Module 3

Database structures

Module Overview

- Database structure internals
- Best practices for database files
- Best practices for tempdb

Lesson: Database Structure Internals

- Database file types
- Filegroups and files
- Pages and extents
- Shared vs uniform extents
- Page structure
- Page types
- Record types
- Allocation bitmaps and special pages
- Allocation units
- Investigating page structure

Database file types

- Data files
  - Divided into 8KB blocks called pages
  - Pages are grouped in 8, called extent
  - One primary file (mdf recommended)
  - X number of secondary files (ndf recommended)
  - More than one file can improve
    - Manageability
    - Performance (the disk subsystem might have limitations per file)
- Transaction log file
  - Written sequentially
  - No performance gain having more than one file
  - Initially buffered in memory, hardened at commit

Database Files and Filegroups
https://docs.microsoft.com/en-us/sql/relational-databases/databases/database-files-and-filegroups

SQL Server Transaction Log Architecture and Management Guide
https://docs.microsoft.com/en-us/sql/relational-databases/sql-server-transaction-log-architecture-and-management-guide

Database files types, continued

- Filestream
  - Option for varbinary(max) data type
  - More on this later
- Memory optimized
  - Use with memory-optimized tables
  - More on this in a later module

Filegroups and files

- Filegroups
  - Logical grouping of data files
  - Can bring administration and performance benefits
  - FILESTREAM and memory optimized tables has their own file groups
- Example:
  - Mandatory
  - Optional

Demo Create database

Pages and extents

- An extent is physically continuous pages
- Mixed/shared: shared between different database objects (*allocations* to be precise)
- Uniform: owned by a single database object (*allocation* to be precise)

0	1	2	3	4	5	6	7
8 Blue	9 Blue	10 Blue	11 Green	12 Green	13 Blue	14 Red	15 Green
16 Blue	17 Red	18 Blue	19 Blue	20 Green	21 Blue	22	23
24 Blue	25 Blue	26	27	28	29	30	31
32	33	…					

System

Shared

Shared

Uniform

Unallocated

Unallocated

Unallocated

Unallocated

Denna slide förklarar konceptet med sidor och extenter i SQL Server. En extent består av fysiskt sammanhängande sidor. Det finns två typer av extenter: **blandade** (shared), där olika databastabeller delar på samma extent, och **uniforma**, där en enda tabell äger hela extenten. Bilden visar en tabell där vissa sidor tillhör specifika databastabeller (markerade i blått, grönt och rött), medan andra är systemreserverade eller oallokerade. Systemallokerade sidor används för interna SQL Server-funktioner. De oallokerade sidorna är ännu inte tilldelade någon specifik användning. Syftet är att förklara hur lagringshantering fungerar på en låg nivå i SQL Server.

Pages and Extents Architecture Guide
https://docs.microsoft.com/en-us/sql/relational-databases/pages-and-extents-architecture-guide

Shared vs uniform extents

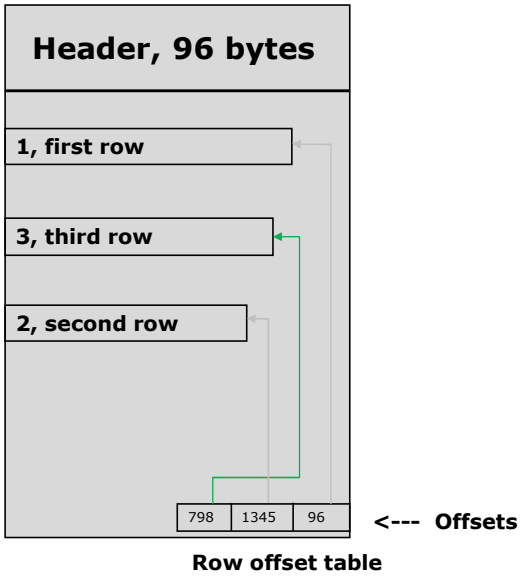
- Traditionally, the first 8 pages allocated from shared extents
- As of SQL Server 2016, Uniform extents are used from 1:st page
  - Controlled by database setting:
  - SET MIXED_PAGE_ALLOCATION ON/OFF
  - This could be requested using trace flag 1118 prior to 2016
  - Trace flag 1118 has no effect as of 2016
  - Shared extents are still used for other things
    - Such as IAM pages
- SQL Server keep track of the first 8 pages and each extents that an allocation is using by the IAM page
  - Can be investigated using sys.dm_db_database_page_allocations()

Denna slide förklarar skillnaden mellan **delade (shared) och enhetliga (uniform) extenter** i SQL Server. Traditionellt har de första åtta sidorna i en tabell allokerats från delade extenter, där flera objekt delar på samma extent. Från och med **SQL Server 2016** används istället **enhetliga extenter direkt från första sidan**, vilket innebär att en tabell får sin egen extent från början. Detta styrs av databasalternativet **MIXED_PAGE_ALLOCATION**, som kan vara på eller av. Före SQL Server 2016 kunde detta styras med **trace flag 1118**, men det flagget har ingen effekt i nyare versioner. Dock används fortfarande **delade extenter** för vissa systemrelaterade data, exempelvis **IAM-sidor (Index Allocation Map)**, som spårar tilldelning av sidor.
SQL Server håller koll på de första åtta sidorna och de extenter som en allokering använder genom **IAM-sidor**. För att analysera sidallokeringar kan man använda DMV:n **sys.dm_db_database_page_allocations()**, vilket ger insikt i hur sidor och extenter fördelas. Denna förändring i SQL Server 2016 förbättrar prestanda genom att minska fragmentering och förenkla hanteringen av sidallokeringar, särskilt i databaser med många små tabeller.

Page Structure

Pages:

- 8 KB in size
- 96-byte page header



Denna slide förklarar **sidstrukturen** i SQL Server. Varje sida är **8 KB** stor och har en **96-byte header**, som innehåller metadata om sidan. Data radas inte nödvändigtvis upp sekventiellt, utan en **radoffset-tabell** längst ner på sidan håller koll på var varje rad lagras.
Radoffset-tabellen gör att SQL Server snabbt kan hitta rader utan att behöva skanna hela sidan. Detta möjliggör effektiv datalagring och snabb åtkomst. Om rader flyttas inom sidan uppdateras endast offset-tabellen, vilket minskar fragmentering och förbättrar prestanda vid insättningar och uppdateringar.

Inside the Storage Engine: Anatomy of a page
https://www.sqlskills.com/blogs/paul/inside-the-storage-engine-anatomy-of-a-page/

Understanding the Internals of a Data Page
https://www.sqlservercentral.com/articles/understanding-the-internals-of-a-data-page

Page Types

Pages:

- 8 KB in size
- 96-byte page header

	Page Types	
1	Data	Heap and clustered leaf pages
2	Index	Index pages (not including clustered leaf pages)
3, 4	LOB	Large value type pages, Row overflow pages
13	Boot	Page 9 in file 1
15	File Header	Page 0
11	PFS	More on this later
8	GAM	More on this later
9	SGAM	More on this later
10	IAM	More on this later
16	DIFF_MAP	More on this later
17	ML_MAP	More on this later

Denna slide visar olika **sidtyper (Page Types)** i SQL Server. Varje sida är **8 KB stor** och har en **96-byte header**. Här är några viktiga sidtyper:
•**Data (1):** Innehåller heap-data och löv-noder i klustrade index.
•**Index (2):** Indexsidor för icke-klustrade index och mellanliggande nivåer i klustrade index.
•**LOB (3,4):** Lagrar stora objekt (t.ex. text, varbinary).
•**Boot (13):** Viktig systeminformation om databasen.
•**File Header (15):** Innehåller metadata om databasen.
•**PFS (11), GAM (8), SGAM (9), IAM (10):** Hanterar allokering och ledigt utrymme i databasen.
För att visa sidtyper i en databas kan du använda denna fråga:

DBCC IND('DatabasNamn', 'TabellNamn', -1) Den listar alla sidor i tabellen och visar deras **PageType**, där varje nummer motsvarar en typ enligt sliden.
--Eller

```
SELECT *
FROM
sys.dm_db_database_page_allocations(DB_ID('Adve
ntureworks2017'
), OBJECT_ID('person.address'), NULL, NULL,
'DETAILED');
```

Pages and Extents Architecture Guide
https://docs.microsoft.com/en-us/sql/relational-databases/pages-and-extents-architecture-guide

SQL SERVER – Identifying Page Types
https://blog.sqlauthority.com/2016/04/16/sql-server-identifying-page-types/

SQL Server page types list
https://www.dabrowski.space/posts/sql-server-page-types-list/

Record types

- Record types:
  - Data records
    - Leaf pages for clustered indexes
    - Heap pages for heap tables
  - Index records
    - Leaf level index records
      - One can argue whether a *clustered* leaf page should be considered a data page or an index page. Best answer to that is probably "both".
    - Non-leaf level index records
  - Forwarding/forwarded records
  - Text records
  - Versioned records
  - Ghost records

Denna slide förklarar **record types** i SQL Server, alltså de olika typer av poster som lagras i databassidor.

Data records

•**Leaf pages för klustrade index** – Innehåller själva data i tabeller med klustrade index.

•**Heap pages för heap-tabeller** – Data lagras utan specifik ordning eftersom det saknas klustrat index.

Index records

•**Leaf level index records** – Lagrar indexnycklar på den lägsta nivån i ett index.

•**Non-leaf level index records** – Mellanliggande indexnivåer som används för navigering.

- För klustrade index kan lövnivån räknas som både **data** och **index** eftersom den innehåller själva tabellraderna.

Forwarding/forwarded records

•Förekommer i **heap-tabeller** när en rad flyttas till en annan sida. SQL Server lämnar en **forwarding pointer** för att undvika att uppdatera alla referenser.

Text records

•Används för **LOB-data (Large Objects)** såsom TEXT, IMAGE eller VARBINARY(MAX), som lagras utanför vanliga sidor.

Versioned records

•Används vid **Snapshot Isolation** och **Row Versioning**, där gamla rader sparas temporärt för transaktionshantering.

Ghost records

•Markerade som raderade men finns kvar tills **Ghost Cleanup Task** tar bort dem.

För att inspektera dessa records kan du använda:

DBCC PAGE('DinDatabas', 1, <PageID>, 3);

SQL Server : Understanding the Data Record Structure
https://www.sqlservercentral.com/blogs/sql-serverunderstanding-the-data-record-structure

SQL SERVER – What is Forwarded Records and How to Fix Them?
https://blog.sqlauthority.com/2018/03/08/sql-server-forwarded-records-fix/

SQL Server Storage Engine: LOB Storage
https://aboutsqlserver.com/2013/11/05/sql-server-storage-engine-lob-storage/

Ghost cleanup process guide
https://docs.microsoft.com/en-us/sql/relational-databases/ghost-record-cleanup-process-guide

Allocation Bitmaps and Special Pages

- Page allocations are tracked by special pages:
- Global Allocation Map (GAM)
- Shared Global Allocation Map (SGAM)
- Differential Change Map (DCM)
- Bulk Change Map (BCM)
- Page Free Space (PFS)
  - Maps pages, one byte per page
  - Repeats approx. every 64 MB
- Index Allocation Map (IAM)
- All except File header and PFS:
  - One bit per extent
  - Repeats every 4GB

0 F hdr	1 PFS	2 GAM	3 SGAM	4	5	6 DCM	7 BCM
8 BlueIAM	9 Blue	10 Blue	11 GreenIAM	12 Green	13 Blue	14 RedIAM	15 Green
16 Blue	17 Red	18 Blue	19 Blue	20 Green	21 Blue	22	23
24 Blue	25 Blue	26	27	28	29	30	31
32	33	...					

Absolut, Robert! Dessa specialsidor i SQL Server hjälper till att hålla reda på hur sidor och utrymmen används i databasfiler. Låt oss dyka in:

- **Global Allocation Map (GAM)**: Denna sida håller koll på vilka "extents" (grupper av 8 sidor) som är lediga.

- **Shared Global Allocation Map (SGAM)**: Liknar GAM men fokuserar på extents som är delvis lediga och kan användas för att lagra mer data.

- **Differential Change Map (DCM)**: Spårar vilka sidor som har ändrats sedan senaste säkerhetskopian. Hjälper till att göra inkrementella säkerhetskopior snabbare.

- **Bulk Change Map (BCM)**: Håller reda på sidor som har ändrats genom bulkoperationer. Används också i säkerhetskopieringsscenario.

- **Page Free Space (PFS)**: Denna sida visar hur mycket ledigt utrymme varje sida har. Bra för att snabbt hitta en sida att lägga till data på.

- **Index Allocation Map (IAM)**: Används för att spåra vilka extents som används av en viss tabell eller ett index.

- **All except File header and PFS**: Dessa används oftast inte direkt men är viktiga för databasens interna hantering. De använder en bit per "extent" och repeteras varje 4 GB.

Alla dessa sidor repeteras ungefär varje 64 MB eller 4 GB i databasfilen, beroende på deras

användning och typ.

Förhoppningsvis ger det dig en bättre bild av hur SQL Server håller reda på allt!
Inside The Storage Engine: GAM, SGAM, PFS and other allocation maps
https://www.sqlskills.com/blogs/paul/inside-the-storage-engine-gam-sgam-pfs-and-other-allocation-maps/

SQL Server Extents, PFS, GAM, SGAM and IAM and related corruptions
https://techcommunity.microsoft.com/t5/sql-server-support-blog/sql-server-extents-pfs-gam-sgam-and-iam-and-related-corruptions/ba-p/1606011

Allocation Units

- Three types of allocation units
  - IN_ROW_DATA
  - LOB_DATA (optional)
  - ROW_OVERFLOW_DATA (optional)
- One of above per partition
- Also one per index

Demo LOB data

•**IN_ROW_DATA**: Detta är din standardallokering där de flesta av dina data ligger. Det inkluderar all data som passar inom den vanliga sidstorleken (typiskt 8 KB).
•**LOB_DATA (optional)**: Står för "Large Object Data". Om du har stora textfält eller binärdata som inte passar inom en standardsida, lagras de här.
•**ROW_OVERFLOW_DATA (optional)**: Om en rad i din tabell blir för stor för att passa på en enkel sida, kan den "överflöda" till denna typ av lagring.

SQL Server – Understanding Allocation Units – In Row Data, LOB Data & Row Overflow Data
https://dataginger.com/2013/10/14/sql-server-understanding-allocation-units-in-row-data-lob-data-row-overflow-data/

sys.allocation_units (Transact-SQL)
https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-allocation-units-transact-sql

Investigating page structure

- Decode the IAM page using:
  - sys.dm_db_database_page_allocations()
- Return the file/page/entry in offset table using
  - sys.fn_PhysLocFormatter(%%physloc%%)
    - Scalar function

    The three values combined in one column
  - sys.fn_PhysLocCracker(%%physloc%%)
    - Table function
    - Three columns
      - File #
      - Page #
      - Entry in offset table
    - Use CROSS APPLY to this
- Look at page internals using
  - DBCC PAGE
  - sys.dm_db_page_info

Demo Page structure

Självklart, Robert! Dessa funktioner är verktyg för att förstå var data faktiskt ligger i SQL Server på en fysisk nivå. Låt mig bryta ner dem:

1. **sys.fn_PhysLocFormatter(%%physloc%%)**: Detta är en skalär funktion, vilket betyder att den returnerar ett enkelt värde. Funktionen tar en fysisk lokaliseringsidentifikator och returnerar en sträng som kombinerar Fil-ID, Sidnummer och Position i offset-tabellen. Allt i en enda kolumn.

   Exempel:
   ```sql
   SELECT sys.fn_PhysLocFormatter(%%physloc%%) AS PhysicalLocation FROM YourTable;
   ```

2. **sys.fn_PhysLocCracker(%%physloc%%)**: Detta är en tabellfunktion, vilket betyder att den returnerar en tabell med rader och kolumner. Den bryter upp den fysiska lokaliseringsidentifikatorn i tre separata kolumner: File #, Page # och Entry in offset table.

   Exempel:
   ```sql
   SELECT * FROM YourTable CROSS APPLY sys.fn_PhysLocCracker(%%physloc%%) AS p;
   ```

   Här används `CROSS APPLY` för att koppla varje rad i "YourTable" till dess fysiska position som returneras av `sys.fn_PhysLocCracker()`.

Dessa funktioner är superanvändbara om du behöver dyka djupt in i databasens inre struktur och förstå var saker och ting faktiskt ligger. Hoppas det klargör saker för dig!

Use caution with sys.dm_db_database_page_allocations in SQL Server
https://www.mssqltips.com/sqlservertip/6309/use-caution-with-sysdmdbdatabasepageallocations-in-sql-server/

How to use the SQL Server sys.fn_PhysLocFormatter undocumented function
https://www.mssqltips.com/sqlservertip/1925/how-to-use-the-sql-server-sysfnphyslocformatter-undocumented-function/

SQL Servers Virtual Columns and Row Cracking
http://jongurgul.com/blog/sql-servers-virtual-columns-row-cracking/

Using DBCC PAGE to Examine SQL Server Table and Index Data
https://www.mssqltips.com/sqlservertip/1578/using-dbcc-page-to-examine-sql-server-table-and-index-data/

sys.dm_db_page_info (Transact-SQL)
https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-db-page-info-transact-sql

Lesson: Best practices for database files

- Physical I/O vs. logical I/O
- Allocations within datafiles
- Instant File Initialization
- Autogrow and autoshrink
- Datafile shrinking
- Monitoring database files

Physical I/O vs. Logical I/O

- Logical I/O – pages read from memory
- Physical I/O – pages read from disk
- Buffer cache hit ratio – percentage of pages read from cache without having to be read from disk
  - Watch out: read-ahead reads are not included in this calculation
  - High value tell us nothing
  - Low value is a guarantee that we have a “lots of” physical I/O

Demo logical and physical IO

I/O vs Logical I/O
https://www.sqlservercentral.com/blogs/io-vs-logical-io

The Curious Case of… does SQL Server use a read/write thread per LUN?
https://www.sqlskills.com/blogs/paul/the-curious-case-of-does-sql-server-use-a-readwrite-thread-per-lun/

Reading Pages
https://docs.microsoft.com/en-us/sql/relational-databases/reading-pages

Sequential Read Ahead
https://techcommunity.microsoft.com/t5/sql-server-blog/sequential-read-ahead/ba-p/383480

Allocations within data files and autogrow

- Allocation is simple in a file group that has a single file
- In a multifile group, SQL Server uses:
  - Round robin allocation
  - Proportional fill
- Autogrow can result in different file sizes
  - One file larger than the other
  - Trace flag 1117 prior to 2016 cause all files to grow
  - *Autogrow all files* is default for tempdb
    - Cannot be changed
  - Filegroup setting for other databases
    - Default is OFF

SQL Server Proportional Fill Algorithm Example
https://www.mssqltips.com/sqlservertip/4890/sql-server-proportional-fill-algorithm-example/

Investigating the proportional fill algorithm
https://www.sqlskills.com/blogs/paul/investigating-the-proportional-fill-algorithm/

Expand All Database Files Simultaneously Using SQL Server 2016 AUTOGROW_ALL_FILES
https://www.mssqltips.com/sqlservertip/4937/expand-all-database-files-simultaneously-using-sql-server-2016-autogrowallfiles/

ALTER DATABASE (Transact-SQL) File and Filegroup Options
https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-database-transact-sql-file-and-filegroup-options

Instant File Initialization

- Improves performance by skipping zeroing of data pages on file creation and growth
- Does not apply for log files
  - Until 2022, but max autogrow in order to get IFI for transaction log is 64MB
- Disabled by default
  - Potential security consideration: you can read data from deleted files using DBCC PAGE
- Enable if above is OK
- Check
  - See boot information in errorlog file
  - sys.dm_server_services.instant_file_initialization_enabled olumn
- Checkbox for this in SQL Server setup
- Change using “Perform volume maintenance tasks” security policy
  - Assign to SQL Server service account

Demo Instant File Initialization

Database Instant File Initialization
https://docs.microsoft.com/en-us/sql/relational-databases/databases/database-instant-file-initialization

Instant File Initialization
https://www.brentozar.com/blitz/instant-file-initialization/

In SQL Server 2022, set your transaction log file autogrowth rate to 64 MB
https://www.sqltact.com/2022/11/in-sql-server-2022-set-your-transaction.html

Autogrow

- File property
- Autogrow
  - Automatically expands a file when space is low
  - Manually manage file size for better performance
  - Auto grow best practices:
    - Leave enabled to avoid downtime in case file size limit is reached
    - Set to a fixed rather than a percentage

Considerations for the autogrow and autoshrink settings in SQL Server
https://docs.microsoft.com/en-us/troubleshoot/sql/admin/considerations-autogrow-autoshrink

Adjust autogrow values for database files
https://karaszi.com/adjust-autogrow-values-for-database-files

Datafile shrinking

- Shrink
  - Causes index fragmentation
  - Resource intensive
  - Degrades performance
- Best used for
  - Emptying a file before removing it
  - Changing a file or file group to read only
  - Reclaiming free space after deleting a ***large amount*** of data
- Alternative
  - Move all indexes to a new file group
  - Move heaps by creating clustered index and then possibly dropping it
  - Remove old file group
- Things that can cause shrink to take a very long time
  - LOB data (a table scan for each LOB page moved)
  - Heaps (update all non-clustered indexes for every row on every data-page that was moved)
  - Blocking (shrink waits forever)

Demo Do not shrink

Why you want to be restrictive with shrink of database files
https://karaszi.com/why-you-want-to-be-restrictive-with-shrink-of-database-files

Why you should not shrink your data files
https://www.sqlskills.com/blogs/paul/why-you-should-not-shrink-your-data-files/

Autoshrink

- Database setting
- Just say No
  - Verify that no database has this
  - sys.databases
    - is_auto_shrink_on

Considerations for the autogrow and autoshrink settings in SQL Server
https://docs.microsoft.com/en-us/troubleshoot/sql/admin/considerations-autogrow-autoshrink

Auto-Shrink Enabled or Job has Shrink Steps
https://www.brentozar.com/blitz/auto-shrink-enabled/

SHRINK a data file? Just say NO!!!
https://www.theboreddba.com/Categories/indexes/SHRINK-a-data-file-Just-say-NO.aspx

Auto-shrink – turn it OFF!
https://www.sqlskills.com/blogs/paul/auto-shrink-turn-it-off/

Monitoring database files

- Database file configuration
  - SSMS database properties
  - sys.database_files and sys.filegroups
  - sys.master_files
  - sys.dm_db_file_space_usage
- Database file activity
  - SSMS Activity Monitor – Data File I/O
  - Wait statistics – PAGEIOLATCH_*, WRITELOG
  - sys.dm_io_virtual_file_stats

sys.database_files (Transact-SQL)
https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-database-files-transact-sql

sys.filegroups (Transact-SQL)
https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-filegroups-transact-sql

sys.master_files (Transact-SQL)
https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-master-files-transact-sql

sys.dm_io_virtual_file_stats (Transact-SQL)
https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-io-virtual-file-stats-transact-sql

What Virtual Filestats Do, and Do Not, Tell You About I/O Latency
https://sqlperformance.com/2013/10/t-sql-queries/io-latency

Lesson: Best practices for tempdb

- Tempdb usage
- Tempdb configuration

Tempdb Usage

- Stores:
  - Internal objects
  - Version store, which includes
    - Trigger's *deleted* and *inserted* tables
    - Working space for online index operations
  - Some user objects, for instance
    - Temp tables
    - Table variables
- Recreated/reinitialized at startup
- You don't want tempdb to run out of space

tempdb database
https://docs.microsoft.com/en-us/sql/relational-databases/databases/tempdb-database

Tempdb Configuration

- Tempdb configuration
  - Enable instant file initialization
  - Size for expected workload
  - Consider to isolate tempdb data files for management purposes
  - Use multiple data files
    - 1 per CPU core up to 8 cores as a starting point
  - Enable auto growth as a contingency
  - Make files grow equally
    - Taken care of for you since 2016
  - Consider memory optimized tempdb metadata in 2019
    - Server-level setting
    - Makes 12 system tables use non-durable memory-optimized table
- Further improvements in 2022
  - Concurrent updates to GAM and SGAM pages

Demo Tempdb

Den här delen av sliden handlar om förbättringar i **tempdb** i SQL Server 2019 och 2022.
Memory-Optimized tempdb Metadata (SQL Server 2019)
•**Server-level setting**: Detta är en inställning på servernivå som kan aktiveras för att förbättra prestandan i **tempdb**.
•**12 systemtabeller blir minnesoptimerade och icke-durabla**:

- Detta innebär att vissa systemtabeller i tempdb (t.ex. de som hanterar tillfälliga tabeller och tabellvariabler) lagras i **memory-optimized tables**, vilket minskar I/O-belastningen och låsning (locks).
- Eftersom tempdb är **tillfällig** (återskapas vid omstart) finns inget behov av att skriva dessa tabeller till disk, vilket förbättrar prestandan.

Förbättringar i SQL Server 2022
•**Parallella uppdateringar av GAM och SGAM-sidor**:

- **GAM (Global Allocation Map)** och **SGAM (Shared Global Allocation Map)** spårar vilka extenter som är allokerade i tempdb.
- Tidigare kunde dessa sidor bli flaskhalsar under hög belastning, eftersom flera processer försökte uppdatera dem samtidigt.
- I SQL Server 2022 introducerades **konkurrerande (concurrent) uppdateringar**, vilket minskar blockeringar och förbättrar skalbarheten vid många samtidiga skrivningar i tempdb.

Dessa optimeringar är viktiga för databaser med hög aktivitet i **tempdb**, som ofta används för tillfälliga tabeller, arbetsytor och sessionstillstånd i SQL Server. 🚀

Cheat Sheet: How to Configure TempDB for Microsoft SQL Server
https://www.brentozar.com/archive/2016/01/cheat-sheet-how-to-configure-tempdb-for-

microsoft-sql-server/

Improve scalability with system page latch concurrency enhancements in SQL Server 2022 (includes 'memory-optimized tempdb metadata' config discussion)
https://cloudblogs.microsoft.com/sqlserver/2022/07/21/improve-scalability-with-system-page-latch-concurrency-enhancements-in-sql-server-2022/

V1 Lab 3: Database structures

- Exercise 1: Exploring page allocation structure
  - Do if you are curious, else skip this
  - SQL commands in Lab Answer Keys
- Exercise 2: Configuring Instant File Initialization
  - SQL Server service account is member of Domain Admins
  - This is NOT a best practice!
- Exercise 3: Reconfiguring tempdb datafiles

Estimated Time: 30 minutes

Lab 3: Database structures

- Ex 1: Verifying Instant File Initialization
- Ex 2: Verifying number of tempdb datafiles

Estimated Time: 30 minutes