



Handling der Transaction Log-Files im MS SQL Server

Cagliari Salvatore . Consultant . 21. August 2008

Für die Erstellung einer SQL Server Datenbank sind viele Empfehlungen und Best Practices vorhanden. Für das Transaction Log-File hingegen wird vielfach einfach festgelegt, dass eine Platzierung auf einem separaten, performanten Diskarray nötig ist. Die Sicherung des Logs ist auch ein bekanntes Thema. Über die Grösse wird, insbesondere bei kleineren Projekten, aber nicht viel nachgedacht.

Nachfolgend führe ich ein paar Informationen auf, wie SQL Server das Transaction Log-File handhabt und wie die Grösse dimensioniert werden soll.

1 Wie SQL Server mit dem Transaction Log arbeitet

Eine wichtige Tatsache ist, dass der SQL Server, im Gegensatz zum Datenfile, sequentiell mit dem Transaction Log-File arbeitet.

Weiter ist wichtig, dass SQL Server ein Transaction Log-File als eine Kette von aneinandergereihten einzelnen Log-Files, die sogenannten Virtual Log Files (VLF), behandelt.

1.1 Wofür ein Transaction Log?

Die heute üblichen, transaktionsorientierten Datenbanksysteme nutzen ein, oder mehrere, Transaction Log-File(s) um die Konsistenz im Falle eines Abbruchs der Transaktion, oder wenn ein Fehler auftritt, zu gewährleisten. Aus diesen Gründen besteht jedes Log aus einer Kette von Log Einträgen (Log Records), welche den folgenden Inhalt haben:

- Eine eindeutige Seriennummer der Transaktion (Logical Sequence Number, LSN)
- Das Abbild der Änderung im Originalzustand (vor der Änderung)
- Das Abbild der Änderung danach

Unter anderem haben die folgenden Aktionen einen Log Eintrag zur Folge:

- Start und Ende einer Transaktion
- Jede Änderung an Daten (INSERT, UPDATE, DELETE) und an der Daten Struktur (DDL Statements), wie auch Änderungen, die in Stored Procedures stattfinden
- Jede Inanspruchnahme (Allozierung) von Datenseiten oder Extents

Wenn eine Transaktion widerrufen wird (Rollback), wird dies auch im Transaction Log festgehalten.

Jede Transaktion führt dazu, dass Platz im Transaction Log belegt wird. Der benötigte Platz hängt von der veränderten Datenmenge pro Transaktion ab.

Der Teil eines Transaction Logs, der für Recovery (Redo) oder Rollback (Undo) benötigt wird, wird nicht freigegeben und heisst deshalb auch „aktiver Teil“ des Transaction Logs.



Der aktive Teil des Transaction Logs für Undo Daten, kann nie freigegeben werden, damit eine offene Transaktion jederzeit ein Rollback durchführen kann.

Wird die Transaktion abgeschlossen (COMMIT), wird dies zuerst nur im Transaction Log festgehalten. Dies ist der Redo Teil im Transaction Log. Die Datenbank wird aus Performance Gründen asynchron nachgefahren.

Damit der Redo Teil nicht zu gross wird und freigegeben werden kann, gibt es Checkpoints. Diese aktualisieren die Datenbank mit den abgeschlossenen Transaktionen.

- Ein Checkpoint tritt ein wenn:
 - der Befehl CHECKPOINT manuell abgesetzt wird
 - SQL Server regelmässig einen Checkpoint auslöst
 - die Instanz gestoppt oder gestartet wird
 - ein Backup der Datenbank durchgeführt wird

 - eine Massen Transaktion im Bulk-Logged Recovery Model abgeschlossen wird
 - Datenfiles (ALTER DATABASE) der Datenbank hinzugefügt werden
 - die Datenbank geschlossen wird

Der aktive Teil des Transaction Logs beginnt demzufolge bei der ältesten offenen Transaktion (Undo), oder beim jüngsten Checkpoint (Redo). Was auch immer „kleiner“ ist, also zuerst im Transaction Log vorkommt.

Der inaktive Teil des Transaction Logs, wird freigegeben, wenn eines der folgenden Ereignisse eintritt:

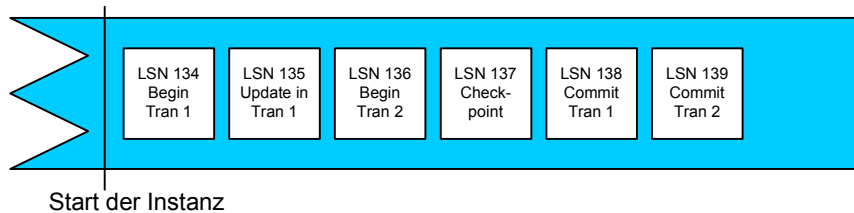
- Transaction Log Backup (nur im Full & Bulk-Logged Recovery Model)
- Ein Checkpoint im Simple Recovery Mode

1.1.1 Das aktive Log

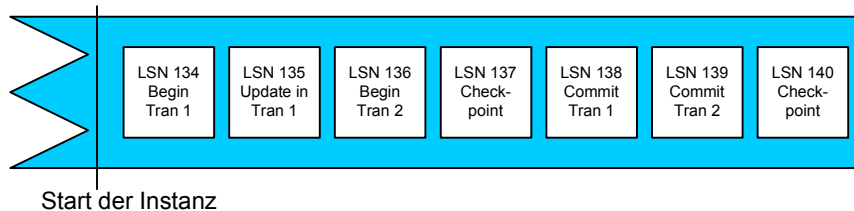
Die kleinste aktive LSN (älteste Transaktion ohne Commit oder jüngster Checkpoint) wird MinLSN genannt und jeweils beim Checkpoint bestimmt. Diese ist die kleinste LSN die für ein Recovery benötigt wird. Diese wird auch in die Bootpage des SQL Server geschrieben.

Wenn eine Transaktion nicht mit Commit abgeschlossen wird, muss der komplette Teil des Logs, seit dem Start der Transaktion, aktiv bleiben, auch wenn zwischendurch ein Checkpoint ausgelöst wird. Dies kann zu massiven Problemen führen, wenn das Transaction Log nicht wachsen darf, oder der Platz auf dem Volume, in der das Transaction Log-File liegt, ausgeht.

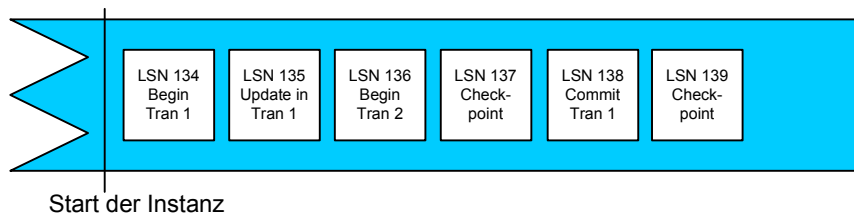
Vereinfacht sieht ein Log File folgendermassen aus:



In diesem Szenario ist MinLSN134. Obwohl Tran1 & 2 abgeschlossen sind, bleibt das Transaction Log aktiv. Damit ist ein Redo (Recovery) möglich, bis die Datenbank synchronisiert ist.



In diesem Szenario ist die MinLSN 140, da alle Transaktionen zu diesem Zeitpunkt abgeschlossen sind und der Checkpoint die Datenbank synchronisiert hat.



In diesem Szenario ist die MinLSN 136, da Tran 2 offen ist und jederzeit ein Undo (Rollback) dieser Transaktion möglich ist. Der nachfolgende Teil des Transaction Logs bleibt ebenfalls aktiv und das Transaction Log wächst, bis Tran 2 abgeschlossen wird. Dies ist die häufigste Ursache für ein grosses Transaction Log.

1.1.2 Replizierte Datenbanken

Bei Replikationen ist die Distribution Datenbank der relevante Punkt für das Transaction-Log Handling in der Publisher Datenbank.

Wenn eine Datenbank mit Transactional Replication repliziert wird, wird jeder Teil des Log aktiv gehalten, der (obwohl es schon abgeschlossene Transaktionen enthält) noch nicht in die Distribution DB geschrieben wurde.

Werden nun bei Transactional Replikation (Nicht bei Merge und Snapshot Replication) die Latenzzeiten zu gross, oder ist die Datenübermittlung unterbrochen, müssen alle Teile des Logs aktiv gehalten werden, die seit der letzten Replikation hinzugekommen sind und noch nicht in die Distribution DB geschrieben wurden. Das führt dazu, dass der für das Transaction Log benötigte Platzbedarf wächst.

Wenn die Latenzzeiten nicht verringert werden können, besteht hier die Möglichkeit den Intervall für den Log Reader zu verkürzen, oder auf Continuous zu schalten. Bevor hier eingegriffen wird, sollten die Intervalle und die Latenzzeiten mit dem Replication Monitor überprüft werden.



Weiterführende Informationen zu Faktoren die verursachen dass das Transaction Log aktiv bleibt, können Sie hier finden: <http://technet.microsoft.com/en-us/library/ms345414.aspx>.

1.1.3 Gespiegelte Datenbanken

Auch bei gespiegelten Datenbanken wird in der Principal Server Instanz jeder Log Record aktiv gehalten, bis die Mirror Server Instanz zurückgemeldet hat, dass dieser in die Zieldatenbank geschrieben wurde.

Beim Auftreten eines „Loss of Mirror“ oder „Mirror Suspend“ Events, müssen in der Primary Datenbank alle Teile des Logs aktiv gehalten werden, die seit der letzten Datenübermittlung hinzugekommen sind. Das führt dazu, dass der für das Transaction Log benötigte Platzbedarf wächst.

Wie in den Books Online beschrieben ist (<http://technet.microsoft.com/en-us/library/ms345414.aspx>) kann, wenn die Latenzen zu gross werden, um die noch vorhandenen Daten zu spiegeln, die Spiegelung unterbrochen werden. Danach muss auf dem Principal ein Transaction Log Backup gefahren werden um dieses auf dem Mirror (mit WITH NORECOVERY) aufzuspielen. Damit kann das Transaction Log auf dem Principal truncated (Siehe Kapitel 3) werden und der, für die Spiegelung, benötigte Platz wird nicht weiter wachsen. Die weiteren Änderungen von Benutzern laufen natürlich weiter im Log auf. Danach kann die Spiegelung wieder aufgesetzt werden.

Wichtig zu beachten ist hier, dass kein weiteres Log Backup gefahren werden darf zwischen dem letzten Log Backup auf dem Principal und dem (wieder-)aufsetzen der Spiegelung.

Falls möglich, empfehlen wir das wiederaufsetzen der Spiegelung auf Zeiten zu legen, bei der die Benutzer-Aktivität auf einem Minimum liegt, um die Grösse der Datenänderungen zwischen dem Log-Backup und dem Restart der Spiegelung zu minimieren.



2 Recovery Models

Der SQL Server kennt drei Recovery Models:

- Simple
- Bulk Logged
- Full

Diese unterscheiden sich in der Art wie die Daten im Transaction Log gespeichert bleiben, wenn die Transaktionen abgeschlossen sind. Dabei ist wichtig zu wissen, dass nur im Bulk-Logged Recovery Model zusammen mit einigen Operationen effektiv weniger ins Transaction Log geschrieben wird.

Abgesehen vom SQL Server Management Studio (Datenbank Eigenschaften/Optionen) kann das Recovery Model einer Datenbank auch in einem Query Window geändert werden:

```
-- Recovery Model der Datenbank AdventureWorks auf Full setzen
USE master
GO
ALTER DATABASE AdventureWorks SET RECOVERY FULL
GO

-- Recovery Model der Datenbank auf Bulk-Logged setzen
ALTER DATABASE AdventureWorks SET RECOVERY BULK_LOGGED
GO

-- Recovery Model der Datenbank AdventureWorks auf Simple setzen
ALTER DATABASE AdventureWorks SET RECOVERY SIMPLE
GO
```

Wichtig: Wenn das Recovery Model einer Datenbank vom Simple in das Full Recovery Model gewechselt wird, muss ein Full Backup gefahren werden, um die Log Kette zu starten. Es ist kein Full Backup nötig wenn zwischen Full und Bulk-Logged Recovery Model gewechselt wird. Transaction Log Backups sind hingegen nötig.

2.1 Full Recovery Model

Mit dem Full Recovery Model kann ein Datenverlust minimiert werden. In diesem Modus werden alle Transaktionen im Transaction Log verzeichnet und gehalten. Diese werden erst freigegeben, wenn ein Log Backup durchgeführt wird. Bis dahin wird der benötigte Platz für das Log wachsen.

Der grosse Vorteil dieses Recovery Models ist, dass wenn die Datenbank-Files beschädigt sind, oder auf diese nicht mehr zugegriffen werden kann, es möglich ist, alle Transaktionen bis zu dem Zeitpunkt des Fehlers wieder herzustellen („nachzufahren“). Dazu muss das Transaction Log aber noch intakt sein und ein Backup des aktiven Transaction Logs ist notwendig. Ein solches Backup wird Tail_Log Backup genannt:

```
-- Ein Tail-Log Backup erstellen
BACKUP LOG AdventureWorks TO DISK = 'D:\Backup\AdventureWorks_Tail.trn' WITH
NO_TRUNCATE

-- Ein Tail-Log Backup erstellen wenn das Log File auch defekt ist
BACKUP LOG AdventureWorks TO DISK = 'D:\Backup\AdventureWorks_Tail.trn' WITH
CONTINUE_AFTER_ERROR
```

Erläuterungen:



- NO_TRUNCATE erstellt ein Backup ohne dabei die nicht mehr benötigten Einträge zu löschen. Bei einem Log Backup ohne NO_TRUNCATE werden nicht mehr benötigte Daten im Transaction Log zum Löschen freigegeben. Siehe Kapitel 3 für eine genauere Erklärung über diesen Mechanismus. Entspricht der Angabe von COPY_ONLY
- CONTINUE_AFTER_ERROR bewirkt, dass allfällige defekte Blöcke ignoriert werden. Dies kann sowohl bei einem BACKUP wie auch bei einem RESTORE angegeben werden.

Damit kann man bei einem Datenbank Crash einen Datenverlust verhindern.

Trivadis empfiehlt aus diesem Grund für alle Produktiv-Datenbanken das Full Recovery Model.

Als Ausnahmen gelten Staging und Data-Warehouse (DWH) Datenbanken. Staging Datenbanken können im Simple Recovery Model betrieben werden, da hier nicht unbedingt die Datensicherheit des Full Recovery Models benötigt wird. In DWH Datenbanken besteht üblicherweise nicht die Notwendigkeit jede Transaktion wiederherzustellen. Aus diesem Grund kann das Simple Recovery Model ausreichend sein.

2.2 Bulk Logged Recovery Model

Das Bulk Logged Recovery Model stimmt in weiten Teilen mit dem Full Recovery Model überein. Der Unterschied ist, dass dieser Modus gewisse Operationen nur minimal in das Log schreibt. Die folgenden Operationen werden im Full Recovery Model voll geloggt, aber nur minimal im Bulk-Logged Model:

- Masseninserts mit bcp, BULK INSERT und OPENROWSET(BULK ...)
- text, ntext, und image Operationen mit den WRITETEXT und UPDATETEXT Statements, wenn neue Daten eingefügt oder angehängt werden. Bitte beachten Sie, dass wenn bestehende Daten verändert werden, wird diese Operation voll im Transaction Log geschrieben
- Veränderung von Daten mit grossen Datentypen (varchar(max),nvarchar(max), varbinary, text, ntext & image) mit der .WRITE Funktion des UPDATE Statements Analog dem oberen Punkt.
- SELECT INTO Operationen
- Gewisse INDEX DDL Operationen werden minimal geloggt:
 - CREATE INDEX Operationen (Incl. indexed Views)
 - ALTER INDEX REBUILD oder DBCC DBREINDEX Operationen
 - Ein Heap Rebuild nach einem DROP INDEX (Beim löschen eines Clustered Index)

Bei Massenoperation werden effektiv nur minimale Daten über die Transaktion in das Log geschrieben.

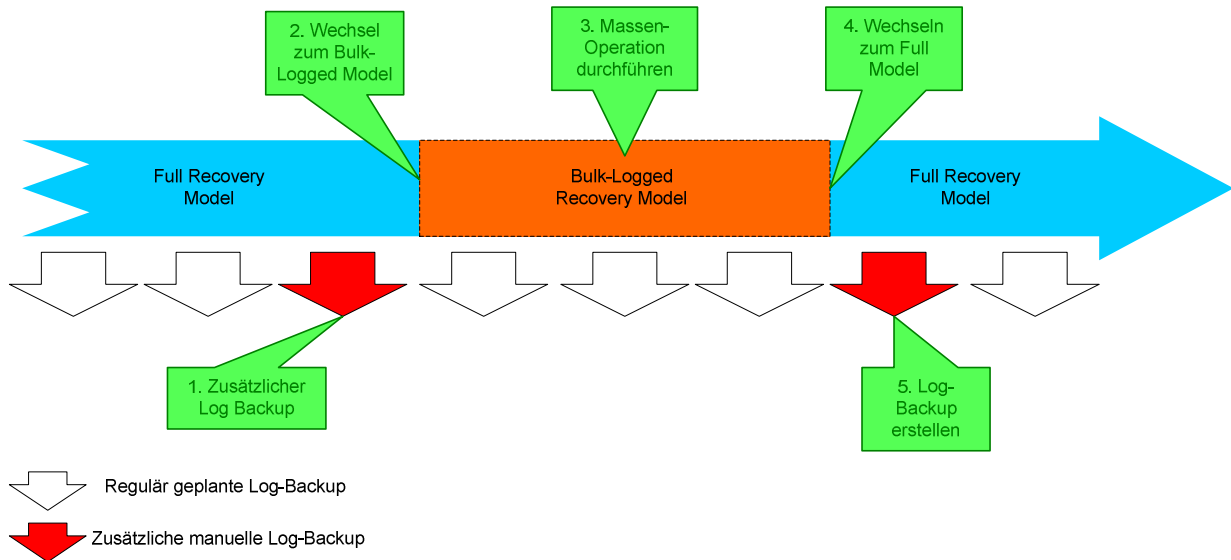
Dies macht dieses Recovery Model für DWH Datenbanken geeigneter als das Full Recovery Model.

Minimal geloggt bedeutet, dass nur minimale Informationen über Transaktionen geloggt werden, die aber kein Point-In-Time Recovery erlauben. So ist unter dem Bulk-Logged Recovery Model nur ein kompletter Log Restore möglich. Weiterführende Informationen sind hier erhältlich: <http://technet.microsoft.com/en-us/library/ms190692.aspx>.



Wenn in eine OLTP Datenbank einmal massenhaft Daten importiert werden müssen, ist es möglich diese vor dem Import in das Bulk-Logged Recovery Model zu wechseln, um ein massives wachsen des Transaction-Log zu verhindern. Nach Abschluss der Operation sollte die Datenbank wieder ins Full Model zurückgewechselt werden.

Im folgenden Bild ist dargestellt, wie eine derartige Operation auf einer Zeitleiste aussehen könnte.



Man sollte Massen Imports im Full Recovery Model vermeiden, da dies eine erhebliche Last für das Transaction Log bedeutet. Jeder einzelne Import wird im Log gespeichert. Dies kann die Grösse des Logs extrem erhöhen und zu Problemen mit der Performance und dem Platz führen.

2.3 Simple Recovery Model

In dieser Betriebsart werden in einem Transaction Log nur die aktiven Transaktionen gespeichert. Sobald eine Transaktion abgeschlossen ist, wird der belegte Platz beim nächsten Checkpoint wieder freigegeben.

Der Vorteil liegt im reduzierten Platz für das Transaction Log. Der SQL Server führt dabei selbständig das Log Management durch, es sind keine Transaction Log Backups nötig (oder möglich). Dafür nimmt man ein erhöhtes Risiko für einen Datenverlust in Kauf, wenn die Datenbank beschädigt wird.

Wird eine Datenbank vom Simple Recovery Model ins Full oder Bulk Logged Model gehoben, muss zuerst ein Full Backup gemacht werden. Erst nach einem Daten Backup startet der Logging Prozess im Full oder Bulk Logged Model. Eine neue Datenbank, welche mit dem Full Recovery Model konfiguriert ist, verhält sich also bis zum ersten Full Backup wie eine Datenbank im Simple Recovery Model.

2.4 Systemdatenbanken

Die Systemdatenbanken einer SQL Server Instanz beinhalten einige Restriktionen in Bezug auf die Recovery Models:



2.4.1 master Datenbank

Standardmässig im Simple Recovery Model kann man das Recovery Model zwar anpassen, sie wird aber immer so betrieben als wäre sie im Simple Model. BACKUP LOG ist auf der master Datenbank nicht unterstützt, auch differential Backups können nicht erstellt werden.

2.4.2 msdb Datenbank

Diese Datenbank wird defaultmässig im Simple Recovery Model betrieben und kann auch so betrieben werden. Wenn aber, z.B. aus Auditing Gründen, eine Lückenlose Sicherung der Backup oder der SQL Agent Job History nötig ist, kann sie auch im Full Recovery Model betrieben werden.

2.4.3 model Datenbank

Als Vorlage für neue Datenbanken ist diese Datenbank Standardmässig im Full Recovery Model. Diese Einstellung kann angepasst werden, wenn man alle zukünftigen Datenbanken auch in einem anderen Recovery Model als Full betreiben möchte.

2.4.4 Resource Datenbank

Diese Datenbank ist eine Nur-Lese Datenbank. Demzufolge ist das Recovery Model irrelevant.

2.4.5 Tempdb

Die tempdb kann nur im Simple Model betrieben werden. Die Datenbank kann nicht gesichert werden.

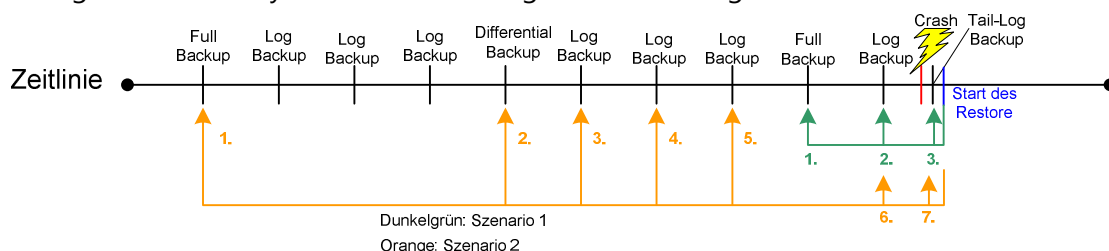
3 Sicherung des Transaction Log

Im Full und im Bulk-Logged Recovery Model muss das Transaction Log zusätzlich gesichert werden, um den maximalen Datenschutz zu ermöglichen. Hiermit ist ein Point in Time Recovery möglich. Im Simple Recovery Model ist ein Log Backup nicht möglich.

Das folgende Statement führt ein Log Backup durch:

```
-- Ein Log Backup erstellen  
BACKUP LOG AdventureWorks TO DISK = 'D:\Backup\AdventureWorks_Log.trn'
```

Ein möglicher Recovery Ablauf wird im folgenden Bild dargestellt:



Das Bild zeigt die Zeitlinie mit den diversen Full, Differential und Log Backups.

In diesem Fall gibt es einen Crash nach einem Log Backup. Da das Transaction Log noch verfügbar ist, wird ein Tail-Log Backup gefahren (Siehe 2.1), um die Änderungen seit dem letzten Log Backup zu sichern. Dieser Schritt wird in der Hektik eines Crashes häufig vergessen. Ohne ein Tail-Log Backup ist jedoch ein vollständiges Recovery nicht möglich. Also unbedingt daran denken!



Im Normalfall (Dunkelgrün im Bild) wird das neueste Full Backup (mit WITH NORECOVERY) wiederhergestellt, gefolgt vom Log Backup (Ebenfalls mit WITH NORECOVERY) und dem Tail-Log Backup (mit WITH RECOVERY). Dieses Vorgehen ermöglicht die Datenbank ohne Datenverlust wiederherzustellen.

Wenn wir nun annehmen, dass das neueste Full Backup nicht verfügbar oder korrupt ist, muss ein anderes Vorgehen gewählt werden (Orange im Bild). In diesem Fall muss das nächst ältere Full Backup (mit WITH NORECOVERY) wiederhergestellt werden, gefolgt vom Differential Backup und allen Transaction Log Backups (mit WITH NORECOVERY) seit dem Differential Backup inkl. dem Tail-Log Backup (mit WITH RECOVERY).

Damit wurden die gleichen Daten wiederhergestellt wie beim ersten Vorgehen.

Dies zeigt, dass die Transaction Log Backups unabhängig und zusätzlich von Full und/oder Differential Backups gemacht werden müssen.

3.1 Point-In-Time Recovery

SQL Server ermöglicht noch weitere Arten von Restore, die hier kurz gezeigt werden:

Wiederherstellung bis zu einer bestimmten Zeit

```
-- Ein Log-Backup bis zu einer bestimmten Zeit wiederherstellen
RESTORE LOG AdventureWorks FROM DISK = 'D:\Backup\AdventureWorks_Tail.trn'
WITH RECOVERY, STOPAT = '15.04.2008 14:00'
```

Wiederherstellung bis zu einer bestimmten Transaktion

```
-- Ein Log-Backup bis zu einer markierten Transaktion wiederherstellen.
RESTORE LOG AdventureWorks FROM DISK = 'D:\Backup\AdventureWorks_Tail.trn'
WITH RECOVERY, STOPATMARK = 'ListPriceUpdate'
```

Dies bedingt dass die Transaktion mit einem Namen markiert ist:

```
-- Ein Transaktion mit einem namen markieren
BEGIN TRANSACTION ListPriceUpdate
    WITH MARK ' ListPriceUpdate ';
GO
```

Wiederherstellung einer bestimmten, korrupten Daten-Seite

```
-- Ein Transaktion mit einem namen markieren
RESTORE DATABASE <database> PAGE='1:57, 1:202, 1:916, 1:1016'
    FROM DISK = 'D:\Backup\AdventureWorks_Tail.trn'
    WITH NORECOVERY;
```

Danach sollten allfällige Differentials und Log Backups wiederhergestellt werden.

Detaillierte Informationen wie korrupte Daten-Seiten gefunden werden können Sie hier finden:

<http://technet.microsoft.com/en-us/library/ms175168.aspx>

Die Möglichkeiten sind hier sehr vielfältig. Wir empfehlen an dieser Stelle die aktuelle SQL Server Books Online zu nutzen (Dort sind die Möglichkeiten sehr gut mit Beispielen dokumentiert).



Books Online für Download: <http://technet.microsoft.com/en-us/sqlserver/bb428874.aspx?wt.slv=TopSection>

Books Online für Online-Navigation: <http://msdn.microsoft.com/en-us/library/ms130214.aspx>

4 Grösse des Logs verwalten

Im Zusammenhang mit der Grösse des Logs ist es wichtig zwei Begriffe zu kennen:

- Truncation
- Shrinking

4.1 Truncation

Dies ist der Prozess der Freigabe der Logs im Transaction Log. Dies erfolgt, wenn ein Log Backup gefahren wird.

Dieser Prozess ist notwendig, um die nicht mehrbenutzten Teile des Transaction Logs wiederverwenden zu können.

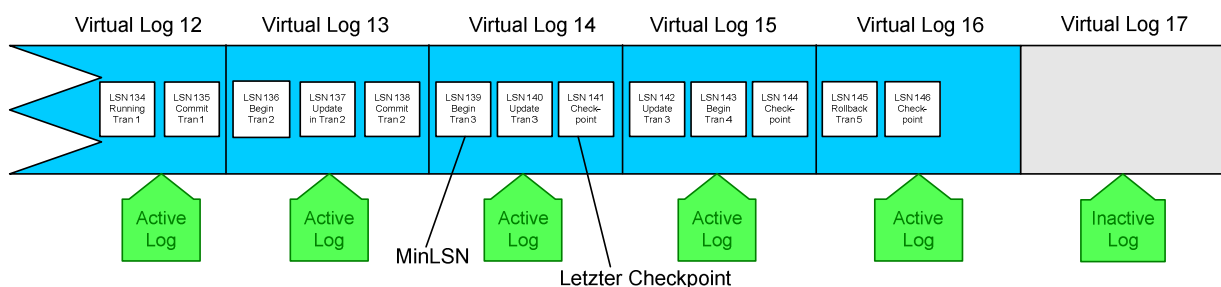
Im Simple Recovery Model löst zudem ein Checkpoint ein Truncation aus.

Vor SQL Server 2008 konnte das Log auch manuell truncated werden. Dies führte aber dazu, dass die Recovery Kette unterbrochen wurde und sofort ein Full Backup gefahren werden musste.

4.1.1 Funktionsweise der Truncation

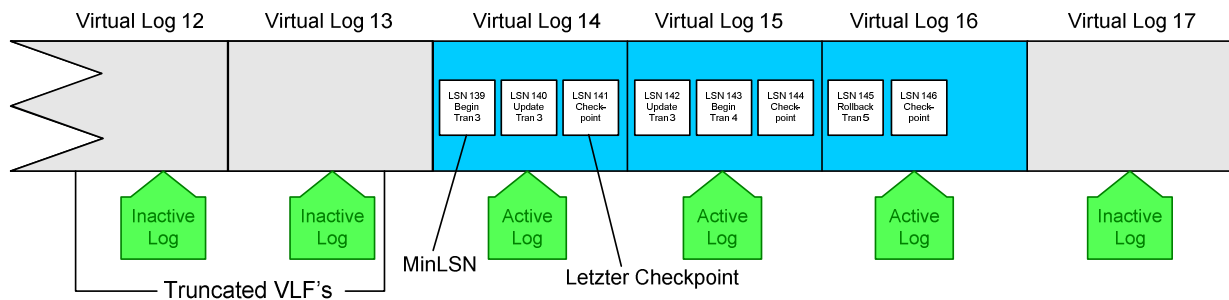
Der SQL Server schreibt sequentiell in das Transaction Log-File. Wenn das Ende erreicht ist und am Anfang Platz frei ist, beginnt er das Log wieder von Anfang an zu beschreiben. Dazu benutzt er Virtual Log Files (VLF) als logische Segmentierung des physischen Logs. Ein VLF kann entweder als ganzes aktiv sein oder nicht.

Das nachfolgende Bild illustriert diesen Status:



In diesem Fall sind alle VLF von 12 – 16 aktiv. Das erste VLF mit aktiven Transaktionen ist aber VLF 14. VLF 12 & 13 sind aktiv bis ein Log Backup diese wieder freigibt.

Nach einem Log Backup sieht die Sache wie folgt aus:



Die VLF 12 & 13 sind nun inaktiv, da die dort enthaltenen Log Records gesichert oder obsolet sind. Nun beginnt der aktive Teil des Logs beim VLF 14.

4.1.2 Überwachen der Truncation Aktivität

Mit Hilfe des Performance Monitors kann die Truncation Aktivität pro Datenbank beobachtet werden. Dazu kann der SQL Server:Databases/Log Truncation Zähler überwacht werden. Hierbei ist es wichtig eine Performance Baseline zu besitzen an der man einen Vergleich zu der aktuellen Aktivität ziehen kann. Ansonsten hat man wenige Möglichkeiten zu ermitteln ob die aktuellen Werte in Ordnung oder zu hoch sind.

4.2 Shrinking

Shrinking ist die Operation bei dem ein Log File physisch verkleinert wird.

```
-- Die Logischen Namen des Transactionlog-Files finden:  
EXEC sp_helpfile  
  
--Ergebnis:  
Name:                usage:  
AdventureWorks_Data  data only  
AdventureWorks_Log   log only  
  
-- Das Log auf 1MB oder die minimale Grösse shrinken  
DBCC SHRINKFILE (AdventureWorks_Log, 1)
```

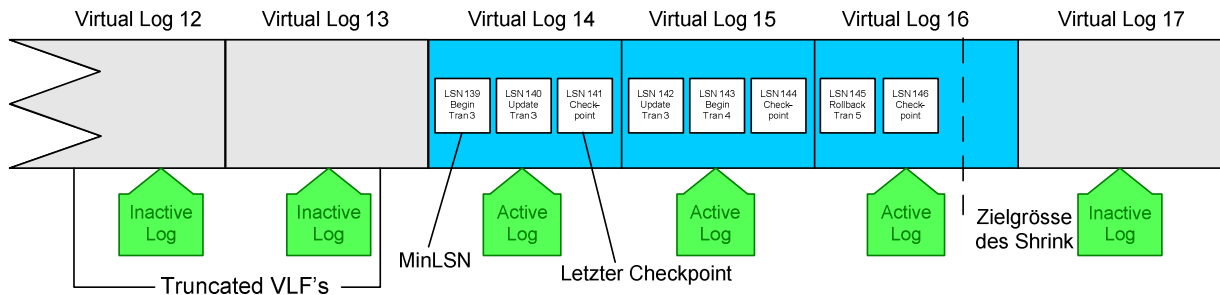
4.2.1 Funktionsweise des Shrinking

Ähnlich wie bei Truncation kann der SQL Server kein VLF freigeben das aktiv ist. Es ist also ratsam den folgenden Prozess zu befolgen, wenn man plant, die Grösse des Logs zu reduzieren:

1. Die Aktion auf eine Zeit legen in der wenig Aktivität auf der Datenbank herrscht
2. Ein Log Backup durchführen
3. Das Log File shrinken

Zum besseren Verständnis ein paar Bilder:

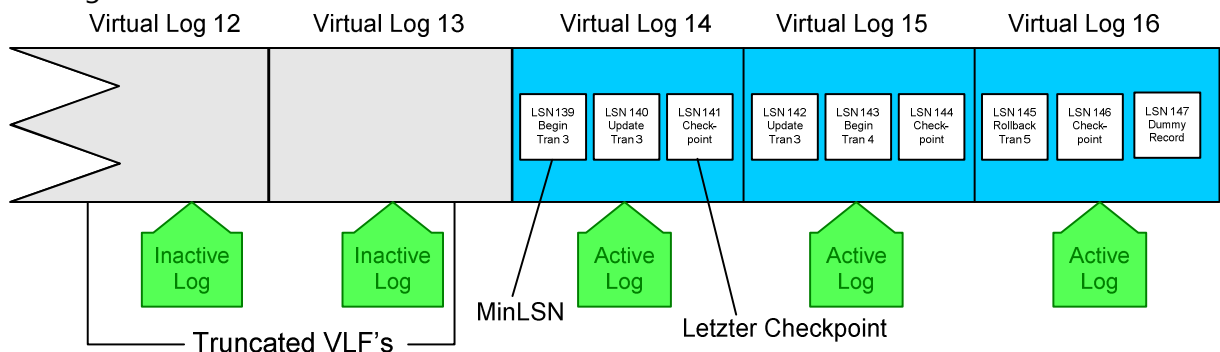
Als erstes die Ausgangslage:



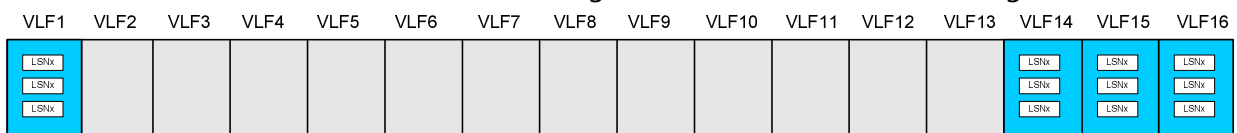
Hier haben wir ein Log in dem die VLF 12, 13 und 17 unbenutzt sind. Das Shrinking kann nur vom Ende des Files freie VLF löschen. Es bewegt keine Daten wie beim Shrink eines Datenfiles.

Wir nehmen nun an, dass bei einem Versuch das File bis zu einer bestimmten Grösse (Im Bild mit Target-Size angegeben) zu shrinken, die Grenze zur angegebenen Grösse am Ende des VLF 16 liegt. In diesem Fall gibt SQL Server eine Information aus, dass es nicht möglich ist die angegebene Grösse zu erreichen, weil Teile des Logs noch aktiv sind.

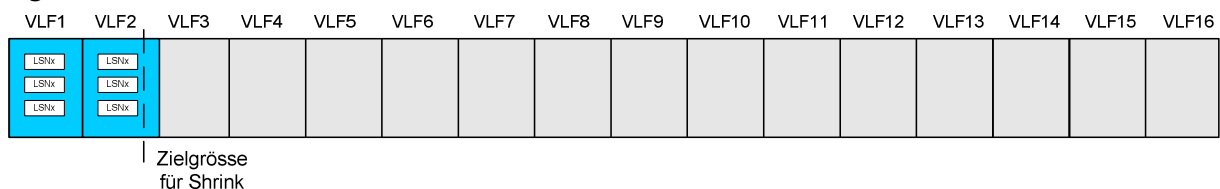
Das Log File sieht danach so aus:



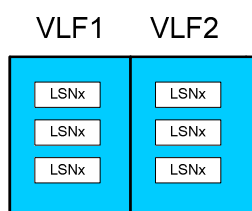
Der SQL Server füllt das VLF 16 mit Dummy Log Records auf, um sicherzustellen, dass dieses VLF nicht mehr benutzt wird. Die nächsten Log Records werden nun im VLF 1 geschrieben:



Nach dem nächsten Log Backup sind die Log Records in den VLF 14 - 16 inaktiv. Nun sieht das Log so aus:



Nun kann erneut versucht werden das Log File zu shrinken. Nun werden alle VLF ab VLF3 gelöscht, da es nicht möglich ist, das Log File auf eine Grösse zu shrinken, die nicht den Grenzen der VLF entspricht.





4.2.2 Shrinken: Ja oder Nein

Im Gegensatz zu Datenfiles, welche normalerweise nicht geshrinkt werden sollten, kann es durchaus einen Nutzen haben das Transaction Log-File zu shrinken. Zum Beispiel, wenn das Log-File wegen eines Massenimportes im Full Recovery Model jenseits der Grösse im Normalbetrieb gewachsen ist, oder wenn das Backup des Logs mehrere Male hintereinander schiefgelaufen ist und das Log entsprechend gewachsen ist. Man sollte sich aber bewusst sein, dass wenn die Transaktionen wieder mehr Platz im Transaction Log-File benötigen, dieses auch wieder wächst. Das Wachstum bedeutet wiederum eine Performanceeinbusse. Solange ein Transaction Log-File wächst, kann nichts im Transaction Log-File geschehen. Sämtliche Transaktionen stehen solange still. Die einfache Empfehlung heisst deshalb, Transaction Log-File genügend gross anlegen und mit Transaction Log Backups die Grösse im Griff halten. Weiteres hierzu in einem nachfolgenden Kapitel.

Sollte es doch nötig sein, das Logfile zu shrinken, dann ist darauf zu achten dass die Zielgrösse des Transaction Logs für den täglichen Betrieb geeignet ist. Damit verhindern Sie die obengenannten Beeinträchtigungen.

Auch die Instant File Initialization Feature (<http://technet.microsoft.com/en-us/library/ms175935.aspx>) hilft in diesem Fall nicht weiter, da eine Erweiterung eines Transaction Log-Files (Ein Fragment) auch die Struktur eines Transaction Log-Files erhalten muss. Das Transaction Log File wird deshalb mit Nullen initialisiert. Dies kann, je nach Performance des Disk Subsystem und Ziel-Grösse des Transaction Log-Files, eine gewisse Zeit dauern.

5 VLF

Das obengenannte Beispiel zeigt einen wichtigen Punkt im Handling des Logs:

Ein VLF ist die kleinste, nicht spaltbare Einheit eines Transaction Log-Files

Der SQL Server definiert die Grösse der VLF automatisch. Er orientiert sich an der Initial Grösse des Transaction Log-Files und der Grösse der Erweiterung(en).

Dabei gelten die folgenden Regeln:

- Wenn ein Fragment weniger als 64MB gross ist, werden max. 4 VLF erstellt. Bei einer Grösse von weniger als 1 MB können auch weniger als 4 Fragmente erstellt werden.
- Wenn ein Fragment mehr als 64MB aber weniger als 1GB gross ist, werden 8 VLF erstellt
- Wenn ein Fragment mehr als 1GB gross ist, werden 16 VLF erstellt

Hier ein Beispiel:

Anmerkung: Die Anzahl VLF kann man mit DBCC LOGINFO anzeigen lassen. Dieser Befehl ist undokumentiert.

Eine Datenbank wurde mit einem Transaction Log-File von 250 MB Grösse angelegt. Hier die Ausgabe des DBCC LOGINFO:

No	FileID	FileSize	StartOffset	FSeqNo	Status	Parity	CreateLSN
1	2	26148864	8192	18	2	64	0
2	2	26148864	26157056	0	0	0	0



3	2	26148864	52305920	0	0	0	0
4	2	26148864	78454784	0	0	0	0
5	2	26148864	104603648	0	0	0	0
6	2	26148864	130752512	0	0	0	0
7	2	26148864	156901376	0	0	0	0
8	2	26664960	183050240	0	0	0	0

Anmerkung: Die Spalte No. wurde für die Übersichtlichkeit eingefügt

Die Spalte FileSize zeigt in Bytes die Grösse des VLF an. Die Spalte Status zeigt an, ob dieses VLF aktiv ist. Der erste VLF ist mit Status 2 aktiv.

Nun wird das Transaction Log-File der Datenbank mit dem folgendem Befehl erweitert:

```
-- Das Log der Datenbank AdventureWorks auf 1.5 GB vergrössern:
ALTER DATABASE [AdventureWorks]
    MODIFY FILE
        (NAME = AdventureWorks_Log',
         SIZE = 1500MB)
```

Ein DBCC LOGINFO führt nun zu folgendem Ergebnis:

No	FileID	FileSize	StartOffset	FSeqNo	Status	Parity	CreateLSN
1	2	26148864	8192	18	2	64	0
2	2	26148864	26157056	0	0	0	0
3	2	26148864	52305920	0	0	0	0
4	2	26148864	78454784	0	0	0	0
5	2	26148864	104603648	0	0	0	0
6	2	26148864	130752512	0	0	0	0
7	2	26148864	156901376	0	0	0	0
8	2	26664960	183050240	0	0	0	0
9	2	85196800	209715200	0	0	0	18000001512700000
10	2	85196800	294912000	0	0	0	18000001512700000
11	2	85196800	380108800	0	0	0	18000001512700000
12	2	85196800	465305600	0	0	0	18000001512700000
13	2	85196800	550502400	0	0	0	18000001512700000
14	2	85196800	635699200	0	0	0	18000001512700000
15	2	85196800	720896000	0	0	0	18000001512700000
16	2	85196800	806092800	0	0	0	18000001512700000
17	2	85196800	891289600	0	0	0	18000001512700000
18	2	85196800	976486400	0	0	0	18000001512700000
19	2	85196800	1061683200	0	0	0	18000001512700000
20	2	85196800	1146880000	0	0	0	18000001512700000
21	2	85196800	1232076800	0	0	0	18000001512700000
22	2	85196800	1317273600	0	0	0	18000001512700000
23	2	85196800	1402470400	0	0	0	18000001512700000
24	2	85196800	1487667200	0	0	0	18000001512700000

Anmerkung: Die Spalte No. wurde für die Übersichtlichkeit eingefügt

Wie Sie sehen, beginnt nach dem VLF 9 das angehängte Fragment.

Diese VLF habe eine andere Grösse als die zu Beginn:

VLF 1 – 7: (26148864 / 1024)/1024 = 24.93 MB

VLF 8: (26664960 / 1024)/1024 = 25.43 MB

VLF 9 – 24: (85196800 / 1024)/1024 = 81.25 MB



Wie Sie weiter sehen, bleibt die Grösse der VLF über das Transaction Log-File nicht in jedem Fall konstant, wenn es vergrössert wird.

Es ist eine Balance zwischen Anzahl und Grösse der VLF zu finden. Die Grösse der VLF sollte aber 512 MB nicht überschreiten, da die VLFs ja als Einheit verwaltet werden (siehe Diskussion bei Truncate und Shrink). Das heisst auch, dass bei Initialgrösse und Wachstumsrate nicht mehr als 8192 MB angegeben werden sollte.

Wir empfehlen an dieser Stelle auch die Wachstumsrate in absoluten Werten (MB oder GB) anzugeben, anstelle in %. Damit kann man genau steuern wie viele VLF's erstellt werden sollen.

Bei sehr grossen Datenbanken kann man unter Umständen eine grosse Anzahl von VLF nicht verhindern. Man sollte sich deshalb überlegen, ob man ein Transaction Log mit einer Zielgrösse von 12GB in mehreren Schritten von z.B. 4 GB aufbaut um die VLF mit 256MB Grösse zu erhalten, anstelle von 768MB pro VLF, wenn es direkt mit 12 GB angelegt wird.

Falls die VLF's zu gross sind, kann ein Performance Problem auftreten, da der SQL Server zuviel Zeit auf einmal damit verbringen muss, die nicht mehr benötigten Transaktionen aus den inaktiven VLF's zu löschen. Bei kleineren VLF kann der Aufwand über die Zeit verteilt werden.

Viele kleine VLF können ein Performance Problem hervorrufen, da sehr oft ein VLF Wechsel stattfindet und entsprechend mehr Aufwand in dieser Richtung entsteht.

5.1 Planen des Transaction Log und der VLF

Um die Grösse der VLF richtig planen zu können, spielen einige Faktoren eine Rolle:

1. Anzahl der Transaktionen in einem bestimmten Zeitraum (z.B. pro Std. oder pro Tag)
2. Durchschnittliche Grösse der Daten pro Transaktion
3. Transaktionsverhalten der Applikation
4. Häufigkeit des Transaction Log Backups, welche den möglichen Datenverlust beeinflusst
5. Zur Verfügung stehender Platz
6. Wachstumsrate der Datenmenge oder der Nutzungsrate der Datenbank
7. Regelmässige Maintenance Tasks für Indexe

Insbesondere Punkt 3 kann nur sehr schwer abgeschätzt werden. Zum Beispiel ist entscheidend, ob eine Applikation Transaktionen unnötig lange offen hält, oder sogar eine unnötige Anzahl von Transaktionen generiert (z.B. BEGIN TRANSACTION in einem Loop).

Als Anfangsgrösse kann man sich auf die weiter unten genannten Best Practices stützen und danach individuell entscheiden, wie das weitere Wachstum gesteuert werden kann.

Die Fragmentierung des File-Systems sollte auch in Betracht gezogen werden, da dies ein wesentlicher Faktor bei der Performance ist.

Wenn man eine Applikation nicht ganz genau kennt, kann man nur das Wachstum des Logs beobachten und entsprechend planen. Wir empfehlen an dieser Stelle eine initiale File Grösse von 1024MB. Dies ergibt eine VLF Grösse von exakt 128MB. Diese Grösse sollte auch als



Wachstumsrate benutzt werden, um eine konstante Grösse der VLF zu gewährleisten. Dies hat sich in der Praxis bewährt und kann als Best Practice angesehen werden.

5.2 Freier Platz im Transaction Log anzeigen

Mit dem Befehl DBCC LOGININFO kann man aber ausrechnen wie weit das Log File mit einem DBCC SHRINKFILE verkleinert werden kann, respektive wie viel Platz im Log Files genutzt wird.

Dafür kann auch ein anderer (dokumentierter) Befehl genutzt werden:

```
-- Der genutzte Platz im Log aller Datenbanken in der Instanz anzeigen  
DBCC SQLPERF (LOGSPACE)
```

Ergebnis:

Database Name	Log Size (MB)	Log Space Used (%)	Status
master	3.054688	41.81586	0
tempdb	0.7421875	52.69737	0
model	0.9921875	34.25197	0
msdb	1.992188	43.33333	0
ReportServer	0.8046875	54.49029	0
ReportServerTempDB	0.8046875	48.60437	0
AdventureWorks	1.992188	47.13235	0

Der freie Platz kann auch über den Performance Monitor (SQL Server:Databases/Percent Log Used Zähler) beobachtet und überwacht werden

5.2.1 Anlegen weiterer Logfiles

Wenn z.B. der Platz auf dem File-System ausgeht, ist es möglich weitere Log-Files anzulegen. Wichtig hierbei ist es zu wissen, dass dies kein Performance-Vorteil bringt, da der SQL Server die vorhandenen Logfiles immer sequentiell beschreibt, ohne Rücksicht ob es mehrere davon gibt oder nicht.

6 Fazit

Das Handling und das Management des Transaction Logs ist eine aufwendige, aber wichtige Angelegenheit. Diese zu unterschätzen oder zu vernachlässigen, wird zu Performance Problemen führen.

Hierbei ist das Handling der Recovery Models, die Backups und das Wachstum des Logs genauso in die Planung mit einzubeziehen, wie auch die gewünschten Wiederherstellungsmöglichkeiten und die vorgegebenen Zeiten (Time to Recover im SLA) dafür.

Unsere Empfehlungen an dieser Stelle sind:

- Sämtliche produktiven OLTP Datenbanken im Full Recovery Model zu fahren
- Als Ausnahmen gelten die im Abschnitt 2.1 erwähnten Gründe wie Staging Datenbanken oder DWH Datenbanken
- Regelmässig, zwischen 5 und 60 Minuten, ein Transaction Log Backup fahren, hängt im Detail vom jeweiligen Backup/Recovery Konzept ab.
- Nach Bedarf tägliche Differential mit wöchentlichen Full Backups oder tägliche Full Backups.



- Die Vergrößerung des Transaction Log so anlegen, dass die VLF's Ihren Anforderungen entsprechen (Siehe oben)

Trivadis kann Ihnen mit unserem Fachwissen helfen die beste Lösung für Ihre Umgebung zu finden.

Viel Erfolg beim Einsatz von Trivadis-Know-how wünscht Ihnen

Salvatore Cagliari

Trivadis AG

Elisabethenanlage 9

CH-4051 Basel

Internet: www.trivadis.com

Tel: +41-61-279 97 55

Fax: +41-61-279 97 56

Mail: info@trivadis.com

Literatur und Links...

www.trivadis.com

Die obenstehenden Informationen wurden aus den folgenden Quellen zusammengefasst:

- SQL Server 2005 Books Online September 2007
- <http://www.sqlskills.com>, Blog von Kimberly Tripp
 - Zum Beispiel Suchen nach VLF
 - <http://www.sqlskills.com/blogs/kimberly/2005/06/25/8StepsToBetterTransactionLogThroughput.aspx>
- <http://sqlblogcasts.com/blogs/tonyrogeron/archive/2007/07/25/sql-2000-yes-lots-of-vlf-s-are-bad-improve-the-performance-of-your-triggers-and-log-backups-on-2000.aspx> Toni Rogerson (MVP)
- http://www.karaszi.com/SQLServer/info_dont_shrink.asp Nukleus Dataconsult
- <http://support.microsoft.com/kb/317375/de> oder in Englisch <http://support.microsoft.com/kb/317375/en-us>
- <http://www.mssqltips.com/tip.asp?tip=1178> Monitoring SQL Server database transaction log space