

Module 1: Understanding Databases

Designing for Performance
Kimberly L. Tripp

Overview

- **Creating Databases**
- **Database Structures**
- **The Anatomy of a Data Modification**
 - **From Data Access**
 - **Locking**
 - **Write-ahead Logging**
- **Data and Log File Placement**
- **How the Transaction Log Works**
- **Optimizing Log Activity**

Creating Databases

- Inherit initial properties from model database
- Changes to model will affect ALL databases created AFTER the changes made.

NOTE: TempDB is re-created on each system restart. Might be a good idea for data types that are needed for temp tables/objects.

- Defines logical file names
- Defines physical location
- Defines autogrowth settings

Creating Databases

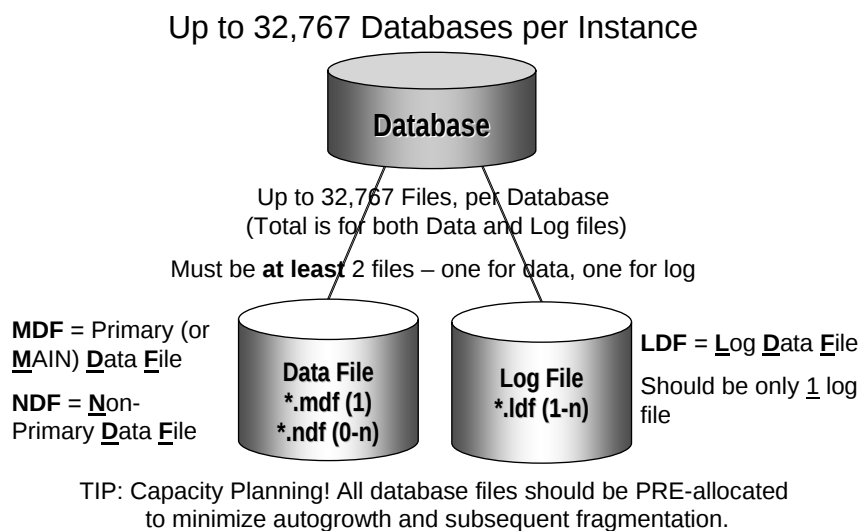
```
CREATE DATABASE [JunkDB]
ON PRIMARY
(NAME = N'JunkDBData',
FILENAME = N'physical filename',
SIZE = 10,          -- kb, mb, gb, tb
FILEGROWTH = 10%, -- kb, mb, gb, tb or %
MAXSIZE = 15)      -- kb, mb, gb, tb
LOG ON
(NAME = N'JunkDBLog',
FILENAME = N'physical filename',
SIZE = 10,
FILEGROWTH = 10%
MAXSIZE = UNLIMITED)
```

- Default for filegrowth is 10%
- Default for MAXSIZE is UNLIMITED

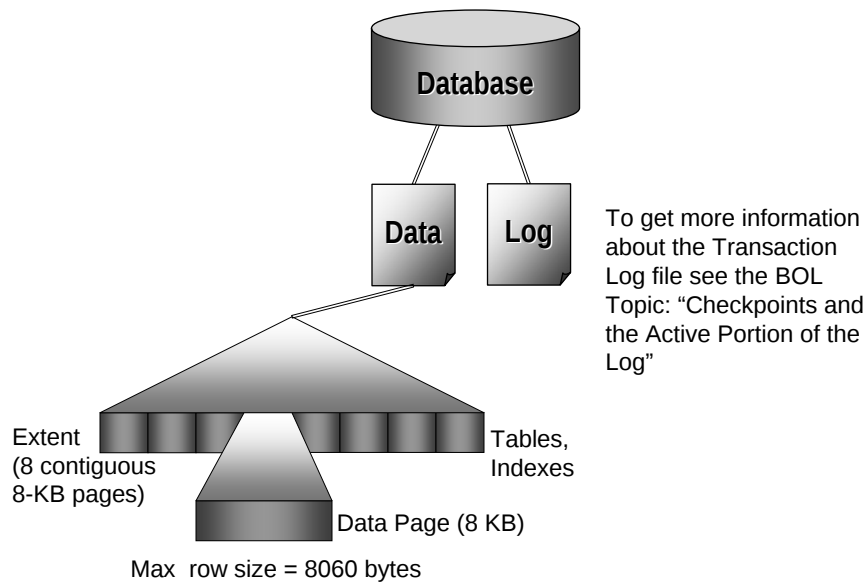
Database Properties

- Viewing Database Properties
 - Use DATABASEPROPERTYEX
 - sp_helpdb
 - Enterprise Manager
- Setting Database Properties
 - ALTER DATABASE
 - Do NOT use sp_dboption

Database Structure



How Data Is Stored



The Anatomy of a Data Modification

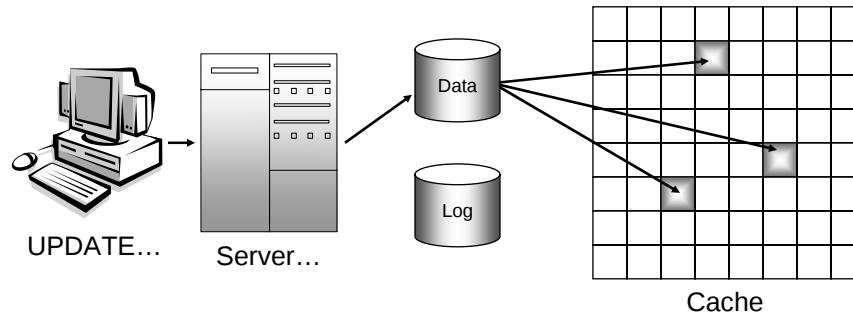
1. User sends UPDATE

- Update is highly selective (only 5 rows)
- Indexes exist to aid in finding these rows efficiently
- The update is a SINGLE statement batch NOT enclosed in BEGIN TRAN...COMMIT TRAN block therefore this is IMPLICIT transaction

2. Server receives the request and locates the data in cache OR reads the data from disk into cache

- Since this is highly selective only the necessary pages are read into cache (maybe a few extra but that's not important here)
- Let's use an example where the 5 rows being modified are located on 3 data pages

What it looks like - Data



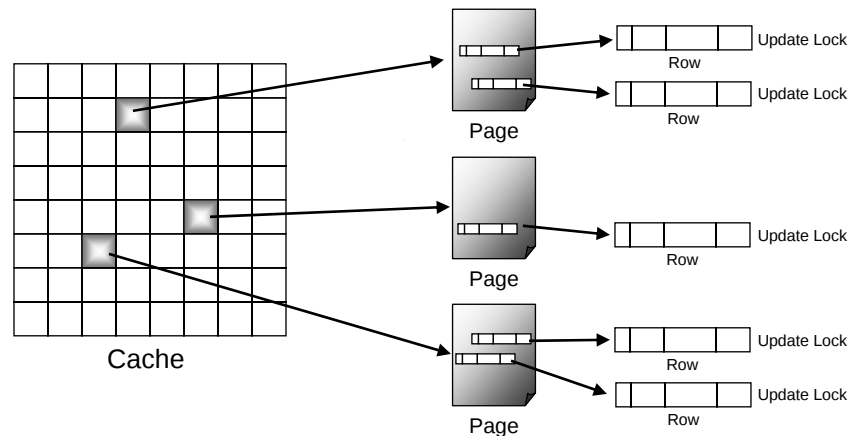
The Anatomy of a Data Modification

3. SQL Server proceeds to lock the necessary data

- Locks are necessary to give us a consistent point FOR ALL rows from which to start
- If any other transaction(s) have ANY of these rows locked we will wait until ALL locks have been acquired before we can proceed.
- In the case of this update (*because it's highly selective and because indexes exist to make this possible*) SQL Server will use row level locking.
- The rows are locked but there are also "intent" locks at higher levels to make sure other larger locks (like page or table level locks) are not attempted (and fail)
- There are a few locks that have already occurred – within indexes, etc. to read the data – but they are not significant here

This sounds complex but it's not too bad...

What it looks like - Locks

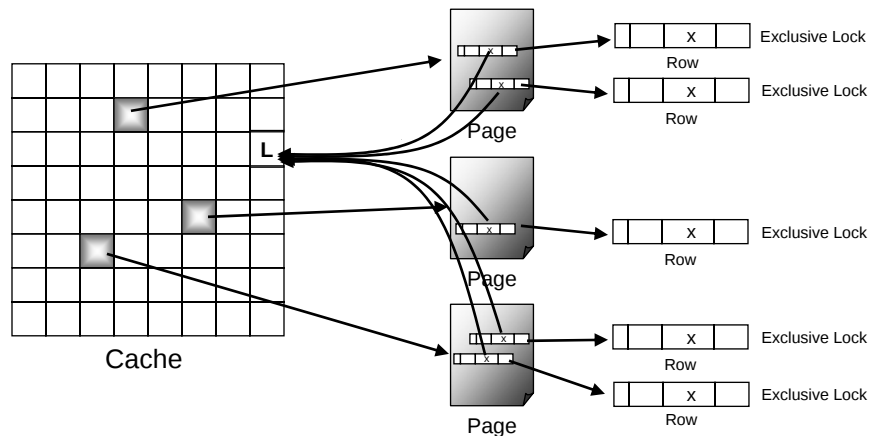


The Anatomy of a Data Modification

4. SQL Server can now begin to make the modifications – for EVERY row the process will include:

- 1. Change the lock to a stricter lock (eXclusive lock)**
 - An update lock helps to allow better concurrency by being compatible with shared locks (readers). Readers can read the pre-modified data as it is still transactionally consistent and has not YET been modified
 - The eXclusive lock is required in order to change the data because once modified no other reads will be allowed to see this transient/un-committed state of the data
- 2. Make the modification to the data row (yes, in cache)**
- 3. Log the modification to the transaction log pages (also in cache)**

What it looks like - Modifications

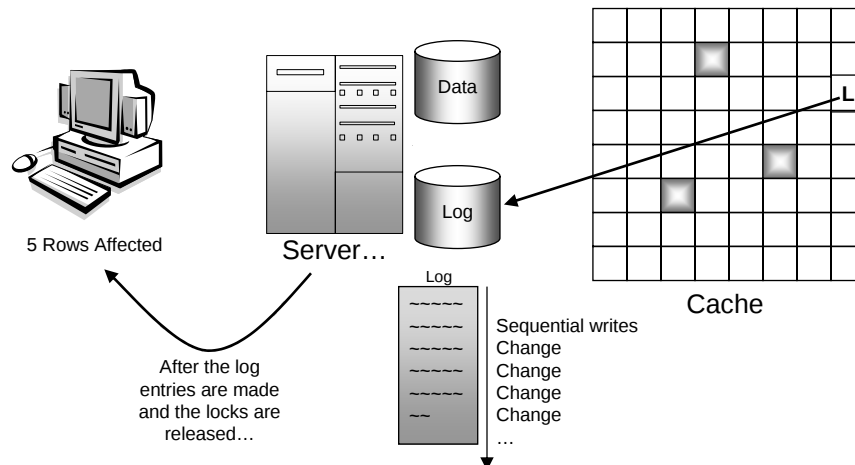


The Anatomy of a Data Modification

5. Finally, the transaction is complete – this is the **MOST** critical step

- All rows have been modified
- There are no other statements in this transaction – i.e. this is an implicit transaction in “Autocommit” mode. **NOTE: the session setting SET IMPLICIT_TRANSACTIONS ON cannot be turned on and rarely should be! Makes for sloppy developers!**
- Steps to ensure durability are:
 1. Write all log pages to transaction log ON DISK
 2. Release the locks
 3. Send a message to the user:
(5 Rows Affected)

What it looks like Write-Ahead Logging



So now what?

- The transaction log portion of the database ON DISK – is up to date
- The data in CACHE – is up to date
- But when does the data get written from cache to disk?

CHECKPOINT

It's important to realize that it's not the sole purpose of checkpoint to just to write committed pages... Instead a checkpoint writes ALL pages which have changed since they were brought into cache – regardless of the state of the transaction which changed them!

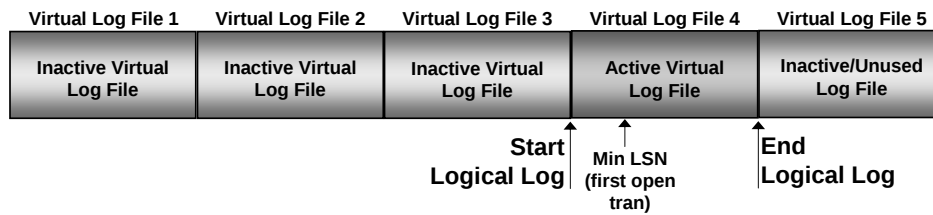
Data files

- Generally READ activity
- Lots of users = more random reads (some sequential)
- Separate from Log portion of database
- Optimize for read performance
- RAID 0+1, RAID 1+0 or RAID 5
- RAID 0+1 faster than 1+0 but less reliable
- RAID 5 is OK for data but not great on write intensive operations –
BACKUP/RESTORE/CHECKPOINT

Log files

- In OLTP DB Log file should be REDUNDANT
 - Most critical portion of the database if failure
 - NO data loss if log is accessible
 - = Up-to-the-minute recovery (NO_TRUNCATE)
- Generally write activity (transactions)
- Usually, sequential write activity
- Limited read activity (replication, rollback, triggers)
- Optimize for write activity (RAID 1, RAID 0+1 *w/hot spare*, RAID 1+0 *better redundancy, preferably NOT RAID 5*)
- Optimize for log activity
 - Capacity Planning (initial and reasonable size)
 - Minimize excessive autogrowths (clean up VLFs)

How the Transaction Log Works



- On commit, activity is written to the log – active and likely sequential
- Activity moves through log sequentially, fills and then goes to a second file or autogrows
- Excessive autogrowth causes:
 - Slight pause on autogrow
 - Windows call to extend a file (may cause file fragmentation)
 - Adds “VLFs” to the file on each autogrowth

Transaction Log Optimization

- Pre-allocate the log to appropriate size
- Minimize autogrowths
- Log might NOT show 0 Percent Log Used after backup for example - the % Log Used will be ~6%
- Disk Bound Systems *might* experience some performance degradation at log backup
- Consider RAID 1+0 instead of RAID 1, RAID 5 generally is not recommended
- Optimize existing file – minimize VLFs

Minimize VLFs

- Excessive VLFs add overhead to log related activities (transaction logging, log backups, logreader, triggers (*inserted/deleted*), etc.)
- Execute DBCC LOGINFO
 - Number of rows = Number of VLFs
- If excessive then:
 - Free up log space using BACKUP LOG...
 - Shrink the transaction log file
DBCC SHRINKFILE(logfilename, TRUNCATEONLY)
 - Alter the Database and modify log file size
ALTER DATABASE dbname
MODIFY FILE (NAME = file_name, SIZE = new_size)

Database Changes

- Databases can be moved:
 - sp_detach_db
 - sp_attach_db
- Databases can be enlarged with ALTER DATABASE
- Database properties should be set with ALTER DATABASE (not sp_dboption)
- See KB Q224071 – for details on changing the location of system databases – i.e. moving TempDB
- Optimizing TempDB if on multiproc KB Q328551

Review

- **Creating Databases**
- **Database Structures**
- **The Anatomy of a Data Modification**
 - **From Data Access**
 - **Locking**
 - **Write-ahead Logging**
- **Data and Log File Placement**
- **How the Transaction Log Works**
- **Optimizing Log Activity**