

SQLintersection

Session: Wednesday, 1:30pm-2:30pm

Understanding Logging and Recovery

Paul S. Randal

paul@SQLskills.com




SQL
intersection



Paul S. Randal



- **Consultant/Trainer/Speaker/Author**
- **CEO, [SQLskills.com](http://www.SQLskills.com)**
 - Email: Paul@SQLskills.com
 - Blog: <http://www.SQLskills.com/blogs/Paul>
 - Twitter: @PaulRandal
 - 5 years at DEC responsible for the VMS file-system and chkdsk
 - Almost 9 years as developer/manager in the SQL Storage Engine team through August 2007, ultimately responsible for Core Storage Engine
- **Instructor-led training (US, UK, Ireland, Australia), consulting (anything you need)**
- **Online training:  <http://pluralsight.com/>**
- **Get our bi-weekly newsletter: <http://www.sqlskills.com/Insider>**



Reminder: Intersect with Speakers and Attendees

- **Tweet tips and tricks that you learn and follow tweets posted by your peers!**
 - Follow: #SQLIntersection and/or #DEVIntersection
- **Join us – Wednesday Evening – for SQLafterDark**
 - Doors open at **7:00 pm**
 - Trivia game starts at **7:30 pm**
 - Winning team receives something fun!*
 - Raffle at the end of the night
 - Lots of great items to win including a seat in a SQLskills Immersion Event!*
 - The first round of drinks is sponsored by SentryOne and SQLskills



Why Is This Important?

- **The transaction log is a black-box to many people**
 - It's very misunderstood: why is it necessary? why does it grow?
 - It gets abused: misconfigured, shrunk, grown, deleted
- **The transaction log is THE most important part of an active database**
 - Logging allows transactions to be made durable and recoverable in the event of a crash
 - Without logging, a database would be transactionally inconsistent, and potentially corrupt after a crash
 - Without logging, how would a transaction roll back?
 - Without logging, how would backups work? Replication? Mirroring? Availability Groups? Log shipping?

Overview

- **Transaction log architecture**
- **Crash recovery**
- **Recovery models**
- **Log file provisioning and maintenance**

Write-Ahead Logging

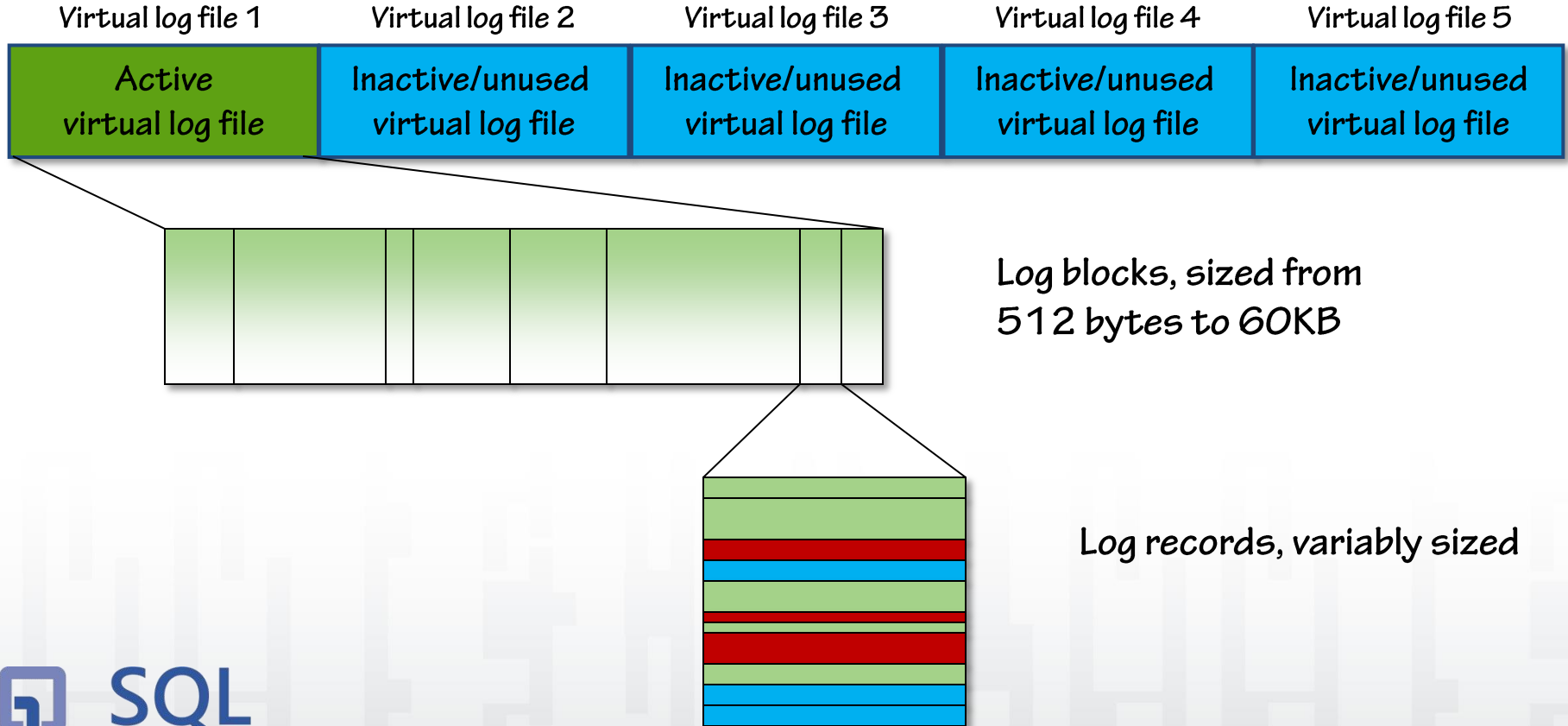
- SQL Server uses a mechanism called 'write-ahead logging'
- This ensures that a data-file page change cannot EVER be written to disk before the log records describing the change are written to disk
 - It's an invariant, even when delayed durability is used in SQL Server 2014+
- Without the write-ahead logging guarantee, how would recovery work?
- Write-ahead logging combined with periodic checkpoints also increases the efficiency of persisting changes to disk
- Books Online description: <http://bit.ly/UewypW>

Architecture: Virtual Log Files

Virtual log file 1	Virtual log file 2	Virtual log file 3	Virtual log file 4	Virtual log file 5
Active virtual log file	Inactive/unused virtual log file	Inactive/unused virtual log file	Inactive/unused virtual log file	Inactive/unused virtual log file

- **The transaction log is divided up into chunks called virtual log files, or VLFs for short**
- **Newly created VLFs are inactive and unused**
 - An active VLF cannot be reused until it is made inactive by log clearing
- **Except that in a new database, the first VLF is always active**
 - There must always be at least one active VLF in the transaction log
- **There is an 8KB file header page at the start of the transaction log file**
 - Stores metadata about the file such as size and auto-growth settings

Architecture: VLFs and Log Blocks



Log blocks, sized from
512 bytes to 60KB

Log records, variably sized

Log Sequence Numbers (LSNs)

- **LSN = <VLF sequence #>:<log block #>:<log record #>**
 - VLF sequence number is 4-bytes
 - The log block number within the VLF is 4 bytes
 - The log record number within the log block is two bytes
- **LSNs are ever-increasing**
- **Each log record has a unique LSN that allows the log record to be found in the transaction log**
- **Each data-file page has an LSN in its page header that identifies the most recent log record whose change is reflected on the page**
 - This is critical for recovery

What are Log Records?

- A log record describes a single change in the database
- Each log record has a unique Log Sequence Number (LSN)
- Log records for concurrent transactions are intermingled in the transaction log according to when they occurred in time
- Log records are stored in log blocks in the buffer pool until they are flushed to disk
- There are usually **NO non-logged operations in user/system databases**
 - In tempdb, version store and workfile operations are non-logged
 - In-memory OLTP in 2014+
- **Log records never move in the transaction log**

Log Records in Transactions

- **Multiple log records are generated by transactions, always in sequence:**
 - LOP_BEGIN_XACT
 - This includes information like the SPID, transaction name, start time
 - All transactions started by SQL Server are named to describe the operation
 - E.g. AllocFirstPage, DROPOBJ, INSERT
 - Other records...
 - LOP_COMMIT_XACT (if the transaction commits)
 - LOP_ABORT_XACT (if the transaction rolls back)
 - These both include the end time
- **Log records in a transaction are linked together backwards by LSN**
 - This allows the transaction to be rolled back correctly

The Transaction has Committed... Now What?

- **All log records have been ‘hardened’ on disk in the transaction log**
 - The transaction is durable and its effects can be replayed after a crash
- **Data-file pages with the changes on them are still in memory only**
 - These are called ‘dirty’ pages
- **Why are the changed data-file pages not written to disk when the transaction commits?**
 - They do not need to be written to disk as the description of the changes applied to them is already on disk in the transaction log
- **Changed data-file pages are written to the data files periodically by a mechanism called ‘checkpoint’**

Why Do Checkpoints Exist?

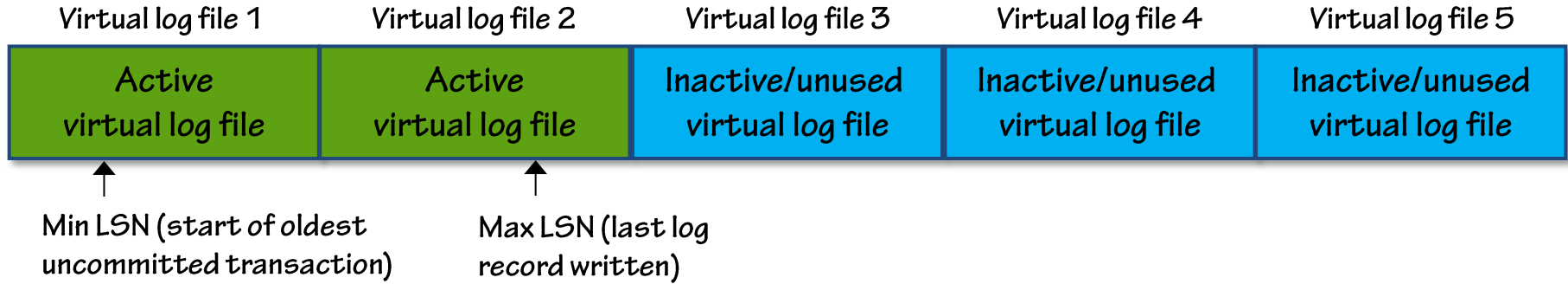
- **To reduce crash-recovery time**
 - By having as many up-to-date data-file pages in the database as possible, this reduces the amount of redo that must be performed
- **To batch I/Os to disk and improve performance**
 - Imagine 1000 update transactions to a single data page
 - Is it more efficient to write the data page image to disk after each change (i.e. 1000 times) or just once, during a checkpoint?

Making a VLF Active

Virtual log file 1	Virtual log file 2	Virtual log file 3	Virtual log file 4	Virtual log file 5
Active virtual log file	Inactive/unused virtual log file	Inactive/unused virtual log file	Inactive/unused virtual log file	Inactive/unused virtual log file

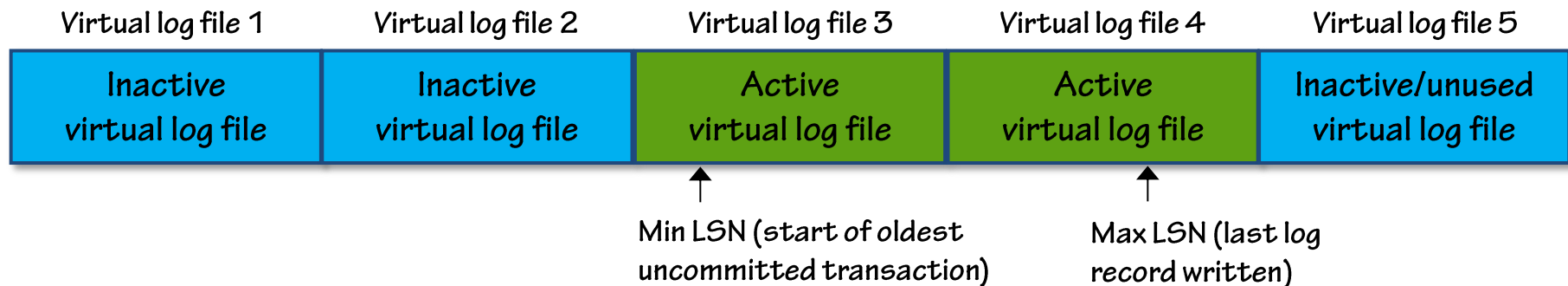
- **When a change is made, a log record is written to the transaction log**
- **The first log record written to a VLF makes it become active**
 - A VLF is either wholly active or inactive
- **An active VLF cannot be overwritten until it becomes inactive**
 - A VLF remains active until all log records in it are not needed for:
 - Log or data backups
 - Transactional replication or change data capture
 - Database mirroring/availability groups
 - Long-running transaction rollback or crash recovery

Moving Through the Transaction Log



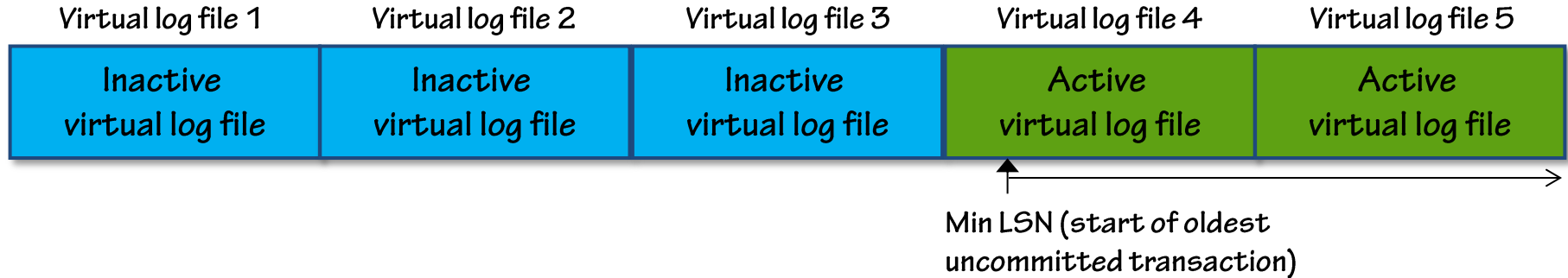
- **As more log records are written, more VLFs become active**
- **SQL Server tracks:**
 - Which portion of the transaction log is still required
 - The start of the oldest uncommitted transaction
 - Defines the oldest active LSN and therefore the oldest active VLF
 - The most recent log record written

Transaction Log Clearing



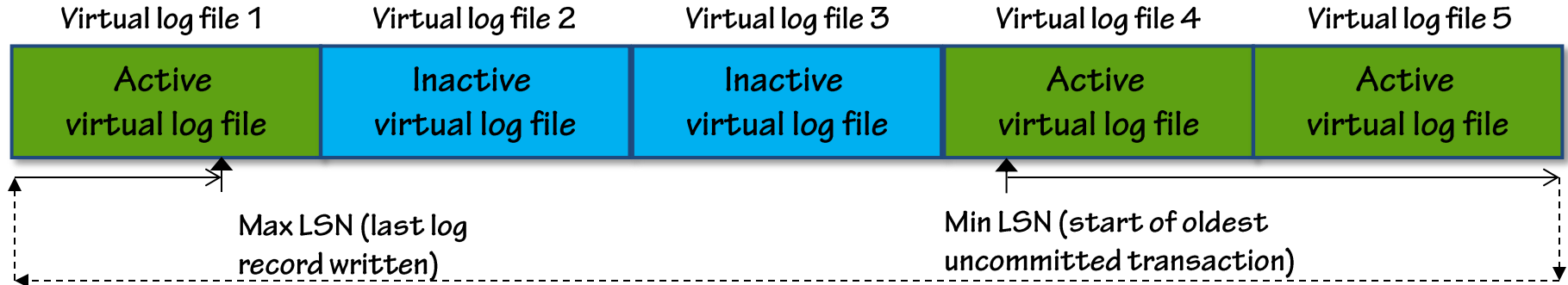
- **A VLF can be made inactive once all log records are not required**
 - This is called clearing (or truncating) the transaction log
- **Log clearing is done by a log backup in FULL or BULK_LOGGED recovery models, or by a checkpoint in SIMPLE recovery model**
- **All that happens is that zero or more VLFs are marked inactive**
- **If log clearing does not occur, the transaction log will grow forever**

Circular Nature of the Transaction Log



- Once the end of the transaction log is reached, it would like to wrap around and start using the first VLF again, as long as log clearing has occurred, without having to grow the transaction log file
- The log manager checks the active status of the first VLF in the transaction log

Circular Nature of the Transaction Log (2)

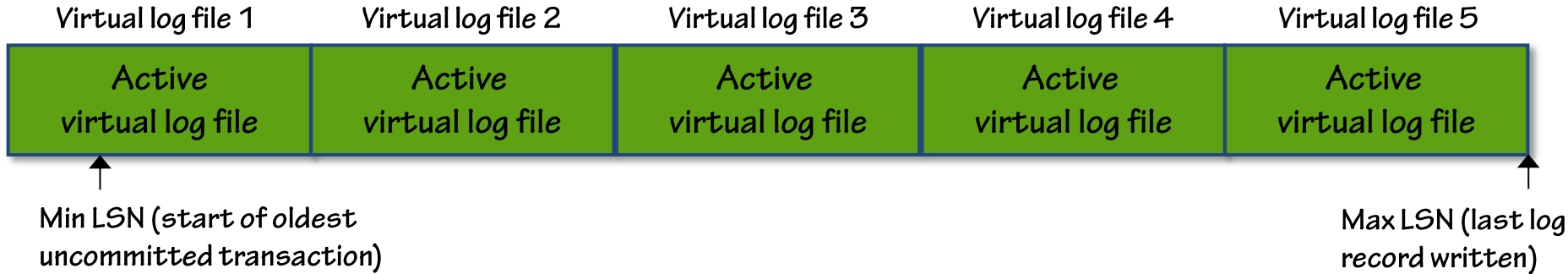


- **If the first VLF is inactive, the transaction log can wrap**
 - Note that VLF 1 is reactivated and becomes VLF 6
- **Log records continue to be written**
- **This post has an example: <http://bit.ly/bQ3CzS>**

Demo

Circular nature of the log

If the Transaction Log Fills Up...



- **The transaction log will auto-grow (if configured and possible)**
 - Transaction log auto-grow means a pause for zero-initialization
- **If the transaction log cannot auto-grow, write activity stops**
 - Uncommitted transactions will roll back
- **Avoid this situation by allowing the transaction log to clear!**

Why Did the Transaction Log Fill Up?

- **See log_reuse_wait_desc in sys.databases for why the log can't clear**
 - Examples are LOG_BACKUP, DATABASE_MIRRORING, NOTHING
 - This is the reason why log clearing failed the last time it was attempted
 - E.g. it might be ACTIVE_BACKUP_OR_RESTORE but the backup already finished
 - Comprehensive blog post on all of them: <http://bit.ly/1pOTjQP>
- **Take whatever corrective action can be done:**
 - Take a log backup (lack of backups is the #1 cause of full transaction logs!)
 - Kill a long-running transaction
 - List all transactions using code from Ian Stirk at <http://bit.ly/1aWpRCD>
 - Manually grow the existing log file(s) or add another log file (temporarily)
 - Switch to the SIMPLE recovery model as last resort, as it breaks log chain

Tracking Transaction Log Space Usage

- **Using Performance Monitor, from the Databases performance object:**
 - Log File(s) Size (KB)
 - Log File(s) Used Size (KB)
 - Percent Log Used
 - Log Growths and Log Shrinks
- **DBCC SQLPERF (LOGSPACE)**
 - Returns information for all databases including size and percentage used
 - Processing of the output requires creating a table and then using INSERT ... EXEC to capture the results
- **sys.dm_db_log_space_usage**
 - Returns information for the current database only

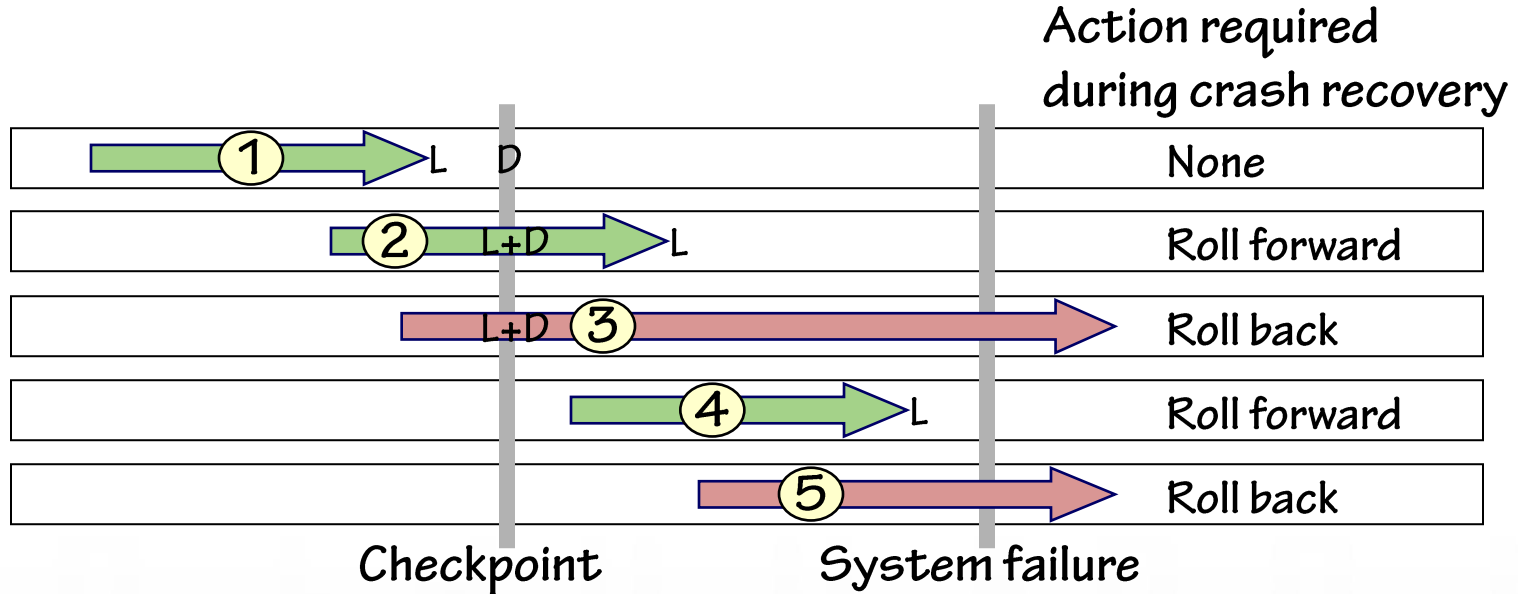
Demo

Runaway log file

Crash Recovery

- **For each database:**
 - Boot page is examined to get the LSN of the most recent checkpoint
 - Checkpoint log records are examined to get oldest uncommitted transaction
 - Recovery starts from the LSN of the LOP_BEGIN_XACT log record of the oldest uncommitted transaction at the time of the most recent checkpoint
- **Three passes are made through the transaction log:**
 - 1) read all required data file pages into memory, build list of committed and uncommitted transactions, and discovers the 'end' of the log
 - 2) perform redo of log records for committed transactions
 - 3) perform undo of log record for uncommitted transactions
- **Track recovery in 2016+ with XEvents: see <http://bit.ly/1MXHn7P>**

Crash Recovery Visualized



Each arrow is a transaction

L = Log records written to disk; D = Data file pages written to disk

Recovery Models

- **FULL: everything is fully logged**
 - ❑ Log truncation usually not possible until log backup
 - ❑ Operations like creating or rebuilding an index creates as much log as the size of the index being created/rebuilt
- **BULK_LOGGED: minimal logging for SOME things**
 - ❑ Log truncation usually not possible until log backup
 - ❑ NOT non-logged, just minimally-logged
 - ❑ Log backups will NOT be smaller then when in FULL!
- **SIMPLE: same logging as for BULK_LOGGED**
 - ❑ Log is cleared/truncated on checkpoint
 - ❑ No log backups possible

Minimally-Logged Operations

- **Small list of operations that can be minimally-logged**
 - Complete list is in Books Online: <http://bit.ly/PoX2xe>
- **Creating, dropping or rebuilding indexes**
 - Reorganizing indexes is always fully logged
- **Bulk-load operations**
 - BCP, OPENROWSET (BULK, ...), BULK INSERT
- **SELECT INTO a new permanent table**
- **Whitepaper: Data Loading Performance Guide (<http://bit.ly/7rTwT9>)**
 - Complete guide to all necessary conditions for minimal logging

Switching Recovery Models

- **When switching to FULL for the first time, the database behaves as if it is in the SIMPLE recovery model until the first full backup is performed**
 - This is called being in 'pseudo-SIMPLE' (log clears on checkpoint)
- **#1 cause of log files growing out of control is someone takes a backup, really switching the database into FULL**
 - And so it will grow forever unless log backups are also being taken
- **Switching between FULL and BULK_LOGGED does NOT break the log backup chain, and does not affect log shipping**
 - This is NOT possible with database mirroring or availability groups
- **Switching to SIMPLE breaks the log backup chain**

Log File Provisioning

- **Only ONE transaction log file is necessary**
 - Log write activity is NOT parallelized so no perf gain from multiple files
 - Jonathan Kehayias proved this and blogged about it (<http://bit.ly/gQbb52>)
 - Only time another log file may be needed is if transaction log runs out of space and there is no other option
- **Isolate transaction log file from data files to avoid possible I/O subsystem contention problems**
 - Potentially place the transaction log on a very fast I/O subsystem like an SSD
- **Choose an appropriate RAID level**
 - RAID 5 not recommended as log is write-heavy, and poor failure protection
 - RAID 1 is good, RAID 10 is better for failure protection

Configuring Transaction Log Auto-Growth

- **Auto-growth should be configured for transaction log files for the emergency case where monitoring fails**
- **Consider that new log space has to be zero-initialized**
 - Takes time and pauses write activity if auto-grow triggered the growth
- **The auto-growth size to use needs to take into account:**
 - Potential for workload delays from having the growth amount set too high
 - Potential for VLF fragmentation from having the growth amount set too low
 - Log generation rate so it may fill again and require more auto-growth
- **Default is 10% prior to SQL Server 2016 (changed to 64MB)**
- **Do not use a percentage as the growth amount will increase over time**

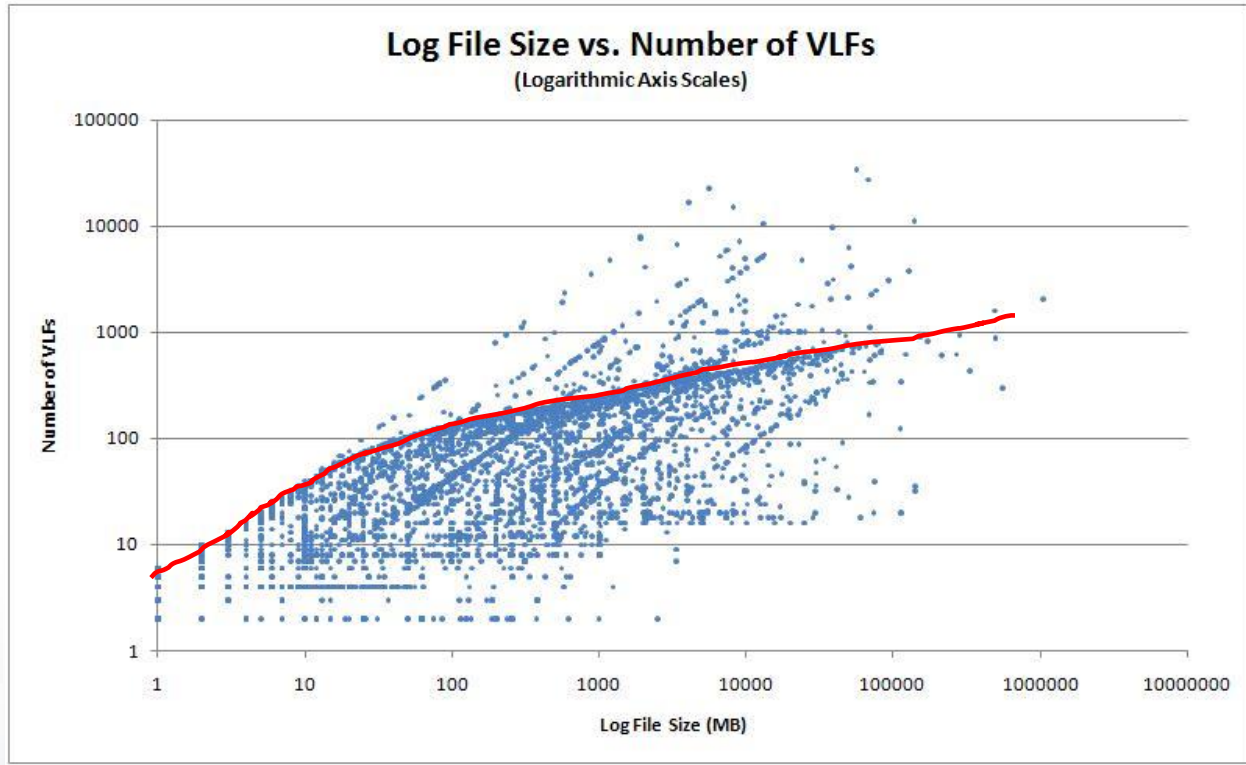
Transaction Log File Shrinking

- **The only way to reduce the transaction log size is DBCC SHRINKFILE**
 - Do not use the EMPTYFILE option as it has no effect
 - Do not use the NOTRUNCATE option as it will not shrink the file
- **Shrinking removes inactive VLFs from the end of the log file**
 - Shrinking does not move or remove log records
 - Transaction log shrinking is completely different from data file shrinking in that no index fragmentation is caused
- **Shrink will terminate when it encounters an active VLF**
- **Do not regularly shrink the transaction log**
 - If it keeps growing again, leave it at the steady-state size
 - Otherwise it is repeatedly having to zero-initialize the newly-added space

VLF Fragmentation: Too Many VLFs?

- **Excessive VLFs ('VLF fragmentation') add overhead to transaction log activities because finding a VLF means traversing the list of VLFs**
 - All transaction log activity can be affected, including crash recovery
 - Log backups and log readers (replication, change data capture, database mirroring, availability groups, restoring on log shipping secondary)
 - Empirical evidence: <http://tinyurl.com/3ur7j5f>
 - Many bugs fixed for crash recovery in all versions from SQL Server 2005+:
 - [KB 2455009](#) and [KB 2028436](#)
- **What is excessive?**
 - Depends on log file size: many thousands are not ok
- **Careful size management is required**

Survey: Transaction Log File Size vs. VLFs



Source: <http://bit.ly/bffg7W>

Removing VLF Fragmentation

- **Execute DBCC LOGINFO to find the number of VLFs**
 - Number of rows = number of VLFs
- **If excessive (> many hundreds or thousands) then need to remove some**
- **Free up log space by first clearing the log**
- **Manually shrink the transaction log file: DBCC SHRINKFILE (logfilename)**
 - If highest active VLF is towards end of the transaction log file, perform another log clearing to make the log manager wrap the log
 - Can be difficult to reach minimum size on a busy production system
- **Modify the transaction log file size to a more appropriate size**
 - ALTER DATABASE dbname MODIFY FILE (NAME = name, SIZE = new_size);
 - Remember to keep the resulting VLF sizes around 512MB or less

Review and References

- Transaction log architecture
- Crash recovery
- Recovery models
- Log file provisioning and maintenance

- Blog: <http://www.sqlskills.com/blogs/paul/category/transaction-log/>
- TechNet article: Understanding Logging and Recovery in SQL Server
 - <http://technet.microsoft.com/en-us/magazine/2009.02.logging.aspx>
- 8-hour Pluralsight online course: <http://bit.ly/W2koLC>

Questions?



Don't forget to complete an online evaluation!

Understanding Logging and Recovery

Your evaluation helps organizers build better conferences and helps speakers improve their sessions.

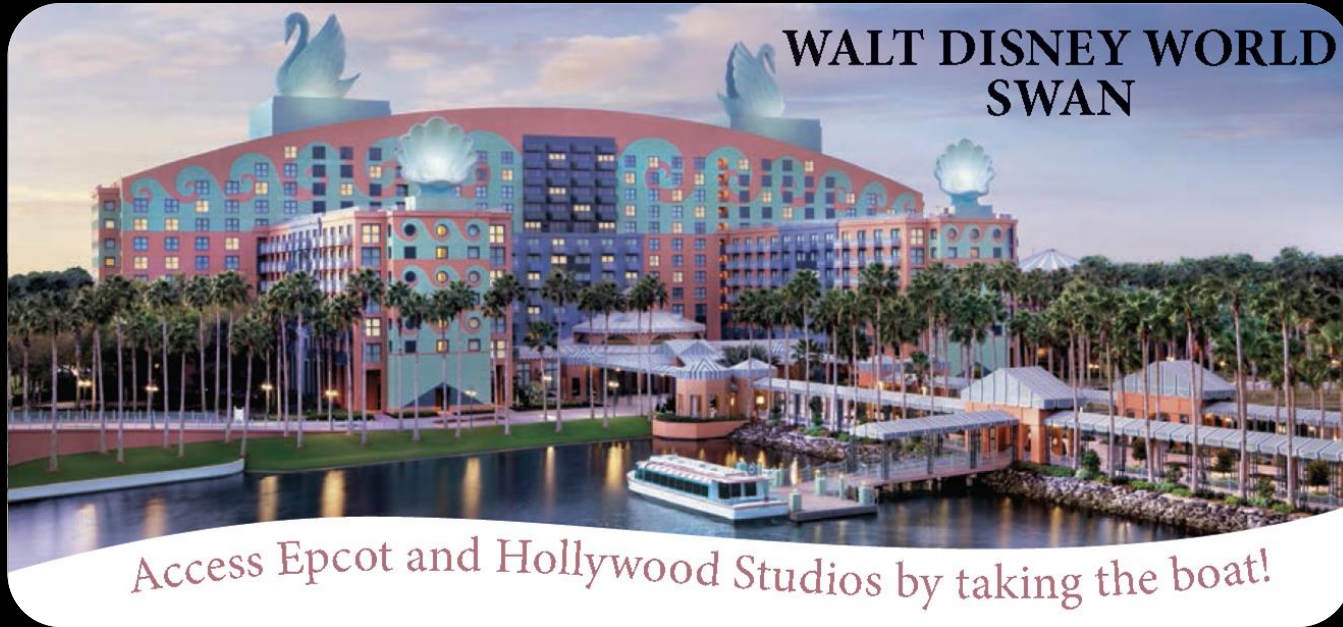


SQL
intersection

Thank you!

Save the Date!

www.SQLintersection.com



2018

Mar 25-28

We're back in Orlando!



Leave the every day behind and enter a world of wonder and enchantment at the Walt Disney World® Resort. Located in the heart of the most magical place on earth, the Walt Disney World Swan and Dolphin Resort provides a truly extraordinary backdrop for our event! Beautiful tropical landscaping, tranquil waterways, and classic art and architecture work together to create a stunning landmark!