**http://www.sqlservercentral.com/articles/Design+and+Theory/63350/**
Printed 2008/07/04 06:05PM

# Introduction to the Transaction Log

**By James Rea, 2008/06/05**

## Summary

Transaction Log is probably one of the most underutilized, underappreciated, and least publicized part of SQL. This is probably due to its simplicity, but this also makes it one of the most misunderstood parts of SQL Server. The transaction log can actually help to lower the server's resources, increase productivity, all the while providing a method of an additional safety layer for you to fall back on!

Transaction logs are not just for the big boys (OLTP, OLAP and databases with large amounts of activities/transactions); this is for everyone…even the small shops with only 1 database and 10 users. Before we get into how to use transaction logs to their best abilities, we need to understand why they exist, and how they work.

Lastly, we need to know what are some of the good practices to follow are. Such as how we can maintain the Transaction Log and, and especially to know when we can count on the Transaction Log helping us out of a jam.

## Why the Transaction Log Exists

This history of the existence of the transaction log is difficult to trace; and is to a small degree controversial due to certain aspects are based on other applications. An example is the method to use transaction logs to restore databases is primarily based on proven methods used in Microsoft Access, which in turn is based on other application(s) and standards. This means there is no single document that states the history of Transaction Logs and most current documentation is partially based on personal experience, knowledge, and/or interpretations.

I'll make no attempt to determine the exact history of transaction logs. I just want to point out that transaction logs are largely based on Microsoft Access methods to store information for transactions. This is important to know because it has become a common misunderstanding that using the TRUNCATE command should not only remove old transaction information, but also should SHRINK the file down. This is absolutely, without a doubt…WRONG! I'll get into this more as this article progresses. First, to be able to understand why TRUNCATE will not make the log smaller in physical size you will need to understand how the transactions are inserted into the log; both physically and logically.

The primary purpose the transaction log is to provide a method to be able to restore a database to a point-in-time when necessary. This can include rolling back transactions to a certain time, or to roll forward transactions from a full backup restoration.

In recent years the transaction log is starting to be used for additional purposes. They are being used in mirroring databases, quickly importing data from old database versions to a new database version, and security audits. This is just the tip of the iceberg for the way these logs are being utilized to gain "useful" information. The last point, security audits, is a more relatively new and undocumented (officially, that is) use of transaction logs. Information stored within transaction logs can be very powerful, but also tricky to learn at first while the structure and information being stored is new to you. If you take your time to learn what you can get out of the transaction log, you can produce very useful information that can help to increase your productivity (especially in the Security Audits area)!

An example of how to use these logs for security audits is that you can view a transaction log and be able to detect transactions that have occurred and whom (what user or system account) had produced the transaction (this is no trivial task to accomplish); and if you know your database very well you can quickly deduce if someone is accessing parts of the database they shouldn't be; or even worse, if an unauthorized user has accessed your database! The techniques to use the transaction logs for security audits go beyond the scope of this article; the preceding is intended to provide a broader understanding of methods being used to obtain useful information. You can search sites such as MSDN, SQLServerCentral.com, and Google for discussions and articles detailing many different ways to audit the logs and obtain additional useful information. It is also worthwhile to note that there are 3rd party applications that utilize the transaction logs for security auditing, data movement, and many other purposes. As always, evaluate any 3rd party application when possible; and test on TEST servers before ever using new techniques or applications in a production environment.

# Structure of the Transaction Log

A transaction log is a file (physical and logical) that contains information regarding every change that has been made to the database. This includes data modifications (transactions), database modifications, and backup/restore events.

## Logical Architecture (Concept)

These "transactions" (and other events) are logged in the transaction using a Log Sequence Number (LSN). The LSN for each transaction that is subsequent to the previous logged transaction will have a value that is higher (incremented) than the previous LSN. I like to use the analogy of the Transaction Log file is a journal, and each logged transaction is a journal entry. Try to picture a paper with a line being used for each transaction, and the next obvious entry would be the following line (keep this in mind when working with these logs, it will help you to understand why things work the way they do, I'll elaborate in a little bit on this analogy).

Each transaction reserves enough space to support a successful rollback, either by an explicit action requesting the rollback or from an error occurring within the database. This amount of space can vary; however, it typically mirrors the amount of space that is used to store the logged operation. Transaction logs are loaded into a virtual log; this cannot be controlled or set by the administrator. The virtual log is used and maintained, as needed, by the SQL Server Database Engine.

At the time this article was written you can read the MSDN: Transaction Log Logical Architecture at: http://msdn.microsoft.com/en-us/library/ms180892.aspx

## Physical Architecture (Concept)

Transaction logs can span multiple physical files. You can truncate the transaction log to free internal log space (this will not shrink the log file; I'll cover shrinking the log file later in this article). The basic concept of the physical file is it will append to the end of the log file, once the physical end of the log is reached the transactions will wrap around to the beginning of the log file (assuming there is free space at the beginning).

If the log file does not contain any free space (internally); there are two possible outcomes:

1) If FILEGROWTH is enabled and there is free disk space on the hard drive, the file will automatically increase in accordance to the settings and then will re-organize the transaction log and append the new

record to the end of the log, OR

2) If FILEGROWTH is NOT enabled, or there is NOT enough free disk space on the hard drive, then SQL Server will return a 9002 error code.

At the time this article was written you can read the MSDN: Transaction Log Physical Architecture at: http://msdn.microsoft.com/en-us/library/ms179355.aspx

## How the Transaction Log Works

As mentioned in the "Logical Architecture (Concept)" portion, transactions are recorded into the log in a sequential manner. The LSN for each transaction will have a value that is higher than the previously written transaction. It's actually just that simple; no complicated mathematical methods are used. There is no clever programming to add new transactions. It's literally the same as taking a piece of paper and writing in each event that has happened in your day and using an ID number starting with 1 and increasing by 1 for each event occurring to give the order of the sequence. And just like your day, the next event that occurs to you must happen after the previous event; thus you'd give the next event a higher ID number than the previous. It's safe to say that most of the following events have happened in this order for you to read this article:

1) You told your computer to go to the link, or to open a document, containing this article.

2) Your device displayed the article on your display, or printed the article to another format to be viewed.

3) You started reading this article.

As with these events, you can't have the 2nd and 3rd events occur without first event obtaining the article, and you can't read the article unless you view/print it (thus #3 can't occur without #2 occurring first). As you can see, you can't have these steps logged out of order to recreate this process; so each step has a higher ID number than the previous step.

A transaction log is just exactly that in concept, a journal of events that occur within your database. The main points to remember here is that each event is given a LSN that is higher than the previous LSN (to keep track of what event happened in what sequence), and that there is enough information being stored so that SQL Server may recreate the event should a restore of the database be required.

## How to Maintain the Transaction Log

### Truncating the Log

TRUNCATE is the most misunderstood command used with transaction logs. It is quite common to find a conversation within a group of SQL DBAs about how they are using TRUNCATE and don't see their Transaction Logs decreasing in size. This is most common with beginning DBAs; but there are a few seasoned DBAs with this misconception. This is the reason I've created this article! I'm not saying that this should be something everyone automatically understands, I know this is difficult to understand at first; the purpose of the article is to provide a point of reference where someone (such as you) can gain the knowledge and understanding.

If you have a piece of paper and erase some entries from earlier transactions, you wouldn't tear up the paper and start a new one. The most you might do, is to move some entries up a few lines (or down a few lines) to ensure the entries are sequential and no blank area exists. This process on the paper obviously would NOT make the paper any smaller in actual size; all you have done is to make the writing area on the paper easier to read (by grouping the entries together, if there were any blank lines between entries) and freed up space to enter new journal entries (transactions).  To make the paper smaller you'd have to cut or tear off the blank lines; which obviously wouldn't be caused from erasing the old entries. Making the paper smaller involves an entirely different process. This is exactly how the TRUNCATE command works with transaction logs!

At the time of writing this article you can find the exact definition and recommended usage by Microsoft at: http://msdn.microsoft.com/en-us/library/ms189085.aspx

In particular note that under the section labeled “How Log Truncation Works”, the first two sentences state:

“Truncation does not reduce the size of a physical log file. Reducing the physical size of a log file requires shrinking the file.”

## Shrinking the Log

As discussed in the previous section, to shrink the transaction log the process is completely different than truncating the log. You actually will need to use the FILESHRINK command to make the log smaller. But, before you use this command take a moment to think about if you want to do this. There are a few things to keep in mind. Some have already been covered; others are good practices to think about.

First off, think of why did the transaction log get larger? Remember; that a transaction log will increase in size automatically (default behavior) when there is no additional space within transaction log to store the transaction being written to the log. Because SQL Server must keep track of the transactions (to allow for point-in-time restores), the log MUST be increased to accommodate the space required to rollback (or forward) the transaction.

Why does this matter? Why should you take this into consideration? Well, this can mean a few different things. First, how did this transaction being recorded come about? Was it from normal use of the database, or was it from an extraordinary event? If this is caused from normal use, then this will meant that after you shrink the file it will most likely need to increase in size. This is assuming that the next time the database is backed up or the log is truncated will be after the same amount of use (number of transactions) have occurred. So, this also means that when SQL Server needs to increase the file size there are two concerns that come to mind immediately. First concern is if the hard drive has the space available; if you are out of space then this can cause problems. Second, it takes resources to process the increasing of the file size; this will include CPU, RAM, and I/O resources. Using up additional resources can mean lower performance from SQL Server until the increasing of the file has been completed; also this process will be repeated to decrease the file size.

Now, if this log had increased in size due to unusual circumstances then shrinking the file could be optimal for performance and drive space considerations. An example of an unusual circumstance would be during a year end archiving of data; this would mean to export a year's worth of data to an archiving database or backup file, and this means you are “modifying data” within your database. So, this means SQL Server will record the action within your transaction log(s) in case you need to roll the transaction back; and most likely would create a ‘ballooning’ effect of your log. Obviously, in this case it is a yearly occurrence and thus depending on the size of the data being archived can create a large amount of space reserved in the transaction log (after backing up your database or truncating the log) that is unused and won't be used again for a year.  In this case I'd recommend shrinking the log after you've successfully backed up your database, of course to use care and common sense as to when to perform this due to the resources that will be required to shrink the log.

At the time of this writing, you can find more information about shrinking the transaction log on the MSDN website at: http://msdn.microsoft.com/en-us/library/ms178037.aspx

# Analogy anyone can follow (and even perform)

Here is where picturing the journal entries will help the most. Remember that the log nearly duplicates the information being stored in the database. Let's say you are recording each event that occurs within your database onto a piece of college ruled paper. So, you can fit around 34 lines of events (assuming each event can be fit on to a single line). Now, let's say you have around 100 pages filled with events. Now, say you want to archive a few records (to make room for new records), maybe 20 records. So you would record the archive records onto a different piece of paper to be stored in an archive location, and erase the original records. So, now you have freed up some space.

So, now you decide you need to TRUNCATE the paper before adding new records; this is so you can easily find the transactions should you be required to replicate them if something occurs within your database. After all, you don't want data transactions intertwined with old and new; otherwise you have to sort through that to figure out which belongs where if you need to only repeat the new data. The best method to follow here would be to compact all of the lines on a single page into a sequential order. Mark where the old data ends and the new data will begin. After all, you don't want to do this for the entire 100 pages of data…that would take up a lot of time and resources to rewrite all 100 pages for 20 lines of data being removed.

This is exactly how TRUNCATE works with transaction logs; you don't want to re-adjust the entire log because of a few rows were removed. This takes up time and effort (not to mention that if you need to insert 25 lines of new data and only have 20 available, you will have to start another page anyways). In this case it would be best to just use up the 20 free on a single page and then put the remaining 5 onto a new page, and leave the other 99 pages alone. Now, there are circumstances where you will want to shrink the file; such as if you freed up 50 pages and expect to never use more than 30 pages to store the new data. In this case you can gain performance and disk space by removing the 20 pages; but if you know you'll need all of that space again at some point, then you may as well leave that space available otherwise the server will have to recreate that space when the data is being inserted and you won't know when this will be!

# Good Practices to Use with the Transaction Log

## Microsoft Recommendations

The Transaction Log is recommended, by Microsoft, to be located on a mirrored drive in a fault-tolerant storage system. It is also recommended, by many experts, to ensure that the log is located on its own physical hard drive. These two recommendations are based on the idea that should the database (or the hard drive holding the database) become corrupt that you can ensure that you have access to the Transaction Log during your recovery operation. Also, this setup will help to ensure you have minimal impact on your database performance due to the consistent writing to the Transaction Log; obviously to have a hard drive write to the database and then to write to the Transaction Log would require it to physically write twice. If the database and hard drive are on separate drives then each drive can only write once, thus improving your performance.

## Author's Recommendation

I personally recommend that when creating a database to run some test server data if possible; and to try to mimic the amount of data being passed through the database on a regular basis. If you are unsure of the amount of data being passed, or are unable to simulate the proper amount of data, then I'd recommend starting off with a modest amount of space for your transaction log (ranging from 5% to 10% of the database size). The objective is to get a picture of how big the transaction log will need to be from regular usage. Assuming there are not any drastic unusual amounts of data being passed through the system, after about 3 full backups (using your regularly scheduled intervals) with NOT shrinking the log file, you can feel fairly confident in knowing the size the transaction log needs to be.

It is common to use the log size being produced after the database has had enough regular usage data passed through plus another 10 – 20% of the transaction log size for small abnormal transactions and maintenance; such as a small number of records being archived (if archiving is done on a more regular basis, such as monthly or quarterly), additional tables are added later to hold new data requirements, or reports are being created and stored within the database. These are all concerns that are best addressed earlier in the development cycle; however, if the transaction log needs to be increased later that can always be accomplished by the administrator (or by SQL Server if left at defaults).

Keep track of the log file size throughout the life span of the database; if you notice it grow suddenly then look into the cause. The highest concern would be a massive amount of data being modified (either from archiving, or even worse deleting); if either of these are the case then make sure that the transaction log is not truncated. You may find that you will need this to rollback that large transaction, if this occurred during a non-planned time (such as a misuse of the DELETE or DROP commands by a database user). This is where the transaction log can literally save your job!

The last and probably most helpful recommendation I can give is to take a proactive approach to sizing your database's transaction log file. Be generous in the amount of space you reserve for the transaction log. And most importantly if you expect (or even suspect) the log will need to be increased do this as soon as possible during off-peak hours; this will have the least amount of impact on your system and will result in the optimal performance of the system (while avoiding any unexpected performance degrading that might occur during peak hours!); remember that the less resources you utilize means the more resources that can be given to help your queries and reports!

## Conclusion

As you can see from the analogy the Transaction Log is a fairly simple concept to get; yet it has a level of complexity due to the amount of data it will hold to allow you to be able to restore to a specific point in time. Hopefully, you can also understand why TRUNCATE only frees up space for internal use by SQL Server; as well as, the reasons you will typically not want to shrink the transaction log file. As with any rule or concept, there are exceptions. I've pointed out common exceptions that would warrant at least the consideration of shrinking the file.

I've also given a brief introduction to the non-traditional ways of using the transaction log to increase productivity, such as Security Audits. And I've also pointed out how you can control the size of the transaction log to ensure you get the most optimal performance. Remember that as with any other area of SQL Server gaining the optimal performance can be a fine line, and there can be occasions where the optimal settings are no longer optimal (rather it be for an unusual circumstance, or the database requirements have changed).

I'll close with one final recommendation of periodically reviewing the capacity of the transaction log by using the "sys.database_files" view, this syntax will show how much space is currently available within the log (Ref: MSDN – DBCC SQLPERF- http://msdn.microsoft.com/en-us/library/ms189768.aspx):

```
DBCC SQLPERF(LOGSPACE);

GO
```

This can be very helpful in determining when you should proactively increase the log size; if you start to notice the free space level dropping over time and continues to drop even after having backed up the database, this could then be an early indication that there is more data flowing through the database then originally estimated. At this point keep an active watch (potentially daily if getting very low) and determine the cause of the log increasing in size, and the appropriate actions to take (such as increasing log size or stopping excessive transactions).

With all of this information, you are prepared to handle transaction logs with grace and know that you can control how the transaction logs will affect the performance of your system!

## Additional Resources:

TechNet: Working with Transaction Log Backups - http://technet.microsoft.com/en-us/library/ms190440.aspx
TechNet: Restoring a Database to a Point Within a Backup - http://technet.microsoft.com/en-us/library/ms190244.aspx
TechNet: Recovering to a Log Sequence Number (LSN) - http://technet.microsoft.com/en-us/library/ms191459.aspx
TechNet: Best Practices for Recovering a Database to a Specific Recovery Point - http://technet.microsoft.com/en-us/library/ms191468.aspx
TechNet: Security Considerations for Backup and Restore - http://technet.microsoft.com/en-us/library/ms190964.aspx
MSDN: DBCC SHRINKFILE (Transact-SQL): http://msdn.microsoft.com/en-us/library/ms189493.aspx
MSDN: DBCC SQLPERF (Transact-SQL): http://msdn.microsoft.com/en-us/library/ms189768.aspx