Visualize Your Transaction Log

Brian Hansen
brian@tf3604.com
@tf3604

SQL Saturday #839
Virginia Beach, Virginia
8 June 2019

Brian Hansen


brian@tf3604.com


@tf3604.com

children.org

- 20+ Years working with SQL Server
  - Development work since 7.0
  - Administration going back to 6.5
  - Fascinated with SQL internals

www.tf3604.com/poshadmin

Agenda

- Purpose of the transaction log
- Organization of the transaction log
- Flushing & clearing the log / checkpoints
- Rollback operations
- VLF fragmentation
- Log monitoring

Purpose of the Transaction Log

- Primary purposes
  - Durability
    - Write-ahead logging
    - Crash recovery / restore operations
  - Atomicity
  - Thought experiment
    - What would SQL be like without a transaction log?
- Secondary purposes
  - Log reader (replication, CDC)
  - Mirroring / Availability Groups / log shipping
  - Snapshots

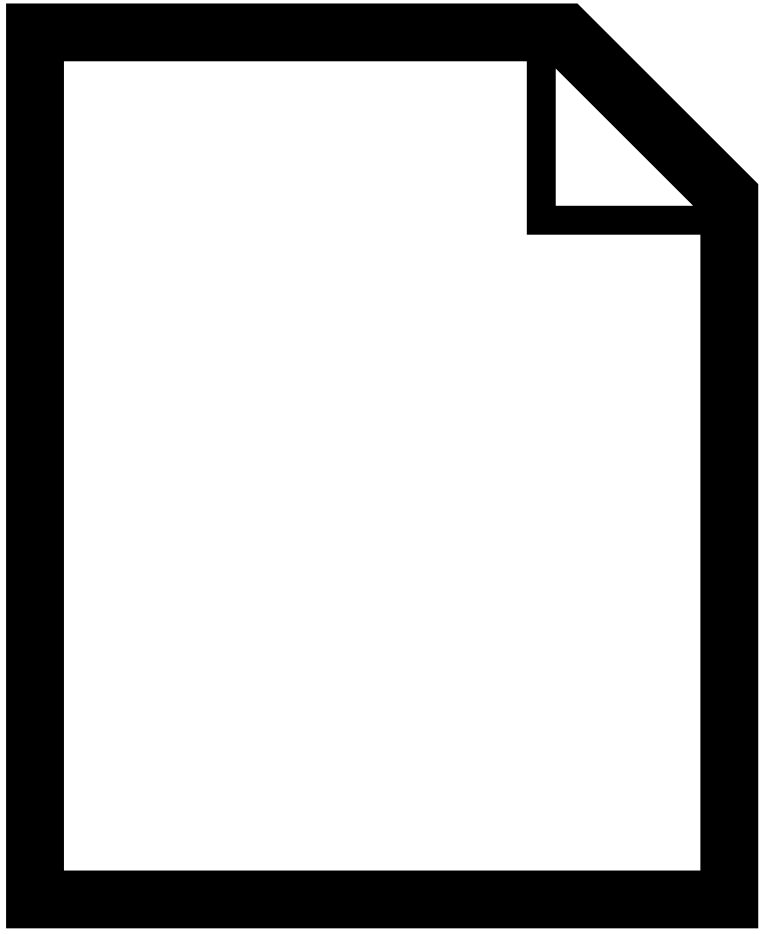
What Goes in the Transaction Log?

- Everything that modifies the state any database in SQL
  - Includes data to **redo** an operation
  - Includes data to **undo** an operation
- Very limited exceptions for some tempdb operations

Physical vs Logical Log File

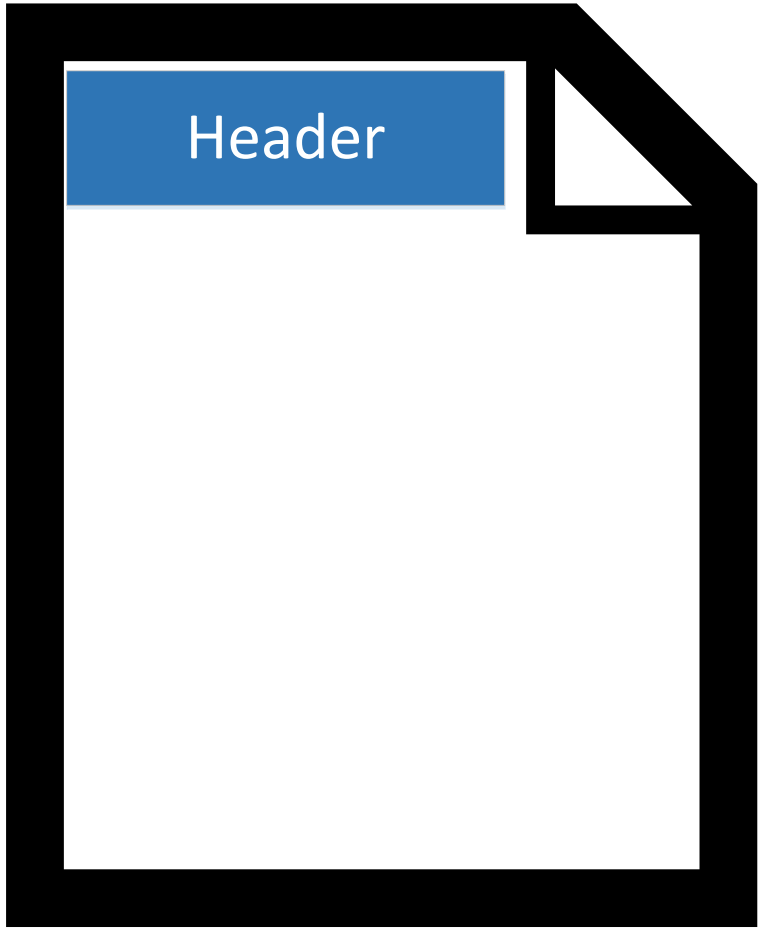
- Logical Log File
  - Always growing
  - Write once / read many
    - After being written, log records are **never** changed
- Physical Log File
  - Divided into virtual log files (VLFs)
  - Only grows when full (or manually grown)
  - VLFs are inactivated when possible and over-written

Organization of the Transaction Log



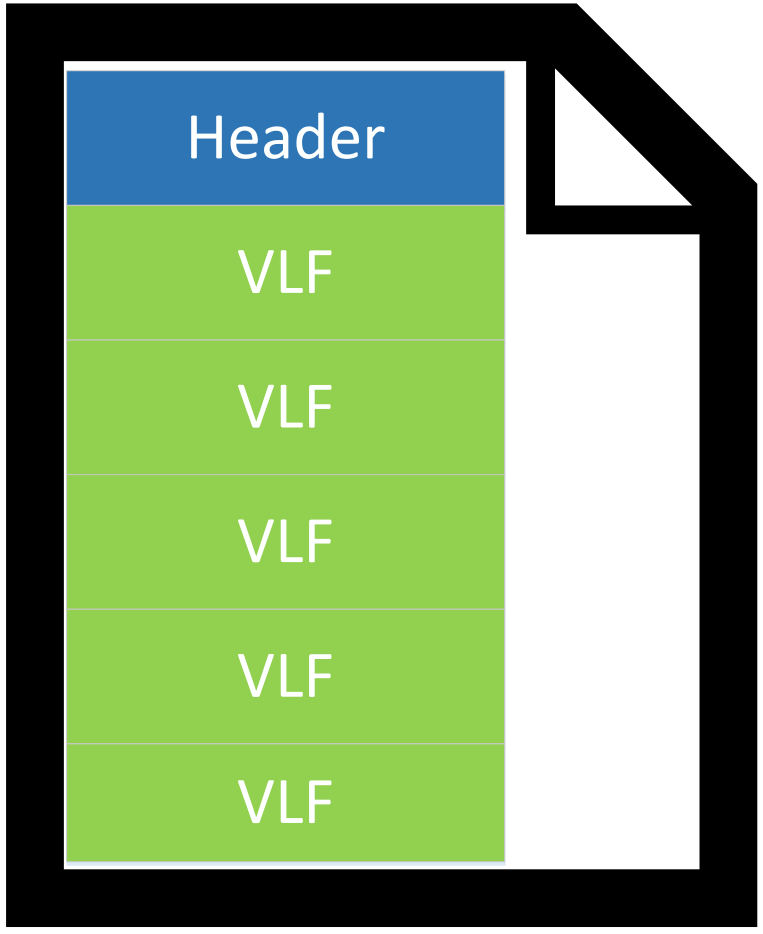
- The Transaction Log is just a file …

Organization of the Transaction Log



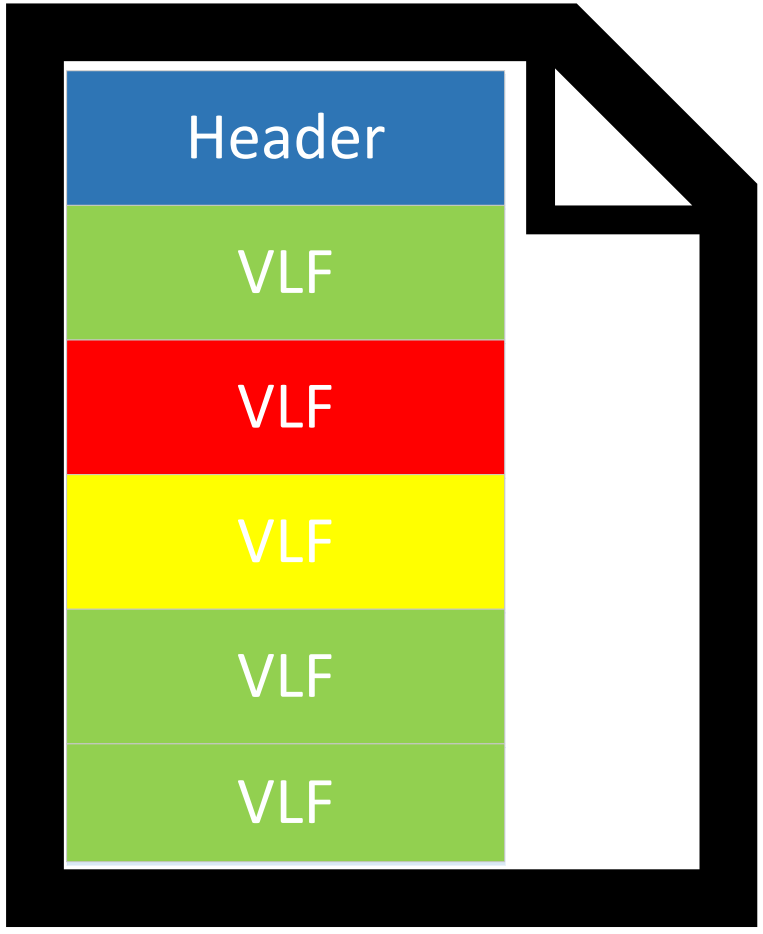
- The Transaction Log is just a file …
- With a bit of header information …

Organization of the Transaction Log



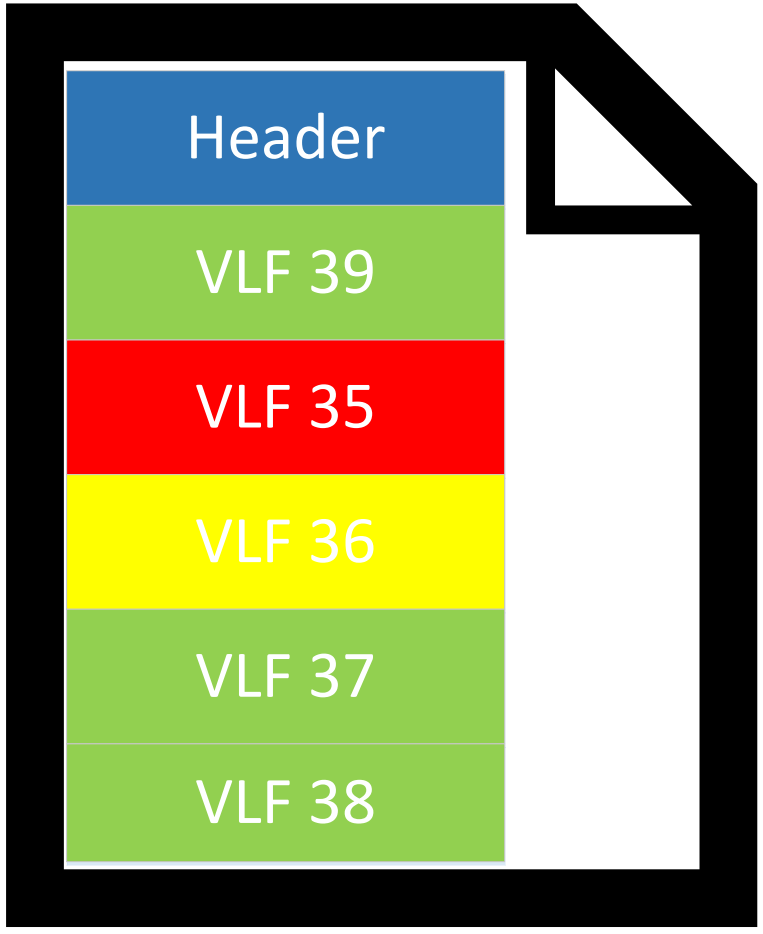
- The Transaction Log is just a file …
- With a bit of header information …
- Then divided into Virtual Log Files.
  - Not necessarily of equal size

Virtual Log Files



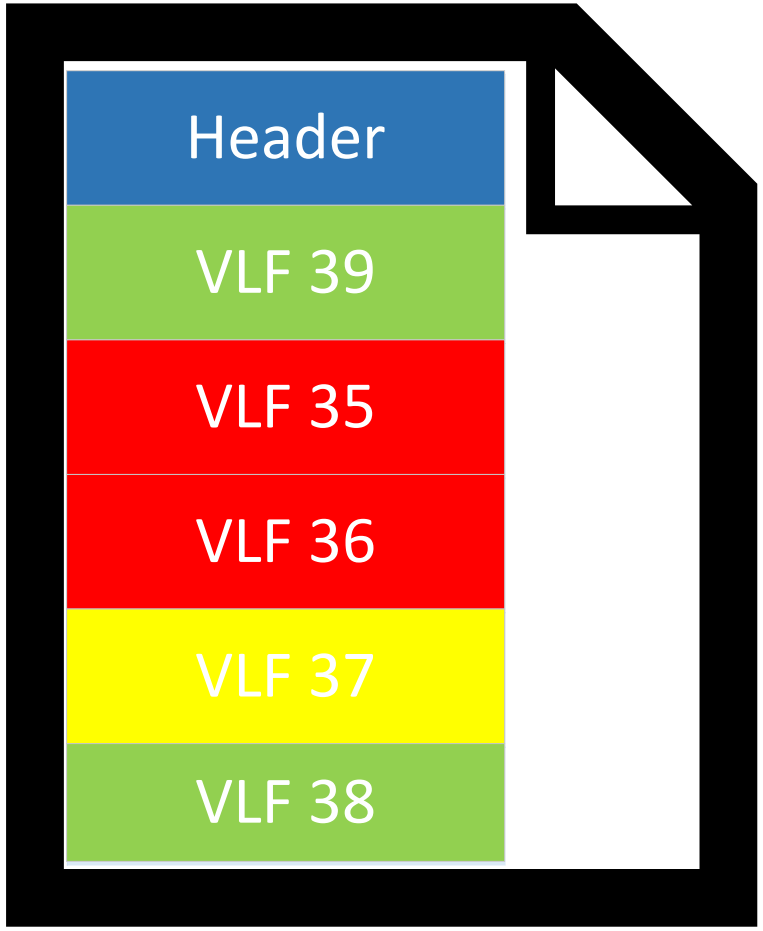
- VLFs can be in one of several statuses:
  - Inactive (never used)
  - Inactive (previously used)
  - Active (current)
  - Active (not usable)
- Only one VLF is current at a time.

Virtual Log Files



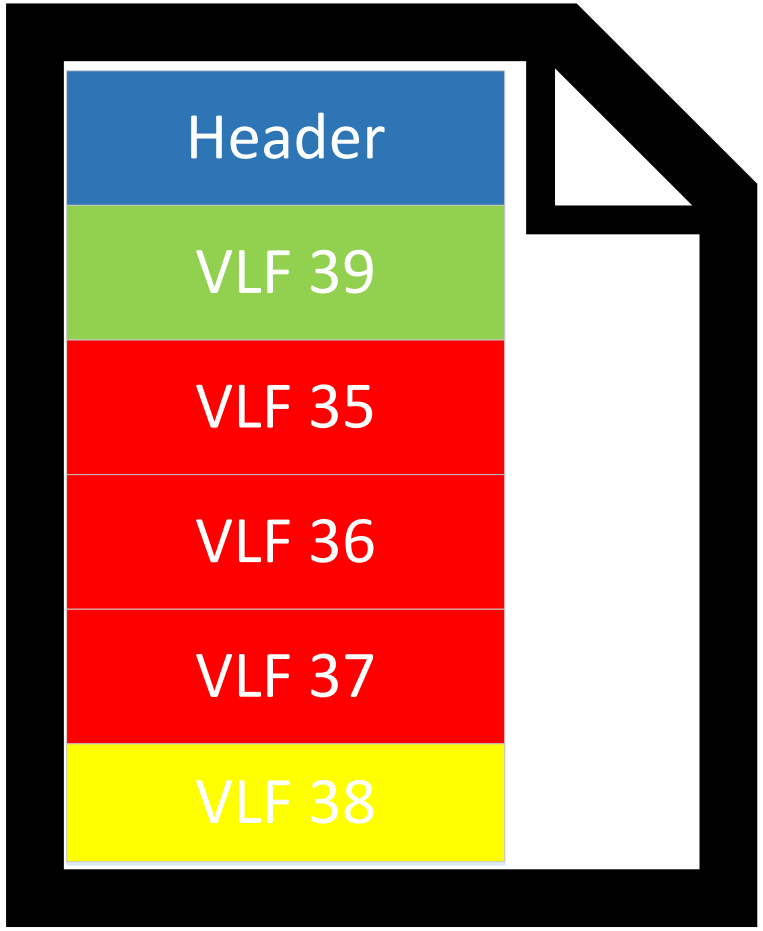
- VLFs can be in one of several statuses:
  - Inactive (never used)
  - Inactive (previously used)
  - Active (current)
  - Active (not usable)
- Only one VLF is current at a time.
- VLFs are numbered.

Virtual Log Files



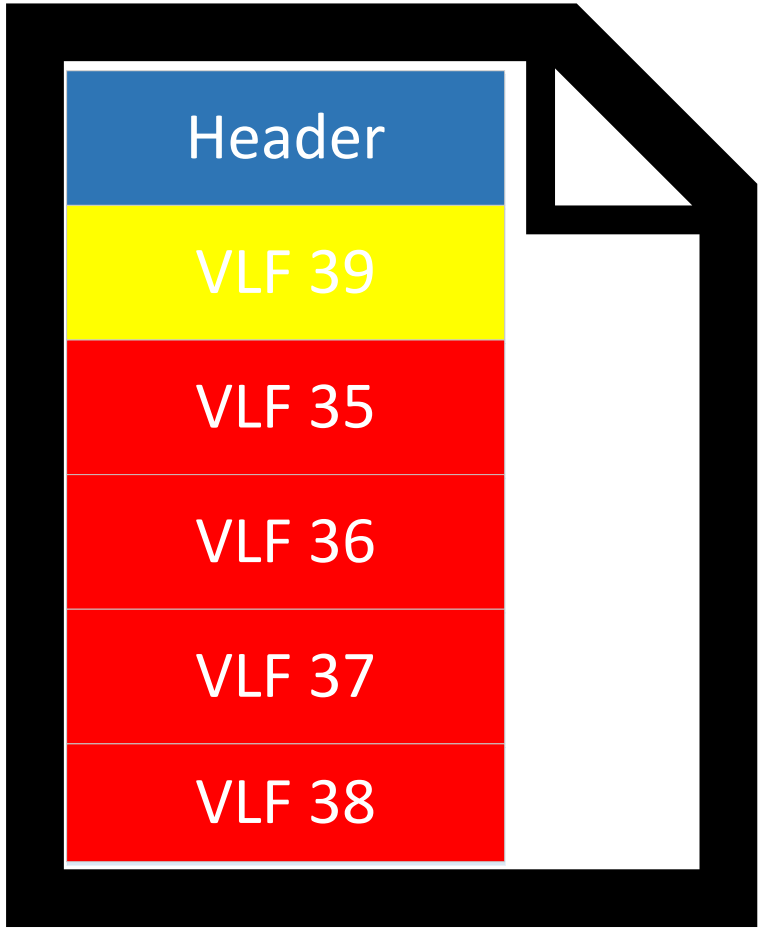
- As more records are added to the log, additional VLFs are allocated.

Virtual Log Files



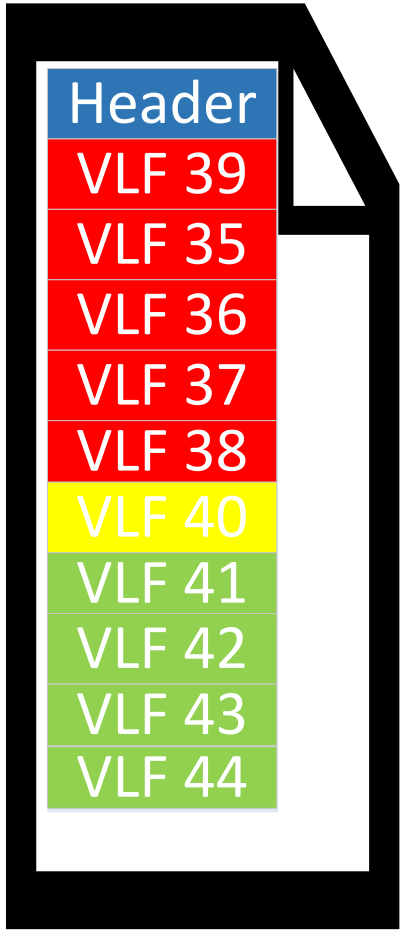
- As more records are added to the log, additional VLFs are allocated.

Virtual Log Files



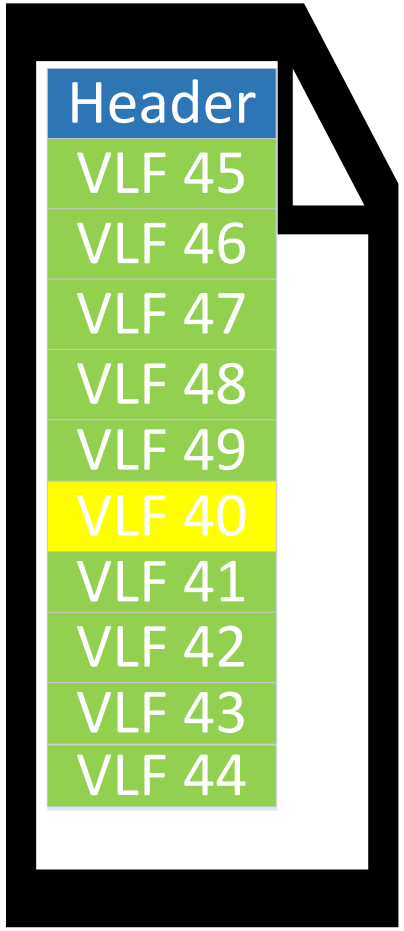
- As more records are added to the log, additional VLFs are put in use.
- Writing to the log is circular so long as VLF are available.
- What happens next?

Virtual Log Files



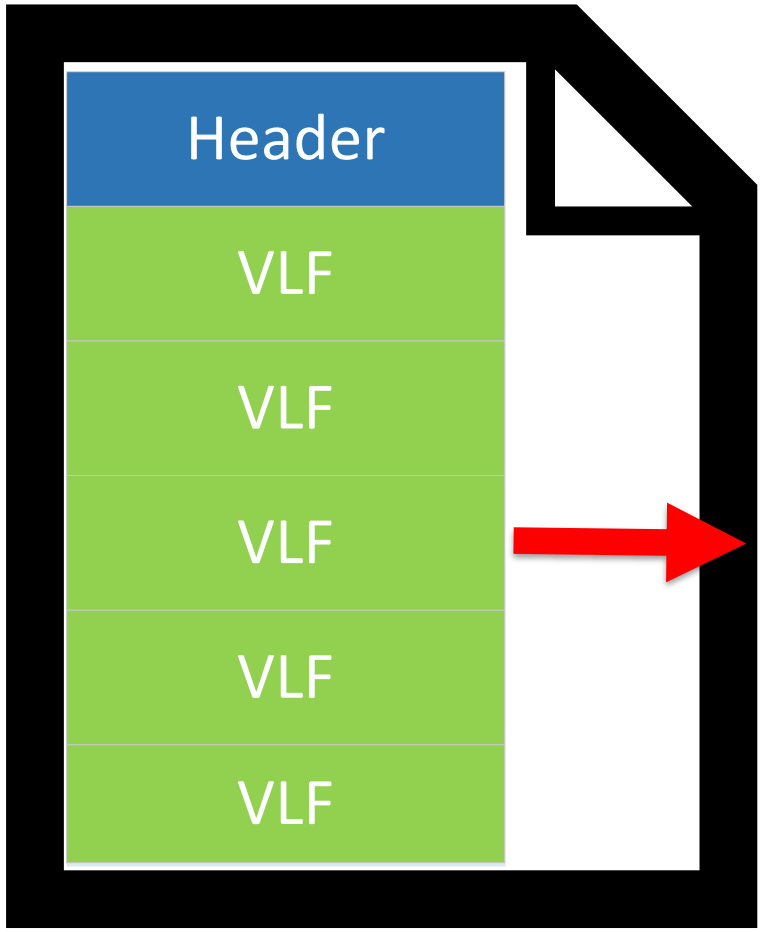
- The log file has to grow
- More VLFs are added

Virtual Log Files



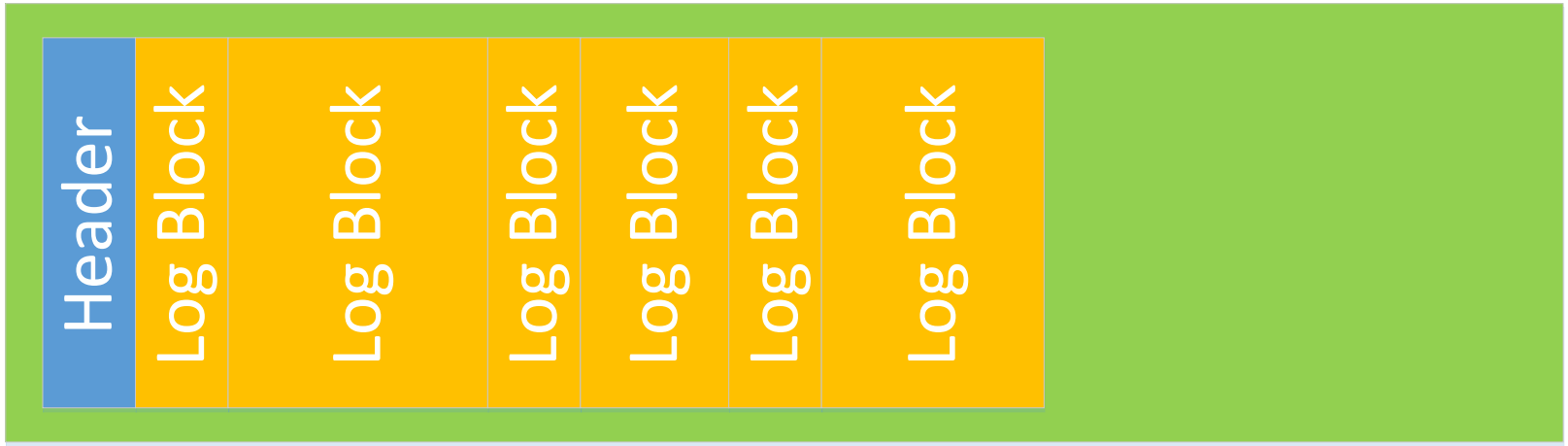
- The log file has to grow
- More VLFs are added
- Eventually the log will be “truncated” or “cleared”

Organization of the Transaction Log



- VLFs are also structured

VLF Detail



- Again there is a header
- Then a series of log blocks
  - In 512 byte increments up to 60K in size

Log Block Detail

- As expected, starts with a header
- Then a series of log records
  - Completely variable in size
- And an index to the log records (slot array)

Log Record Detail

- Of course, a header
  - Record type, transaction ID, length, pointer to previous transaction record, etc.
- Payload
  - Before/after image of changes

Log Sequence Number

- Each log record can be uniquely identified by its Log Sequence Number (LSN)
- An LSN is composed of three parts
  - VLF number
  - Log Block offset (512-byte chunks, not necessarily contiguous)
  - Log Record number (slot number)
- The LSN is, in a very real way, a pointer into the (logical) log file

LSN Representations

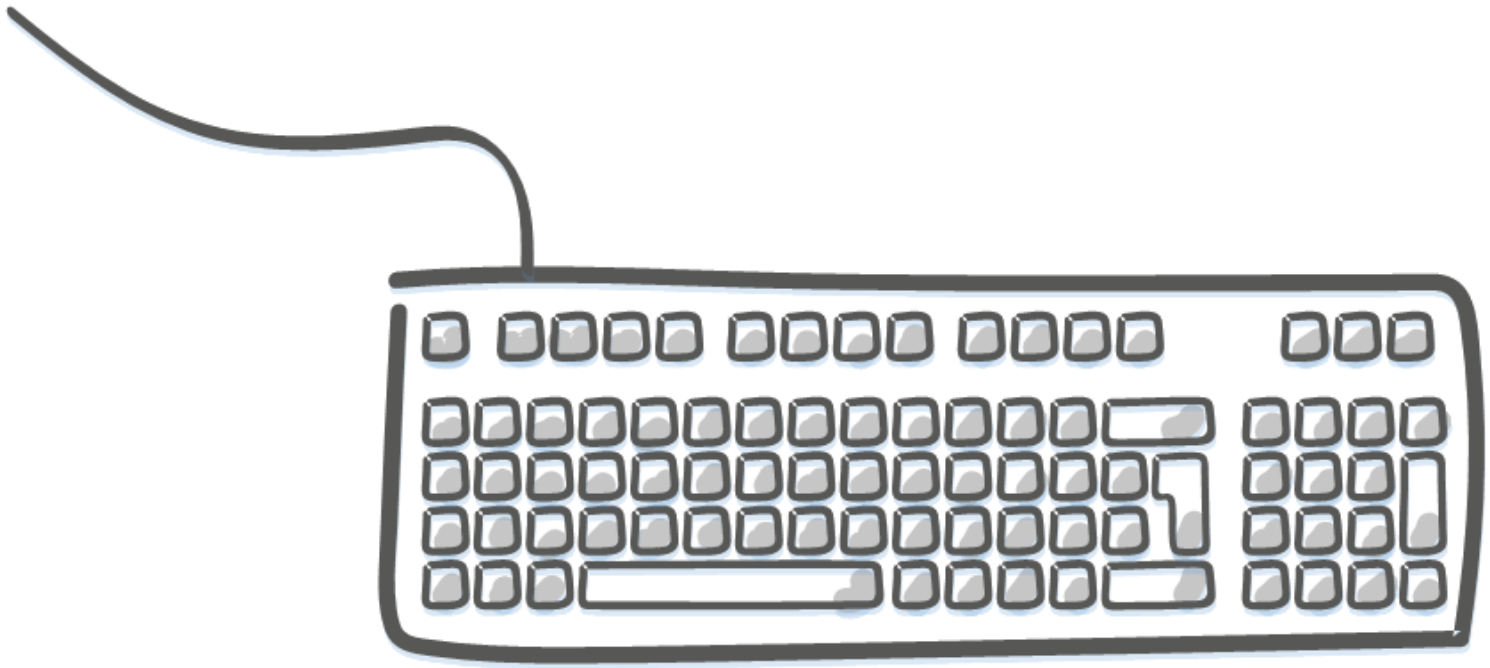
Four common ways to express an LSN

Format	Example	Common uses
Colon-separated (hexadecimal)	000001c0:0000006b:0005	Log management
Hexadecimal	0x000001c00000006b0005	Change data capture
Decimal	448000000010700005	Backup
Colon-separated (decimal)	448:107:5	Input to fn_dblog

These four LSNs are equivalent

Demo

LSN Converter



DBCC LOGINFO('db_name')

Returns one row per VLF

	RecoveryUnitId	FileId	FileSize	StartOffset	FSeqNo	Status	Parity	CreateLSN
1	0	2	253952	8192	1203	0	64	0
2	0	2	253952	262144	1204	0	64	0
3	0	2	253952	516096	1205	0	64	0
4	0	2	278528	770048	1206	0	64	0
5	0	2	524288	1048576	1207	0	64	151000000004800104
6	0	2	524288	1572864	1208	0	64	151000000004800104
7	0	2	524288	2097152	1209	0	64	151000000004800104
8	0	2	524288	2621440	1210	0	64	151000000004800104
9	0	2	524288	3145728	1211	0	64	154000000084800419
10	0	2	524288	3670016	1202	0	128	154000000084800419
11	0	2	524288	4194304	1212	2	128	154000000084800419
12	0	2	524288	4718592	0	0	64	154000000084800419

sys.dm_db_log_info(db_id)

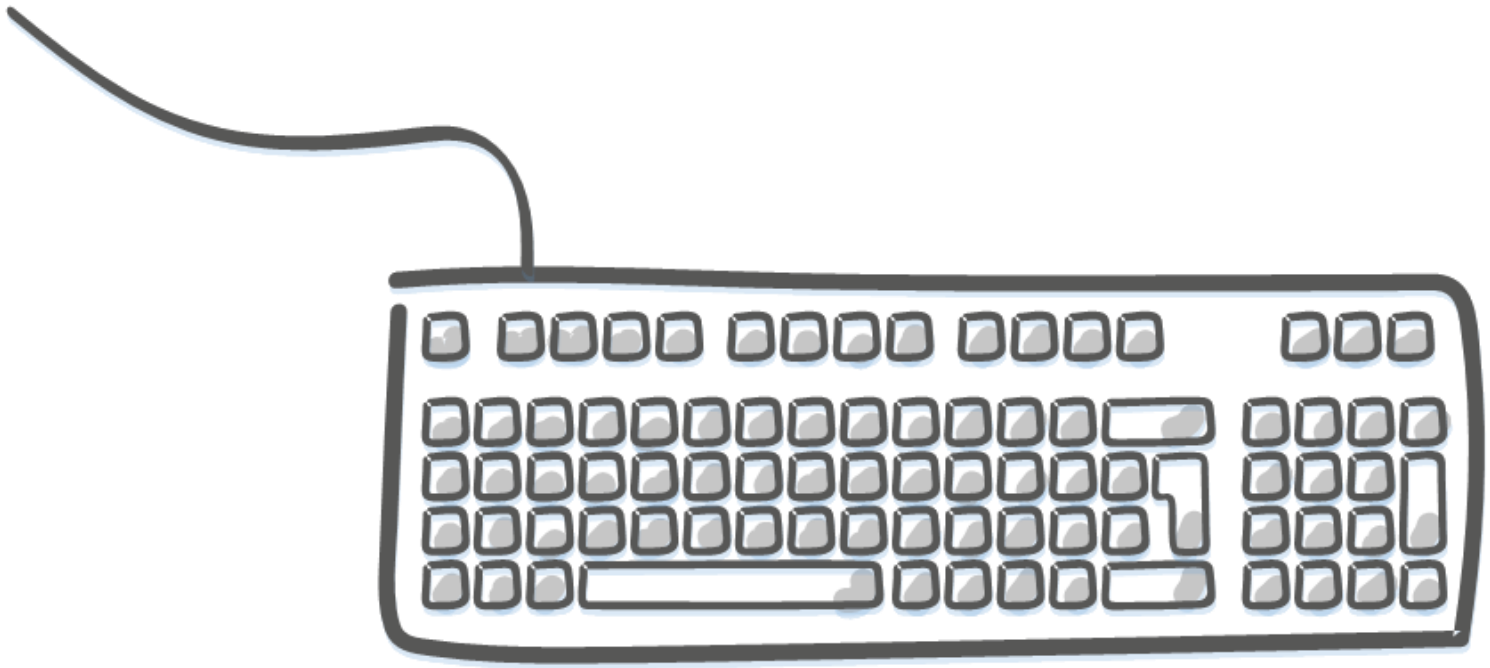
Documented, supported version of DBCC LOGINFO
SQL Server 2016 SP2+
Subtle differences from DBCC LOGINFO

	database_id	file_id	vlf_begin_offset	vlf_size_mb	vlf_sequence_number	vlf_active	vlf_status	vlf_parity	vlf_first_lsn	vlf_create_lsn
1	6	2	8192	2.43	286	1	2	128	0000011e:00000010:0001	00000000:00000000:0000
2	6	2	2564096	2.43	285	0	0	64	00000000:00000000:0000	00000000:00000000:0000
3	6	2	5120000	2.43	0	0	0	128	00000000:00000000:0000	00000000:00000000:0000
4	6	2	7675904	2.67	0	0	0	64	00000000:00000000:0000	00000000:00000000:0000

Demo

DBCC LOGINFO

+ Log File Visualizer



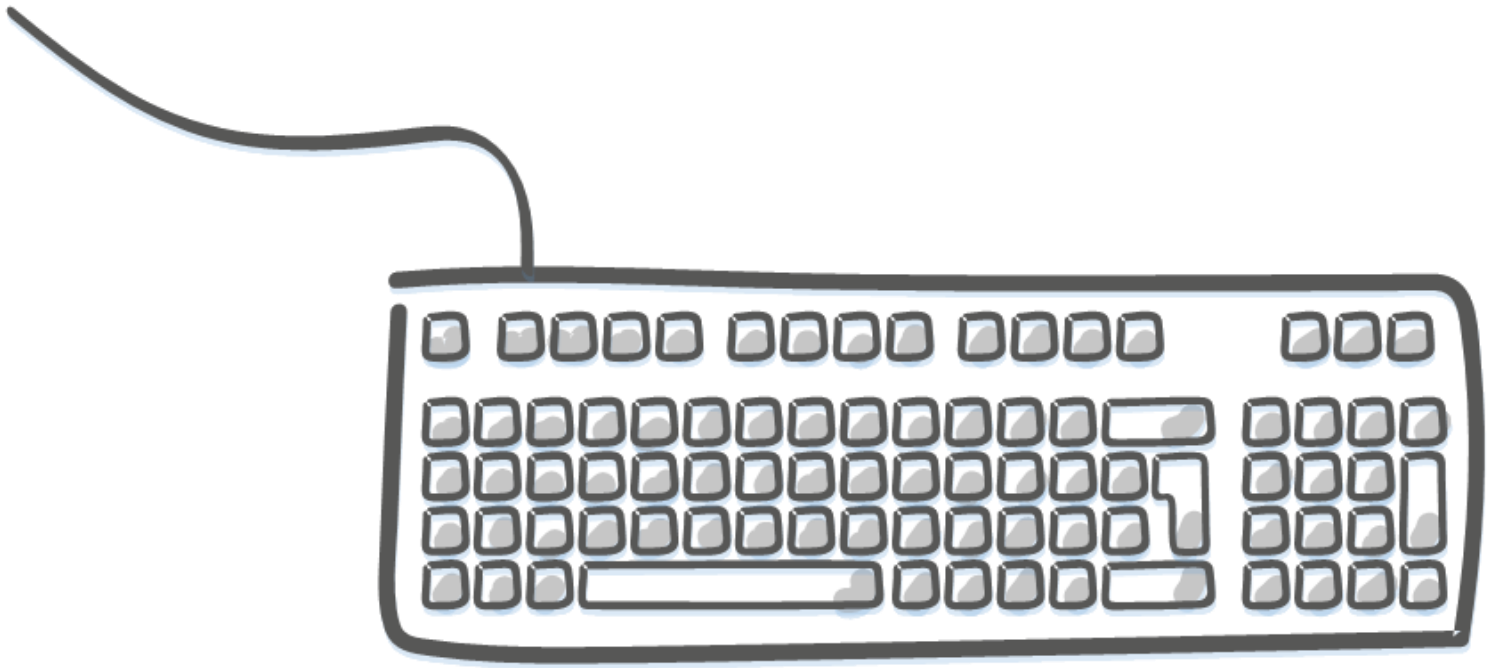
fn_dblog(start_lsn, end_lsn)

Returns one row per log record

	Current LSN	Operation	Context	Transaction ID	LogBlockGeneration	Tag I
1	000001c0:00000044:0049	LOP_BEGIN_CKPT	LCX_NULL	0000:00000000	0	0x00
2	000001c0:00000061:0001	LOP_XACT_CKPT	LCX_BOOT_PAGE_CKPT	0000:00000000	0	0x00
3	000001c0:00000062:0001	LOP_END_CKPT	LCX_NULL	0000:00000000	0	0x00
4	000001c0:00000063:0001	LOP_SET_BITS	LCX_DIFF_MAP	0000:00000000	0	0x00
5	000001c0:00000063:0002	LOP_BEGIN_XACT	LCX_NULL	0000:0000dd29	0	0x00
6	000001c0:00000063:0003	LOP_MODIFY_COLUMNS	LCX_CLUSTERED	0000:0000dd29	0	0x00
7	000001c0:00000063:0004	LOP_MODIFY_COLUMNS	LCX_CLUSTERED	0000:0000dd29	0	0x00
8	000001c0:00000063:0005	LOP_COMMIT_XACT	LCX_NULL	0000:0000dd29	0	0x00
9	000001c0:00000065:0001	LOP_FILE_HDR_MODIFY	LCX_FILE_HEADER	0000:00000000	0	0x00
10	000001c0:00000065:0002	LOP_MODIFY_ROW	LCX_BOOT_PAGE_CKPT	0000:00000000	0	0x00
11	000001c0:00000065:0003	LOP_MODIFY_ROW	LCX_BOOT_PAGE_CKPT	0000:00000000	0	0x00
12	000001c0:00000067:0001	LOP_BEGIN_XACT	LCX_NULL	0000:0000dd2a	0	0x00
13	000001c0:00000067:0002	LOP_MODIFY_ROW	LCX_CLUSTERED	0000:0000dd2a	0	0x00
14	000001c0:00000067:0003	LOP_PREP_XACT	LCX_NULL	0000:0000dd2a	0	0x00
15	000001c0:00000067:0004	LOP_COMMIT_XACT	LCX_NULL	0000:0000dd2a	0	0x00
16	000001c0:00000069:0001	LOP_FORGET_XACT	LCX_NULL	0000:0000dd2a	0	0x00
17	000001c0:00000069:0002	LOP_BEGIN_XACT	LCX_NULL	0000:0000dd2b	0	0x00
18	000001c0:00000069:0003	LOP_MODIFY_COLUMNS	LCX_CLUSTERED	0000:0000dd2b	0	0x00

Demo

fn_dblog



Related command/function

- DBCC SQLPERF(LOGSPACE)
  - Log size, percent used per database
- fn_dump_dblog
  - Similar to fn_dblog, but reads from backup file

Checkpoint

- Process of writing dirty pages from the buffer pool to disk
  - Irrespective of transaction completion

Checkpoint Types

- Automatic
  - Period background thread
  - Instance-wide [`sp_configure 'recovery interval (min)', 2`]
- Indirect (2012+)
  - Database-specific
  - [`alter database myDB set target_recovery_time = 2 minutes`]
  - Off by default in 2012, 2014; on by default in 2016+
- Internal
  - During operations such as backup, snapshots, shutdown
- Manual
  - CHECKPOINT command

Checkpoint Process

- Write to log: checkpoint start
  - Also info about any uncommitted transactions
  - Flush the log
- Identify dirty pages; write to disk
- Update boot page with LSN corresponding to checkpoint start
- (If SIMPLE recovery) clear the log
- Write to log: checkpoint finish

Flushing the Log

- Flushing = closing a log block
- Triggers
  - 60K limit reached
  - Transaction commits
  - Transaction rollbacks
  - Checkpoint

Recovery Models

- Impacts how SQL logs changes
  - Simple
  - Full
  - Bulk-logged

Simple Recovery Model

- Commonly used for test systems or low-volume production systems
  - What is your recovery point objective?
- All changes logged, but can be “discarded” on commit
- Can only recover to the latest full backup

Full Recovery Model

- Probably the most common recovery model for production systems
  - What is your recovery point objective?
- Log records must be kept until log backup completed
- Can recover to an arbitrary point in time

Bulk-logged Recovery Model

- Not frequently used, perhaps temporarily during maintenance windows
  - What is your recovery point objective?
- Similar to full model, but some changes are only “noted” rather than fully logged
- Log backups still include all changes
- Point-in-time recovery not possible

Clearing* the Log (aka Truncating*)

- Marks unneeded portions of log as inactive
- Triggers:
  - Simple recovery**: Checkpoint
  - Full/bulked-log: Log Backup
  - Change from Full or Bulked Logged to Simple***
- Why can't the log clear?
  - Pending log backup
  - Active replication / CDC / AG / mirroring
  - Long-running transaction
  - See `sys.databases.log_reuse_wait_desc`

* Horribly misnamed! This process clears nothing and truncates nothing.

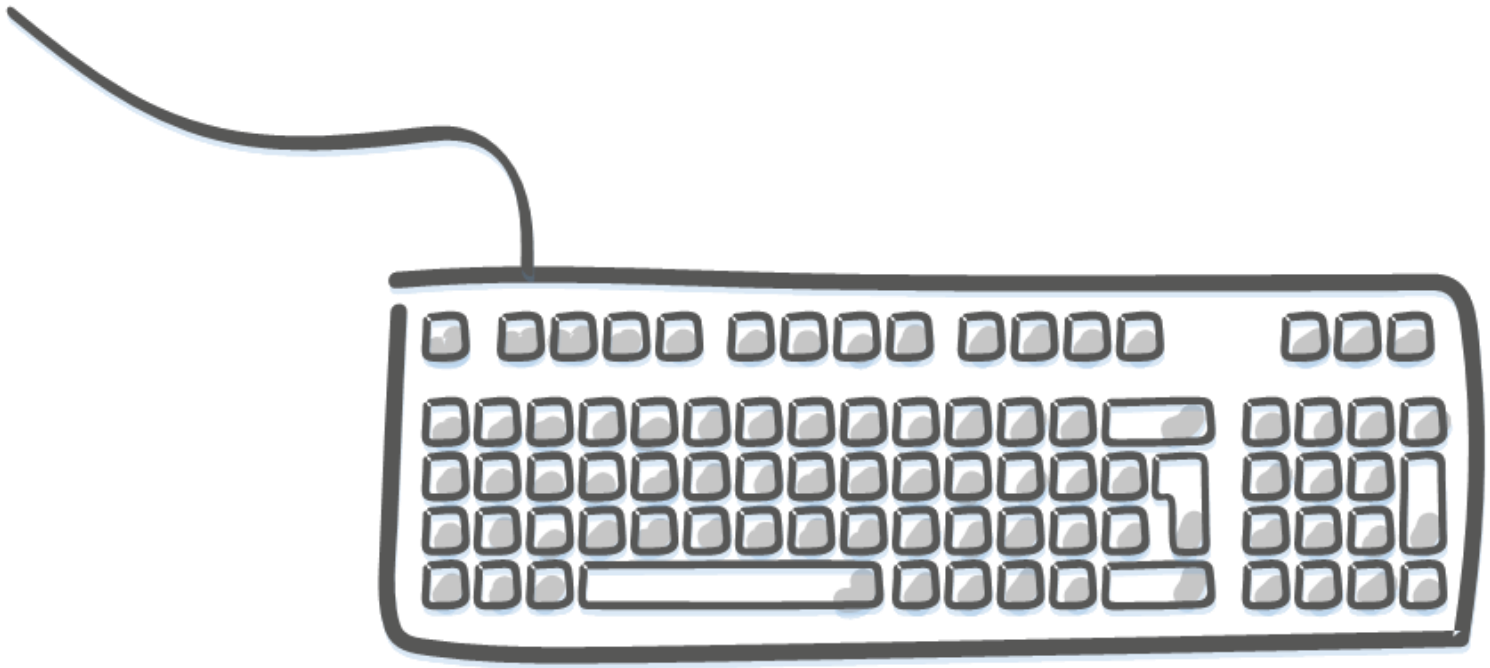
** More technically, when in "auto-truncate" mode.

*** But this breaks the backup chain!

Demo

Simple recovery

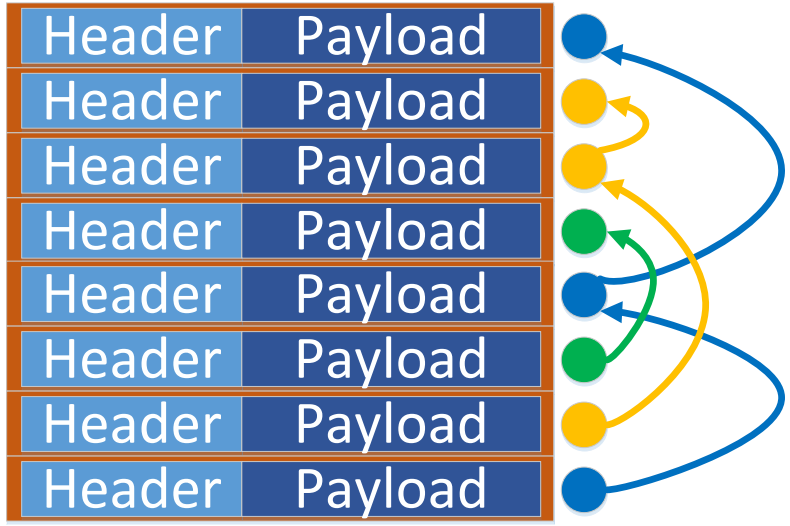
Full recovery



Rolling Back a Transaction

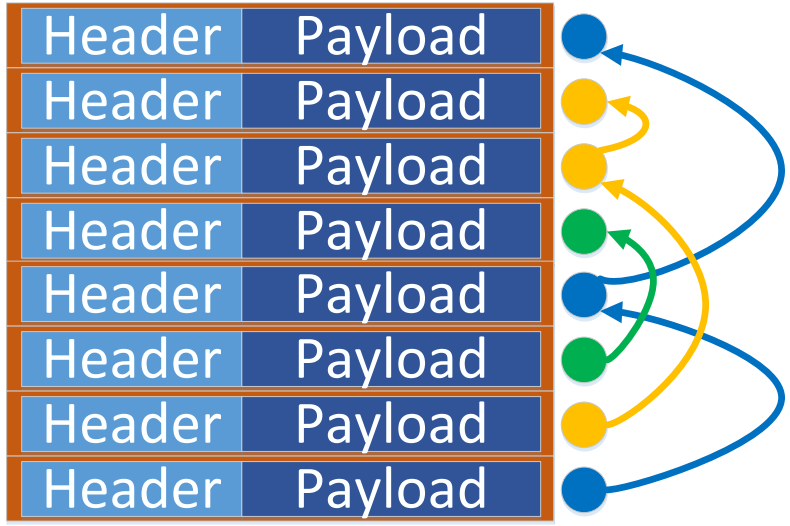
- When a transaction cannot complete, it must rollback
  - ROLLBACK TRANSACTION command
  - Connection is abandoned
    - Network failure, KILL, severe errors, client crash
  - Non-graceful shutdown of SQL (crash recovery)
  - Restore operations
- Rollback operations are single-threaded

Rolling Back a Transaction



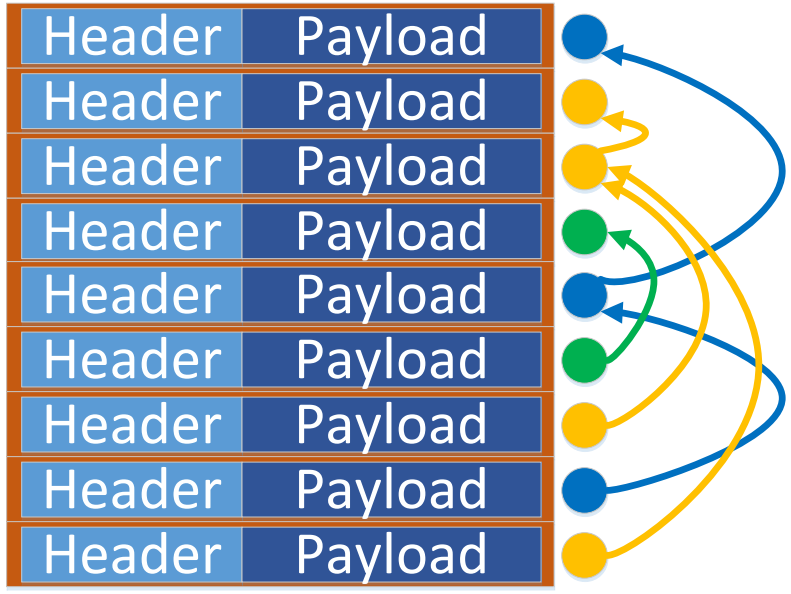
- Log records form a reverse linked list of operations within a transaction.
- Let's suppose the yellow transaction needs to roll back.
- The first record is for "begin transaction."

Rolling Back a Transaction



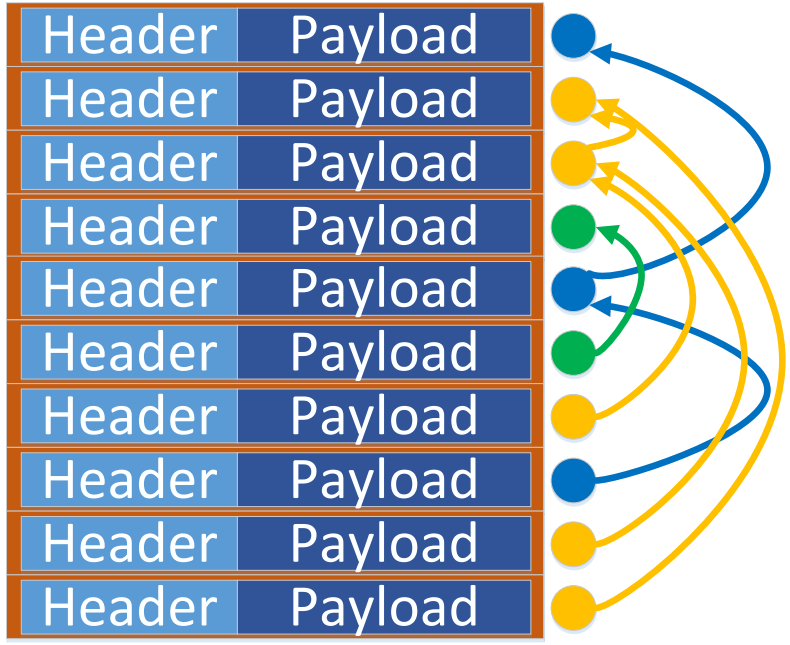
- SQL Server finds the last log record for the transaction.
- SQL reverses the operation in the buffer pool.

Rolling Back a Transaction



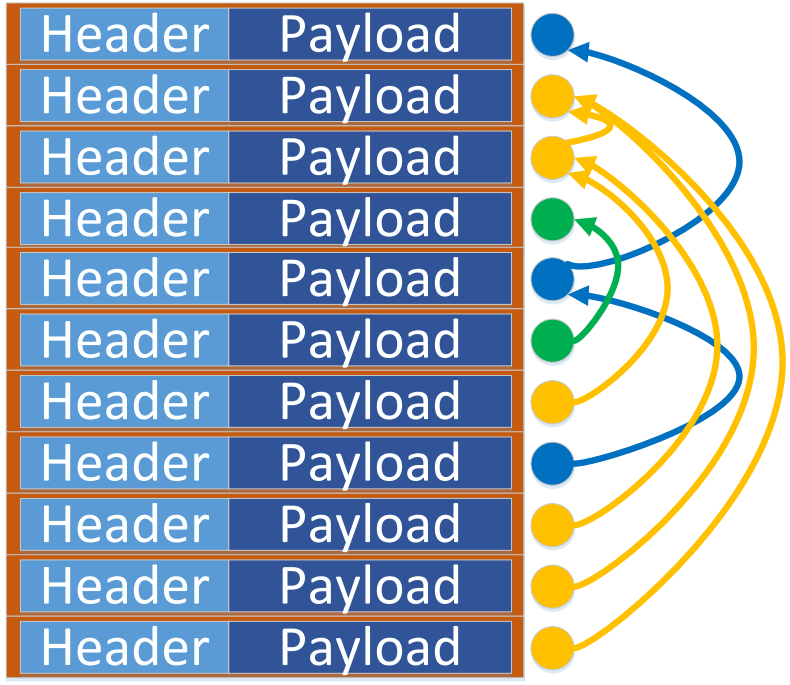
- Creates a new log record indicating that the operation was undone. This is called a "Compensation" record (or "anti-operation").
- This record then points back to the second-to-last record.

Rolling Back a Transaction



- The second to last operation is undone, and a compensation record is written that points back to the first record (the “begin transaction”).

Rolling Back a Transaction



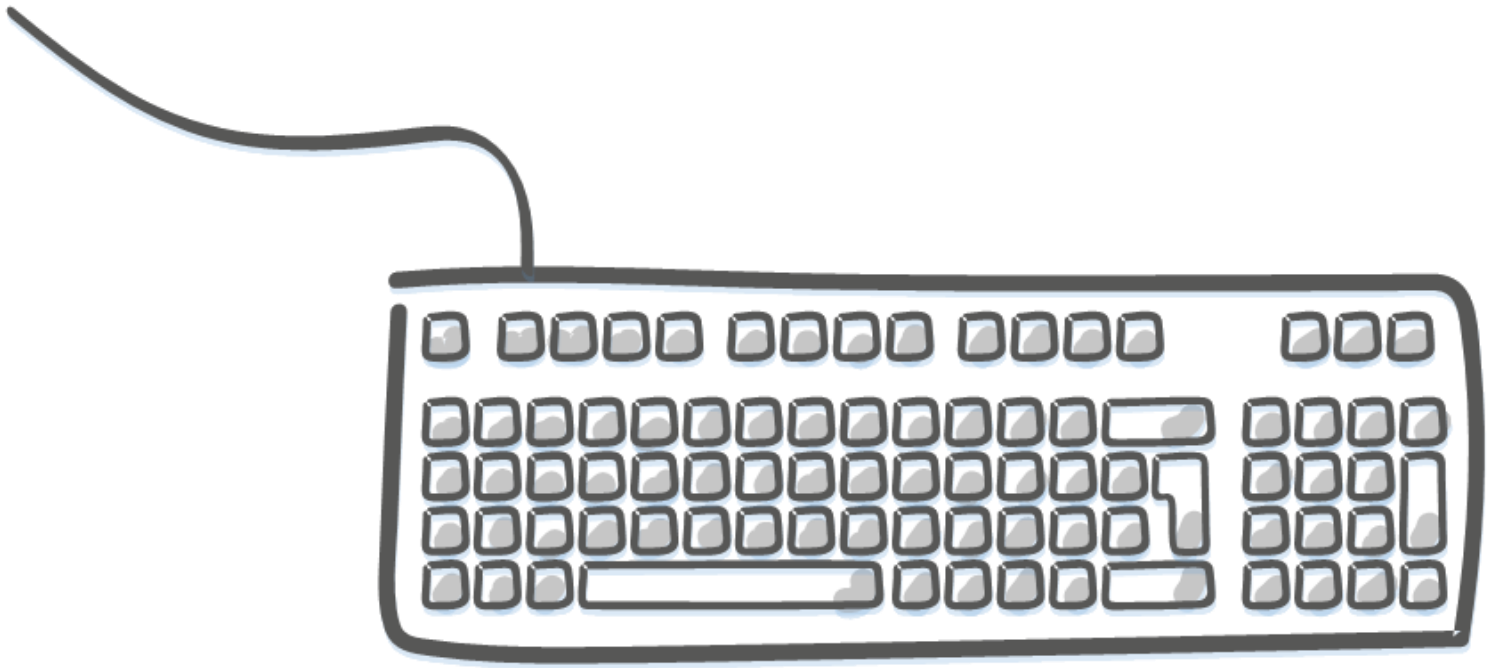
- Finally, an “abort transaction” log record is written. It also points back to the “begin transaction” record.

Rolling Back a Transaction

- Key takeaways:
  - Rollback operations generate log records
  - As the initial operations are performed, SQL Server will “reserve” log space to ensure that a rollback is possible.
    - Very large DML operations will reserve a lot of log space (and will prevent the log from clearing while in process). Often better to split up into smaller transactions.

Demo

Rollback operations



Creating new VLFs

My transaction log grew. How many VLFs?

Log growth size	New VLFs created
1 to 64 MB	4
64 MB to 1 GB	8
Greater than 1 GB	16

Special case for SQL 2014+

- Compute current log size / growth amount
- If greater than 8, add only 1 new VLF

VLF Trade-Offs

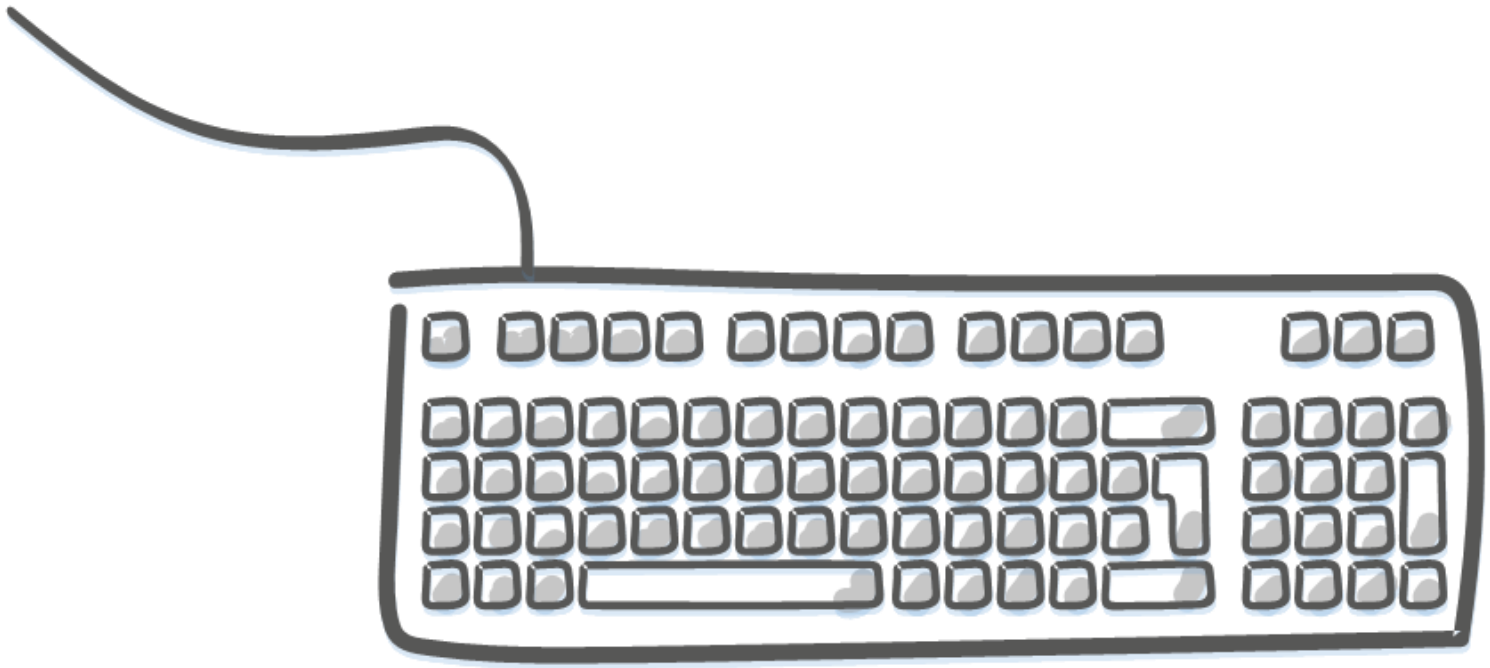
- Too many VLFs create performance problems ("VLF Fragmentation")
  - Slows noticeably any time log is read
    - Start-up time for database, log reader, backup & restore, etc.
  - But smaller VLFs are faster to allocate (zero-init)
- Too few VLFs also create performance problems
  - Clearing the log, especially when long-running transactions are happening

Pre-Allocating the Log

- Why?
  - Eliminate VLF fragmentation
  - Avoid log growth during user operations
    - Can be time-consuming due to zero-initialization
- However, plan for auto-growth
  - Set reasonable auto-growth parameters
  - Fixed growth amount, not percentage

Demo

VLF fragmentation



Controlling VLFs

- See the MS Tiger Team solution
  - A bit of a hammer – it generates scripts for all databases on the instance
  - Review the generated script before running it

https://github.com/Microsoft/tigertoolbox/tree/master/Fixing-VLFs

Log Monitoring

- Watch your VLF count
- Monitor log size over time
- Set SQL Alerts on:
  - Severity 17 errors (will alert on log full)
  - Error 5145

    ```
    Autogrow of file '…' in database '…' was cancelled by user or timed out after xx milliseconds.
    ```

  - Error 5144

    ```
    Autogrow of file '…' in database '…' took xx milliseconds.
    ```

Log Monitoring, continued

- PerfMon counters
  - One row per counter per database (plus rollup)
  - Paul Randal explains what to look for.

```
select object_name, counter_name, instance_name, cntr_value
from sys.dm_os_performance_counters
where counter_name in ('Log Growths', 'Log Shrinks', 'Percent Log
Used', 'Log Flush Waits/sec', 'Log Bytes Flushed/sec', 'Log
Flushes/sec');
```

Thank You

This presentation and supporting materials can be found at www.tf3604.com/log.

- Slide deck
- Scripts
- Sample database
- SQL Server Log File Visualizer & LSN Converter binaries & source

brian@tf3604.com • @tf3604