRUNCATEONLY)  
Go  
 -- Insert 100000 rows  
declare @i int  
set @i = 1  
while @i <= 10000  
begin  
   insert into ExpandDB select replicate ('a',1000)  
   set @i = @i + 1  
end  
go  
 -- Delete 10,000 at a time and monitor the T-Log size and free space:  
begin tran  
set rowcount 1  
while from ExpandDB) begin    delete from ExpandDB end

It appeared that the transaction log grew, albeit slowly