

The logo for the 2010 PASS Summit is a stylized shield shape with a jagged, flame-like border. The text "2010" is in yellow, "PASS" is in red, and "SUMMIT" is in white. The background of the shield is dark blue with orange and yellow lines radiating from the center.

2010  
**PASS**  
SUMMIT

November 8 - 11, Seattle, WA  
[www.sqlpass.org/na2010](http://www.sqlpass.org/na2010)



# How to Optimize TEMPDB Performance

*Brad M McGehee*  
*Director of DBA Education*  
*Red Gate Software*

# My Assumptions About You

You are most likely a DBA or developer with one or more years of SQL Server experience.

You have a basic understanding of how to use Performance Monitor.

You have a basic understanding of DMVs.

You have a basic understanding of SQL Server internals.



# What We Are Going to Learn Today

What Objects are Stored in TEMPDB

TEMPDB Internals

Types of TEMPDB Problems

Ways to Monitor TEMPDB

Ways to Optimize TEMPDB

Note: This session covers SQL Server 2005/2008, and focuses on the most common ways to work with TEMPDB, not including every potential option.



# What Objects are Stored in TEMPDB?

TEMPDB is used to store three different categories of temporary data:

- User Objects
- Internal Objects
- Version Stores



# User Objects

Local and global temporary tables (and indexes if created)

User-defined tables and indexes

Table variables

Tables returned in table-valued functions

Note: This list, and the following lists, are not designed to be all inclusive.



# Internal Objects

Work tables for DBCC CHECKDB and DBCC CHECKTABLE.

Work tables for hash operations, such as joins and aggregations.

Work tables for processing static or keyset cursors.

Work tables for processing Service Broker objects.

Work files needed for many GROUP BY, ORDER BY, UNION, SORT, and SELECT DISTINCT operations.

Work files for sorts that result from creating or rebuilding indexes (SORT\_IN\_TEMPDB).



# Version Stores

The version store is a collection of pages used to store row-level versioning of data.

There are two types of version stores:

- 1. Common Version Store:** Examples include:
  - Triggers.
  - *Snapshot isolation* and *read-committed snapshot isolation*.
  - MARS (when multiple active result sets are used).
- 2. Online-Index-Build Version Store:**
  - Used for online index builds or rebuilds. EE edition only.



# FYI: Quick Definitions

**Snapshot Isolation:** Specifies that data read by any statement in a transaction will be transactionally consistent with the data that existed at the start of the transaction. Data modifications made by other transactions after the start of the transaction are not visible to statements executing in the current transaction. The effect is as if the statements in a transaction gets a snapshot of the committed data as it existed at the start of the transaction.

**Read-Committed Snapshot Isolation:** Row versioning is used to present each statement with a transactionally consistent snapshot of the data as it existed at the start of the statement. Locks are not used to protect the data from updates by other transactions.

**Multiple Active Results Sets (MARS):** Allows applications to have more than one pending request per connection, and in particular, to have more than one active default result set per connection.



# Keep the Following in Mind

If your SQL Server instance doesn't employ many of the activities just described, then *TEMPDB performance may not be an issue for you.*

On the other hand, if your SQL Server instance uses many of the above features, then *TEMPDB could become a significant bottleneck* for your SQL Server instance.

Only by *investigation* will you know.

Keep in mind that there is *only one TEMPDB*, and it is possible for one misbehaved application and database to indirectly affect the performance of all the other databases on the same instance.



# TEMPDB Internals (1)

TEMPDB is dropped and recreated every time the SQL Server service is stopped and restarted.

Each SQL Server instance may have only one TEMPDB, although TEMPDB may have multiple physical files within a single filegroup.

When SQL Server is restarted, TEMPDB inherits many of the characteristics of model, and creates an MDF file of 8MB and an LDF file of 1MB (default setting).

By default, autogrowth is set to grow by 10% with an unlimited MDF file size, and a 2TB maximum LDF file size.



# TEMPDB Internals (2)

Many TEMPDB database options can't be changed (e.g. Database Read-Only, Auto Close, Auto Shrink).

TEMPDB only uses the simple recovery model.

TEMPDB may not be backed up, restored, be mirrored, have database snapshots made of it, or have many DBCC commands run against it.

TEMPDB may not be dropped, detached, or attached.



# TEMPDB Internals (3)

TEMPDB logging works differently from regular logging. Operations are minimally logged, as redo information is not included, which reduces TEMPDB transaction log activity.

The log is truncated constantly during the automatic checkpoint process, and should not grow significantly, although it can grow with long-running transactions, or if disk I/O is bottlenecked.

If a TEMPDB log file grows wildly:

- Check for long-running transactions (and kill them if necessary).
- Check for I/O bottlenecks (and fix them if possible).
- Manually running a checkpoint can often temporarily reduce a wildly growing log file if bottle-necked disk I/O is the problem.



# Execution Plans and TEMPDB

When a query execution plan is cached, temporary objects in TEMPDB that are required by the plan, if any, are often cached as well. This is called *temporary object reuse*.

Obviously, not the entire object is cached, just a portion of it. When a temporary object is cached, *up to nine pages are cached* for reuse.

This improves the performance of the next execution of the query as the object already partially exists, but this also *takes up space* in TEMPDB.

*If the system is low on memory*, the Database Engine removes the execution plan and drops the cached temporary objects.



# Types of TEMPDB Problems

Generally, there are three major problems you run into with TEMPDB:

1. TEMPDB is experiencing an *I/O bottleneck*, hurting performance. Most common type of TEMPDB problem.
2. TEMPDB is experiencing *contention on various global allocation structures* (metadata pages) as temporary objects are being created, populated, and dropped. E.G. *Any space-changing operation acquires a latch on PFS, GAM or SGAM pages to update space allocation metadata*. A large number of such operations can cause excessive waits while latches are acquired, creating a bottleneck (hotspot), and hurting performance. More common in busy OLTP applications.
3. TEMPDB has run *out of space*. Can happen to anyone.

Ideally, you should be monitoring all these on a proactive basis to identify potential problems.



# FYI: Quick Definitions

**PFS (Page Free Space):** Tracks the allocation status of each page, such as whether an individual page has been allocated, and the amount of free space on each page. PFS uses one byte to track each page.

**GAM (Global Allocation Map):** Tracks what extents have been allocated. Each GAM page covers 64,000 extents, or almost 4 GB of data.

**SGAM (Shared Global Allocation Map):** Tracks which extents are currently being used as mixed extents, and also have at least one unused page. Each SGAM covers 64,000 extents, or almost 4 GB of data.



# Identifying TEMPDB I/O Problems

Use **Performance Monitor** to help determine how busy the disk is where your TEMPDB MDF and LDF files are located. Some counters to consider include:

**LogicalDisk Object: Avg. Disk Sec/Read:** The average time, in seconds, of a read of data from disk (read latency).

**LogicalDisk Object: Avg. Disk Sec/Write:** The average time, in seconds, of a write of data to the disk (write latency).

Numbers below are a *general guide* only and may not apply to your hardware configuration.

- Less than 10 milliseconds (ms) = very good
- Between 10-20 ms = okay
- Between 20-50 ms = slow, needs attention
- Greater than 50 ms = serious IO bottleneck



# Identifying Contention on Allocation Structures Using Performance Counters

Consider using some of these performance counters to monitor allocation/deallocation contention in SQL Server.

- **Access Methods:Worktables Created/sec:** The number of *work tables* created per second. Work tables are temporary objects and are used to store results for query spool, LOB variables, and cursors. This number should generally be less than 200, but can vary based on your hardware.
- **Access Methods:Workfiles Created/sec:** Number of *work files* created per second. Work files are similar to work tables but are created by *hashing operations*. Used to store temporary results for hash and hash aggregates. High values may indicate contention potential. *Create a baseline.*
- **Temp Tables Creation Rate:** The number of temporary tables created/sec. High values may indicate contention potential. *Create a baseline.*
- **Temp Tables For Destruction:** The number of temporary tables or variables waiting to be destroyed by the cleanup system thread. Should be near zero, although spikes are common.



# Identifying Contention on Allocation Structures Using Wait States

If latches are waiting to be acquired on TEMPDB pages for various connections, this may indicate allocation page contention.

Use this code to find out:

```
SELECT session_id, wait_duration_ms, resource_description
FROM sys.dm_os_waiting_tasks
WHERE wait_type like 'PAGE%LATCH_%' AND resource_description like '2:%'
```

	session_id	wait_duration_ms	resource_description
1	54	109	2:1:2

## Allocation Page Contention:

2:1:1 = PFS Page

2:1:2 = GAM Page

2:1:3 = SGAM Page



# Identifying Contention on Allocation Structures Using DMVs

This DMV can tell you how much space (in pages) is allocated to the various contents of the TEMPDB, so you know what objects are most used in your TEMPDB.

```
SELECT  
  
SUM (user_object_reserved_page_count)*8 as usr_obj_kb,  
  
SUM (internal_object_reserved_page_count)*8 as  
internal_obj_kb,  
  
SUM (version_store_reserved_page_count)*8 as  
version_store_kb  
  
FROM sys.dm_db_file_space_usage
```



# TEMPDB Space Allocation

A higher % allocation for **user objects** implies that objects that are created by applications are the major consumers of TEMPDB. This may or may not be a cause of concern, but can imply potential allocation page contention.

A higher % allocation for **internal objects** implies that the query plans make heavy use of TEMPDB. This may not be a problem, but you may want to look at the queries and try to optimize them.

A higher % allocation for the **version store** implies that the version store cleanup cannot keep pace with version generation. See if a long-running transaction is preventing version store cleanup. Or, a high transaction throughput might be generating a large number of versions per minute.



# Monitoring TEMPDB Space

Performance counters to consider watching:

- SQL Server: Database: Data File(s) Size(KB): TEMPDB
- SQL Server: Database: Log File(s) Used Size(KB): TEMPDB
- SQL Server: Transactions: Free Space in TEMPDB (KB)

DMV

- `sys.dm_db_file_space_usage`

*Consider creating an alert so you are notified if TEMPDB grows larger than its typical size.*



# If TEMPDB Space Runs Low, Errors Occur

Check the SQL Server error log for these errors:

- 1101 or 1105: A session has to allocate more space in TEMPDB in order to continue
- 3959: The version store is full.
- 3967: The version store has been forced to shrink because TEMPDB is full.
- 3958 or 3966: A transaction is unable to find a required version record in TEMPDB.



# Tips for Optimizing TEMPDB

Minimize the use of TEMPDB

Add more RAM to your server

Locate TEMPDB on its own array

Locate TEMPDB on a fast I/O subsystem

Pre-allocate TEMPDB space – *everyone needs to do this*

Don't shrink TEMPDB if you don't need to

Divide TEMPDB among multiple physical files

Avoid using Transparent Data Encryption (2008)



# How to Optimize TEMPDB—General

Generally, if you are *building a new SQL Server instance*, it is a good idea to *assume that TEMPDB performance will become a problem*, and to *take proactive steps* to deal with this possibility.

It is easier to deal with TEMPDB performance issues *before they occur*, than after they occur.

The following TEMPDB performance tips *may or may not apply* to your particular situation.

It is important to *evaluate each recommendation*, and determine which ones best fit your particular SQL Server's instance. Not a one size fits all approach.



# Minimize the Use of TEMPDB (1)

Use *standard performance tuning techniques to boost your server's performance before you do anything*. Examples include:

- Don't return more rows than you need.
- Don't sort data that doesn't need sorting, or sort it on the client.
- Don't use UNION or SELECT DISTINCT if not needed.
- Keep transactions short.
- Use proper indexing.
- Avoid using local and global temp tables. Either rewrite your code to avoid using them, or consider creating a permanent work table.



# Minimize the Use of TEMPDB (2)

Avoid using *static and keyset-driven cursors*. In most cases, cursors can be avoided by rewriting the code.

Avoid using *large object data types*.

Avoid using *table variables*.

Avoid *aggregating excessive amounts of data*.

Avoid *joins that indicate a hash-type join* in the query execution plan. Rewrite the query or use better indexing.

Avoid using *triggers*.

Avoid using *row-versioning-based transaction isolation levels*.



# Minimize the Use of TEMPDB (3)

Avoid using the *SORT\_IN\_TEMPDB* option when creating or rebuilding an index. If you decide to use this option, schedule it to run during a slower time of the day.

Avoid using *online index rebuilding* (Enterprise Edition), which uses row versioning and, in turn, uses TEMPDB. If you decide to use this option, schedule it to run during a slower time of the day.

Schedule jobs, such as DBCC CHECKDB, that use TEMPDB heavily, at times of the day *when the SQL Server instance is less busy*. Or perform DBCC CHECKDB on a database backup on another server.



# I Need the Features You Told Me to Avoid

The previous slides suggest avoiding taking advantage of a lot of tools that are available to you.

If you need these tools, then use them, but keep in mind that they affect TEMPDB performance, and plan accordingly.

On the other hand, don't be lazy.



# Enhance Temporary Object Reuse

Temporary objects need to be created inside another object in order to be cached, such as:

- Stored procedure
- Trigger
- User-defined function
- Return table of a user-defined table-valued function

Because of the above, temporary object reuse can't be used if the objects are created using dynamic T-SQL.

If a temp table is created, and if any DDL (creating index for example) is run against the table, it can't be cached.

Using named constraints when creating temp tables also prevents temporary object reuse.



# Add More RAM to Your Server

Depending on the operation, SQL Server tries to perform the action in the *buffer cache*. (e.g. sorts)

If the buffer cache does not have enough available space, then the operation may have to *spill to TEMPDB*. This places additional overhead on TEMPDB.

If your server is *experiencing a memory bottleneck*, then adding RAM can help reduce the load on TEMPDB.

On the other hand, *if your server has plenty of memory, adding more won't help TEMPDB performance*.



# Locate TEMPDB on its Own Array

If TEMPDB is very active, disk I/O contention can become an issue.

One way to help mitigate this problem is to locate TEMPDB on its own disk array (or isolated LUN) so that TEMPDB activity doesn't have to compete with other disk I/O activity.

If you can't put TEMPDB on its own array, then at least separate the mdf and ldf files, as you should be doing with your production databases.



# Locate TEMPDB on Fast I/O Subsystem

Always locate TEMPDB on the *fastest I/O subsystem* you have available.

*Prefer RAID 10 or RAID 1.* RAID 5 is slow for writes and should generally be avoided for TEMPDB, as TEMPDB is often write-intensive.

If using a SAN, consult with a SAN engineer to *ensure that TEMPDB won't be affected by other disk I/O.*

*Consider SSD drives* for TEMPDB MDF and LDF files.



# Leave Auto Statistics On

By default, the database options Auto Create Statistics and Auto Update Statistics are turned on for TEMPDB.

In most cases, don't turn these options off, as SQL Server *will automatically create and update statistics as needed in temporary tables and indexes*, helping to boost performance of the many operations performed on them.



# Pre-Allocate TEMPDB Space

The default 8MB of TEMPDB space that is automatically allocated is generally *way too small a value*.

As SQL Server needs more TEMPDB space, it will request it, and through autogrowth, it will get it, 10% at a time.

When autogrowth kicks in, *it uses up resources at often inconvenient times*, and transactions have to wait until autogrowth finishes, maybe causing timeouts. Also, autogrowth can contribute to *physical file fragmentation*, which hurts I/O.

Instead, you should *pre-allocate TEMPDB space* so that when SQL Server is restarted, that it is immediately sized to the optimum size.

Using *Instant File Initialization* can greatly speed the process of growing the TEMPDB at system start (but not for log growth).



# How to Pre-allocate TEMPDB Space

Use ALTER DATABASE, or SSMS, to increase the size of the TEMPDB database MDF and LDF files.

```
USE master;  
  
GO  
  
ALTER DATABASE TEMPDB  
MODIFY FILE  
    (NAME = tempdev,  
     SIZE = 20MB);  
  
GO  
  
ALTER DATABASE TEMPDB  
MODIFY FILE  
    (NAME = templog,  
     SIZE = 10MB)  
  
GO
```



# How to Move TEMPDB

Determine the current location of the MDF and LDF files

Run the ALTER DATABASE command to move the files

```
USE master;  
  
GO  
  
ALTER DATABASE TEMPDB  
  
MODIFY FILE (NAME = tempdev, FILENAME = 'E:\SQLData\TEMPDB.mdf');  
  
GO  
  
ALTER DATABASE TEMPDB  
  
MODIFY FILE (NAME = templog, FILENAME = 'F:\SQLLog\templog.ldf');  
  
GO
```

Stop and restart the SQL Server instance

Delete the old MDF and LDF files



# How Large Should TEMPDB Be?

If you have a *dedicated disk array* for TEMPDB, then *allocate about 80%* of the total available space to TEMPDB.

If you have to share TEMPDB with other database files, start with an *educated guess for new servers*. For *existing servers*, observe how much TEMPDB is used over time, and set its size accordingly, with a little extra room for growth.

*Keep autogrowth on* just in case you are wrong, and some unexpected transactions eats up a lot of additional TEMPDB space.

Assuming that you have pre-allocated TEMPDB, then using a *fixed autogrowth value from 1-10%* of the size of TEMPDB should cover most unexpected growth spurts with minimal resource effect.



# Don't Shrink TEMPDB

The amount of *space actually used out of the amount of space allocated to TEMPDB can vary wildly* over time. This is normal behavior.

*Don't be tempted to shrink TEMPDB* just because it is not using all of its space. If you do, you could start a cycle of growth and shrinkage that wastes server resources.

If cases where *TEMPDB grows excessively* (due to a wild query, for example), and if you don't expect the behavior to reoccur, then you may want to consider shrinking TEMPDB to its “normal” size.

The *best way to shrink TEMPDB is to stop and restart the server* so that the pre-allocated size is recreated.



# Divide TEMPDB Into Multiple Files

By default, when TEMPDB is created, the *MDF and LDF are created as single physical files.*

While the *LDF should always remain as a single physical file*, often *dividing the TEMPDB MDF into multiple physical files* can help performance.

*Multiple files can reduce contention on various global allocation structures by spreading activity over multiple physical files (these don't have to be on different arrays).* This is particularly useful for those TEMPDB database which spend a large percentage of time allocating and de-allocating temporary objects.

Keep in mind that using more physical disk files for TEMPDB can *increase switching costs and file management overhead*, which could potentially hurt performance.

Thus, a balance must be maintained. How is this done?



# How Many Physical Files Should TEMPDB Have?

Microsoft BOL and the SQLCAT team recommend that TEMPDB should have one physical file per CPU core. In many cases (not high-end OLTP systems) this might be too much.

Instead, as a good starting point, *create as many physical files that equal  $\frac{1}{4}$  to  $\frac{1}{2}$  of the CPU cores you have*, up to a maximum of 8 physical files. E.G. 8 CPU cores = 2 or 4 physical files.

When you exceed 8 physical files, *you may be reaching the point where the overhead* of maintaining multiple TEMPDB physical files overcomes the benefits.

Only through *formal testing* will you be able to determine the optimum number of physical files for your particular instance's TEMPDB.



# Recommendations for Multiple TEMPDB Files

If you choose to use multiple physical TEMPDB files, keep the following in mind:

- Each physical file must be identical in size.
- The autogrowth setting for each physical file must be identical, and should autogrowth kick in, manually ensure that all physical files have an identical size. Use an alert to let you know when this happens.

The above is important because SQL Server uses a proportional fill strategy to fill the physical files, and if the files become different sizes, then the benefit of multiple physical files goes away.



# Example for Creating Multiple Files

Determine where current TEMPDB file is at, and its size.

Determine what file(s) have to be added.

Run ALTER DATABASE command, or use SSMS.

```
USE master
GO
ALTER DATABASE [TEMPDB]
ADD FILE ( NAME = 'tempdev1',
FILENAME = 'f:\TEMPDB1.ndf' ,
SIZE = 8192KB , FILEGROWTH = 10%)
```

Note: Restart SQL Server to get physical files in synch



# Be Careful Using TDE

*SQL Server 2008 offers a database-level encryption feature called Transparent Database Encryption (TDE).*

If this is turned on for one or more databases on a SQL Server instance, then *all the activity in TEMPDB* (whether it comes from a encrypted or non-encrypted database) will be encrypted.

Encryption increases CPU usage and *slows down* TEMPDB performance.

If you decide to use TDE, you will want to incorporate as many of the *TEMPDB performance tuning tips* that I have suggested in order to help overcome the additional burden added by TDE.



# Take Aways From This Session

*You don't know if you have a TEMPDB performance issue unless you check. Check often on critical servers.*

*It is easier to prevent TEMPDB performance problems before they occur, than after they occur. This includes both application design and server configuration.*

Implement as many as these recommendations as you can on your SQL Server instances to *be proactive and to help prevent* potential TEMPDB performance issues.

When you get back home, *starting applying what you have learned* as soon as possible.



# Find Out More

## Free E-Books:

- [www.sqlservercentral.com/Books](http://www.sqlservercentral.com/Books)

## Check these out:

- [www.SQLServerCentral.com](http://www.SQLServerCentral.com)
- [www.Simple-Talk.com](http://www.Simple-Talk.com)

## Contact me at:

[bradmcgehee@hotmail.com](mailto:bradmcgehee@hotmail.com)

## Blogs:

[www.bradmcgehee.com](http://www.bradmcgehee.com)

[www.twitter.com/bradmcgehee](http://www.twitter.com/bradmcgehee)



# Complete the Evaluation Form to Win!

Win a Dell Mini Netbook – every day – just for handing in your completed form. Each session evaluation form represents a chance to win.

## Pick up your evaluation form:

- In each presentation room
- At the PASS Booth near registration

## Drop off your completed form:

- Near the exit of each presentation room
- At the PASS Booth near registration

*Sponsored by Dell*



The logo for the 2010 PASS Summit is a stylized, dark blue shield with a jagged, flame-like border. Inside the shield, the year "2010" is written in yellow, and "PASS SUMMIT" is written in red. The background of the shield features a colorful, abstract pattern of lines and shapes in shades of blue, green, and orange.

2010  
**PASS**  
**SUMMIT**

November 8 - 11, Seattle, WA  
[www.sqlpass.org/na2010](http://www.sqlpass.org/na2010)



# Thank you

for attending this session and the  
2010 PASS Summit in Seattle