**Detect and Prevent Unexpected Growth of TempDB - SQL Server**

<https://www.spotlightcloud.io/blog/how-to-detect-and-prevent-unexpected-growth-of-sql-server-database-tempdb>

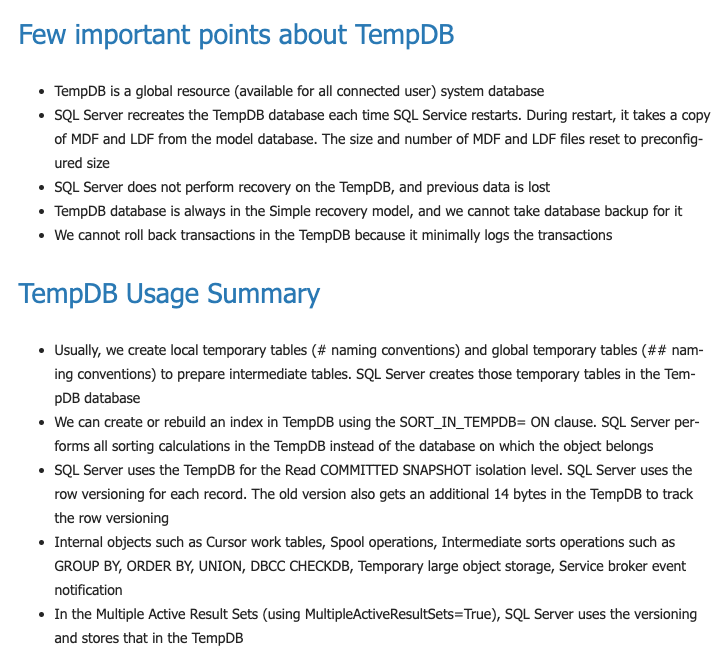
**By:**[**Rajendra Gupta**](https://www.spotlightcloud.io/blog/author/rajendra-gupta)

Each SQL Server instance contains the system SQL Server database called TempDB. It is typical for all database connections, and almost every query makes usage of the [TempDB](https://www.spotlightcloud.io/solutions/root-cause-analysis#temp-db) database. It is like a heart for the SQL Server instance. Practically, we cannot work without the TempDB database.

**Let us have a quick summary of operations in which SQL Server uses TempDB.**

* Order by and Group by clause
* Index creation and online index rebuild
* Temp tables and table variables storage is in the TempDB database.
* Snapshot isolation and read committed snapshot isolation
* DBCC commands
* Hash joins static cursors, long-running transactions.
* XML queries
* Internal objects created by the SQL Server database engine.
* **Version stores**
* Multiple active record sets (MARS)

You can read more about TempDB in this [article.](https://www.sqlshack.com/overview-of-the-shrink-tempdb-database-in-sql-server/)

[](https://www.sqlshack.com/overview-of-the-shrink-tempdb-database-in-sql-server/)

SQL Server recreates this TempDB database on database engine Service restart. This restart can be due to the automatic or manual restart of SQL Service. We can query sys.databases for viewing the TempDB creation date that is also a start-up time of database service:

SELECT create\_date AS 'SQL Service Startup Time'  
FROM sys.databases  
WHERE name = 'tempdb';

### **tempdbblog3**

#### **TempDB SQL Server database configurations and best practices**

Sometimes, we notice unexpected growth of the TempDB database. The first step of avoiding this is to configure it as per the best practices. In this section, let's view the TempDB configuration is different versions of SQL Server.

#### **Configure TempDB for multiple DATA Files with even growth**

As per best practice, we should have multiple data files with even growth of all files. The number of files depends upon the logical processors.

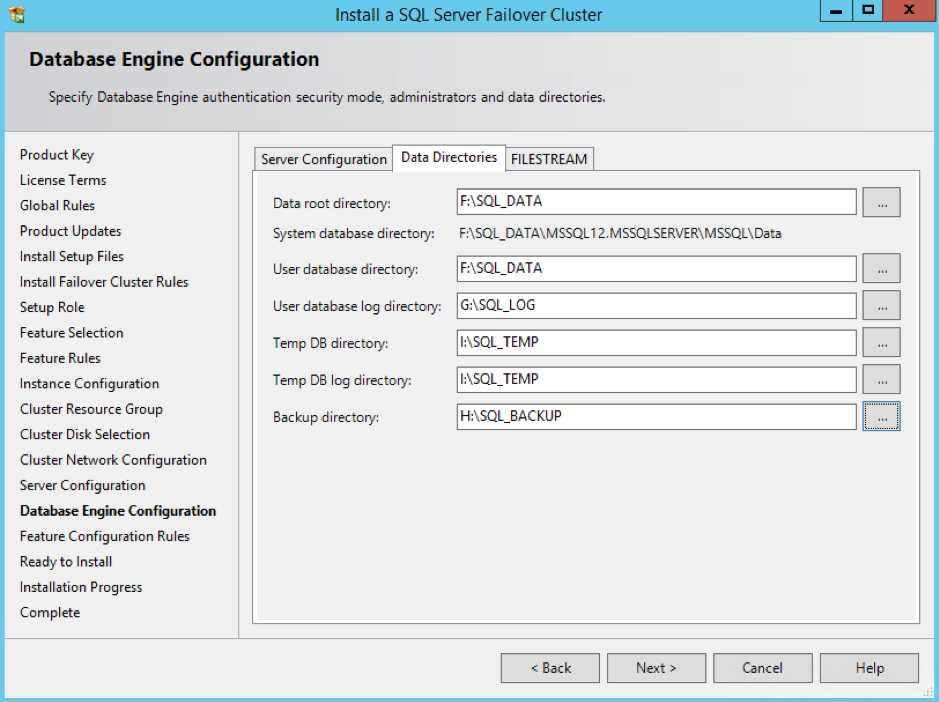
| **Processors** | **Number of TempDB data files** |
| --- | --- |
| Logical processors less than or equals to eight | Eight |
| Logical processors greater than eight | Start with eight data files.  Increase the data files in multiple of four and monitor the performance counters for TempDB contention. |

**For SQL Server versions before 2016, we do not have configuration available during the installation process.**

By default, it creates only one data and log file with the following configurations:

| TempDB Primary file | Auto grow data file by ten percent (until the disk is full) |
| --- | --- |
| TempDB log file | Auto grow  data file by ten percent (until the disk is full or maximum log file size reaches to 2 TB) |

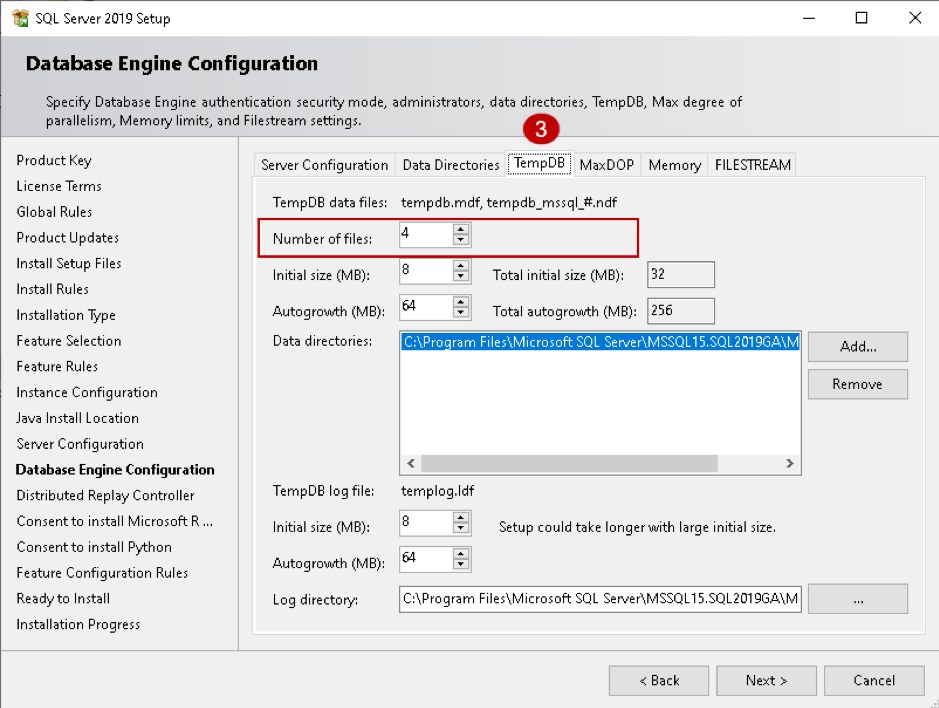
#### **SQL Server 2014 TempDB SQL Server database configuration**



SQL Server 2016 provides enhancements for TempDB configuration during the installation process as per the best practice:

| TempDB Primary and secondary files | Auto grow by 64 MB (until the disk is full) |
| --- | --- |
| TempDB log file | Auto grow by 64 MB (until the disk is full or maximum log file size reaches to 2 TB) |

#### **SQL Server 2016 onwards TempDB configuration**



#### **Uneven auto-growth SQL Server database TempDB**

SQL Server uses a round-robin method to fill multiple data files if they do not have the same size. Sometimes, we see that one file grows huge, but other files remain minimum growth. In case of uneven files, SQL Server uses the larger file for most of the queries, and it would continue growing:

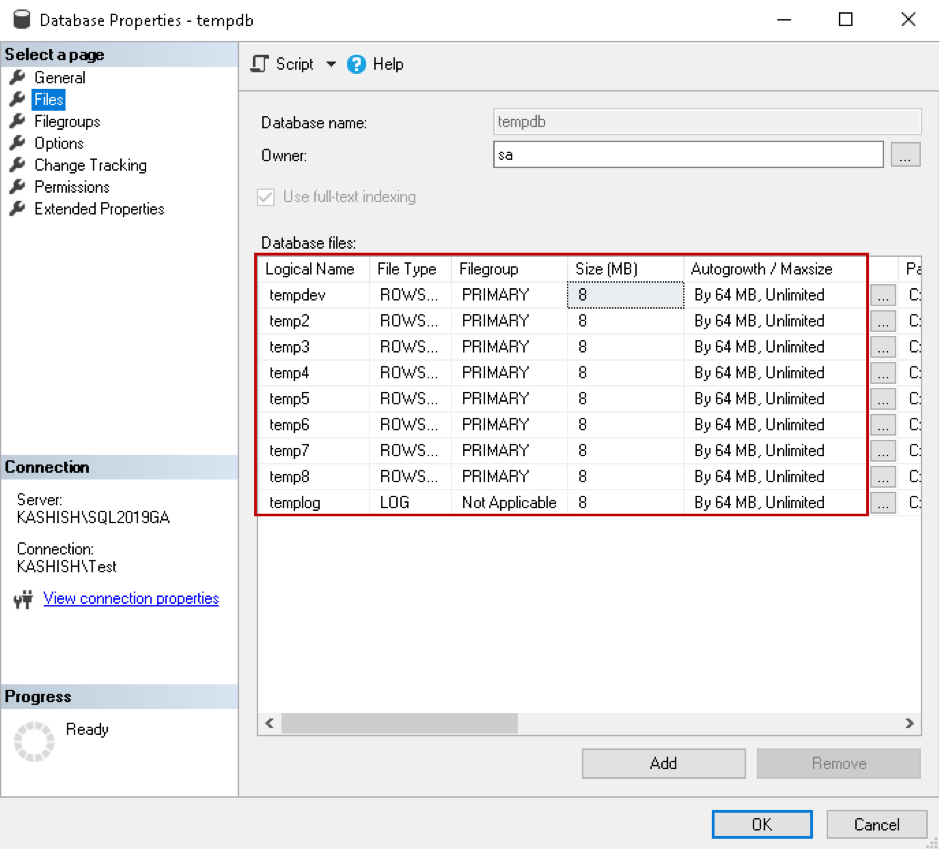
1. Use the same auto-growth of TempDB files (as discussed in the previous point).
2. Enable trace flag 1117 for growing all data files together in a database.

The second point is fixed automatically in SQL Server 2016 onwards however you should enable it in earlier versions. We do not require this trace flag in SQL Server 2016 and higher.

### 

#### **TempDB growth scenarios**

In this section, we will see a few scenarios for SQL Server database TempDB growth. In my SQL instance, I have eight data files with the following configuration:



Now, execute the following query to create a temporary table and perform data insertion. The temporary table storage location is the TempDB database. This query uses a CROSS JOIN operator with multiple columns and further sorts the results using the ORDER BY clause.

**Note:** Do not run this query in the production system; I am using it for demo purpose only.

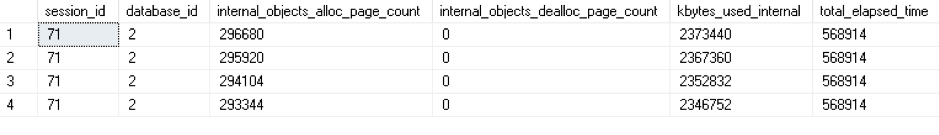
SELECT \*  
FROM sys.configurations  
CROSS JOIN sys.configurations SCA  
CROSS JOIN sys.configurations SCB  
CROSS JOIN sys.configurations SCC  
CROSS JOIN sys.configurations SCD  
CROSS JOIN sys.configurations SCE  
CROSS JOIN sys.configurations SCF  
CROSS JOIN sys.configurations SCG  
CROSS JOIN sys.configurations SCH  
ORDER BY SCA.name,  
SCA.value,  
SCC.value\_in\_use DESC;

This query will take a long time and might result in high CPU usage as well in your system. While the query is running, open another query window and use the DMV **sys.dm\_db\_task\_space\_usage to get information of**page allocation and deallocation activity by the task. We join this DMV with other DMV's to get the required information for the SQL Server database TempDB:

SELECT s.session\_id, dbu.database\_id  
, dbu.internal\_objects\_alloc\_page\_count, dbu.internal\_objects\_dealloc\_page\_count  
, (dbu.internal\_objects\_alloc\_page\_count - dbu.internal\_objects\_dealloc\_page\_count) \* 8192 / 1024 kbytes\_used\_internal  
, r.total\_elapsed\_time  
FROM sys.dm\_Exec\_requests r  
INNER JOIN sys.dm\_exec\_sessions s ON r.session\_id = s.session\_id  
LEFT JOIN sys.dm\_db\_task\_space\_usage dbu ON dbu.session\_id = r.session\_id  
AND dbu.request\_id = r.request\_id  
WHERE internal\_objects\_alloc\_page\_count > 0  
ORDER BY kbytes\_used\_internal DESC;

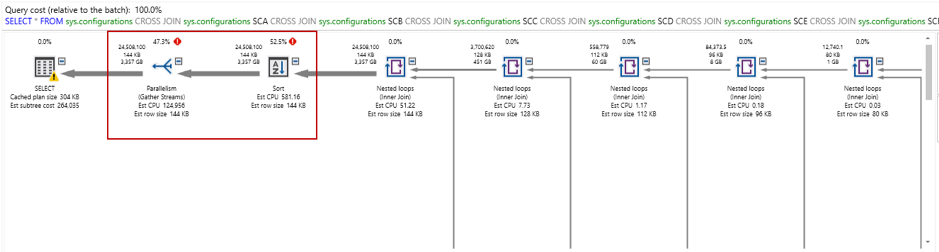
In the output, we see internal object page counts and their sizes (kbytes\_used\_internal) for the session ID 55.

SQL Server query optimizer executing this query in a parallel model; therefore, we can see multiple session ID 71 in the output:

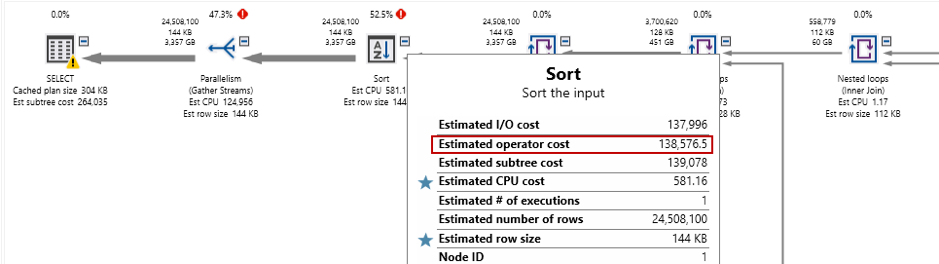


You can also view the estimated execution plan, and as shown below, we get two costly operators:

* Parallelism: 47.3%
* Sort: 52.3%

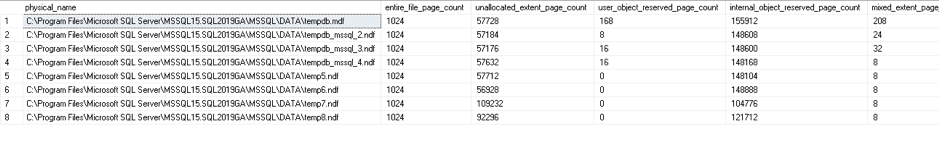


In the sort operator, we can see high estimated operator cost 138,576.5:



The following query uses DMV **sys.dm\_db\_file\_space\_usage**and joins it with sys.master\_files to check the allocated and unallocated extent page counts in the SQL Server database TempDB while the query is executing:

select mf.physical\_name, mf.size as entire\_file\_page\_count,  
dfsu.unallocated\_extent\_page\_count,  
dfsu.user\_object\_reserved\_page\_count,  
dfsu.internal\_object\_reserved\_page\_count,  
dfsu.mixed\_extent\_page\_count  
from sys.dm\_db\_file\_space\_usage dfsu  
join sys.master\_files as mf  
on mf.database\_id = dfsu.database\_id  
and mf.file\_id = dfsu.file\_id



We can monitor the query execution, its usage in the TempDB database and if required, kill the process to release the space immediately. We should also optimize the query causing massive TempDB growth.

#### **Monitor SQL Server database TempDB usage using extended events**

Extended events are useful for TempDB database monitoring. We can add the following extended events using the query:

* database\_file\_size\_change
* databases\_log\_file\_used\_size\_changed

#### Create extended event

CREATE EVENT SESSION [TempDB Usage] ON SERVER  
ADD EVENT sqlserver.database\_file\_size\_change(  
  
ACTION(sqlserver.client\_hostname,sqlserver.database\_id,sqlserver.session\_id,sqlserver.sql\_text)),  
ADD EVENT sqlserver.databases\_log\_file\_used\_size\_changed(  
  
ACTION(sqlserver.client\_hostname,sqlserver.database\_id,sqlserver.session\_id,sqlserver.sql\_text))  
ADD TARGET package0.event\_file(SET filename=N'TempDBUsage',max\_rollover\_files=(0))  
WITH (STARTUP\_STATE=OFF)  
GO

#### **Start extended event session**

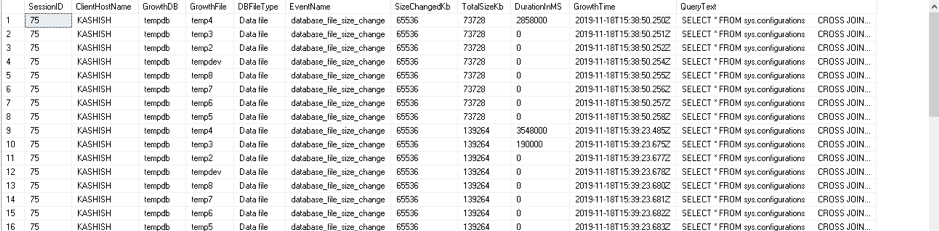
ALTER EVENT SESSION [TempDBTest] ON SERVER STATE = START;

Now, execute your workload to use the TempDB database and grow the data files. The extended events capture data file growth and query that caused this growth.

You can either view the extended event session file in SSMS GUI mode or use the following query to monitor TempDB growth.

#### **Monitor TempDB Growth**

SELECT [eventdata].[event\_data].[value]('(event/action[@name="session\_id"]/value)[1]', 'INT') AS [SessionID],  
[eventdata].[event\_data].[value]('(event/action[@name="client\_hostname"]/value)[1]', 'VARCHAR(100)') AS [ClientHostName],  
DB\_NAME([eventdata].[event\_data].[value]('(event/action[@name="database\_id"]/value)[1]', 'BIGINT')) AS [GrowthDB],  
[eventdata].[event\_data].[value]('(event/data[@name="file\_name"]/value)[1]', 'VARCHAR(200)') AS [GrowthFile],  
[eventdata].[event\_data].[value]('(event/data[@name="file\_type"]/text)[1]', 'VARCHAR(200)') AS [DBFileType],  
[eventdata].[event\_data].[value]('(event/@name)[1]', 'VARCHAR(MAX)') AS [EventName],  
[eventdata].[event\_data].[value]('(event/data[@name="size\_change\_kb"]/value)[1]', 'BIGINT') AS [SizeChangedKb],  
[eventdata].[event\_data].[value]('(event/data[@name="total\_size\_kb"]/value)[1]', 'BIGINT') AS [TotalSizeKb],  
[eventdata].[event\_data].[value]('(event/data[@name="duration"]/value)[1]', 'BIGINT') AS [DurationInMS],  
[eventdata].[event\_data].[value]('(event/@timestamp)[1]', 'VARCHAR(MAX)') AS [GrowthTime],  
[eventdata].[event\_data].[value]('(event/action[@name="sql\_text"]/value)[1]', 'VARCHAR(MAX)') AS [QueryText]  
FROM  
(  
SELECT CAST([event\_data] AS XML) AS [TargetData]  
FROM [sys].[fn\_xe\_file\_target\_read\_file]('C:\TEMP\TempDBusage\*.xel', NULL, NULL, NULL)  
) AS [eventdata]([event\_data])  
WHERE [eventdata].[event\_data].[value]('(event/@name)[1]', 'VARCHAR(100)') = 'database\_file\_size\_change'  
OR [eventdata].[event\_data].[value]('(event/@name)[1]', 'VARCHAR(100)') = 'databases\_log\_file\_used\_size\_changed'  
AND [eventdata].[event\_data].[value]('(event/@name)[1]', 'VARCHAR(MAX)') <> 'databases\_log\_file\_used\_size\_changed'  
ORDER BY [GrowthTime] ASC;



## ***Snapshot Isolation***

You might use snapshot isolation for your queries. In this isolation model, SQL Server stores the updated row versions of each transaction in the TempDB. In case of a large or long-running transaction, you can see a huge TempDB database.

You can execute the transaction with the SET command and specify Snapshot isolation:

SET TRANSACTION ISOLATION LEVEL SNAPSHOT;  
BEGIN TRAN;  
UPDATE [AdventureWorks].[Person].[Person]  
SET  
[Title] = 'Mr.';  
COMMIT TRAN;

You can also query **sys.databases** system view to check if any user database has snapshot isolation.

#### **Query to enable snapshot isolation on AdventureWorks database**

ALTER DATABASE AdventureWorks  
SET ALLOW\_SNAPSHOT\_ISOLATION ON  
GO

#### **Query to check user database with snapshot isolation**

SELECT \*  
FROM sys.databases  
WHERE(snapshot\_isolation\_state = 1  
OR is\_read\_committed\_snapshot\_on = 1)  
AND database\_id > 4;

In the following screenshot, you can see that the AdventureWorks database has snapshot isolation.

TempDB database also has snapshot isolation, but in the query, we have skipped database\_id less than 4:

tempdbblog12

We can use DMV sys.dm\_db\_file\_space\_usage to monitor version store in the TempDB:

SELECT GETDATE() AS runtime,  
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,  
SUM(unallocated\_extent\_page\_count) \* 8 AS freespace\_kb,  
SUM(mixed\_extent\_page\_count) \* 8 AS mixedextent\_kb  
FROM sys.dm\_db\_file\_space\_usage;

Here, we can see Version store size is 67968 KB. For a large or long-running transaction, you can see a huge SQL Server database TempDB size due to this version store:



Another case that might cause a huge size of the version store is Always on read-only Secondary replica. If you execute any query on the secondary database, it automatically uses the snapshot isolation level. As you know, snapshot isolation level copies row version in the TempDB.

**You should monitor the following perfmon counters:**

* **SQLServer:Transactions\Longest Transaction Running Time –** It captures the most extended active transaction.
* **SQLServer:Transactions\Version Store Size (KB)** – It captures the current size of all the version stores in TempDB.
* **SQLServer:Transactions\Version Cleanup rate (KB/s**) – You can use this counter to show the rate of version cleanup in TempDB
* **SQLServer:Transactions\Version Generation rate (KB/s)** – You can capture version store capture rate using this counter.

You should monitor TempDB growth for the versioning in Always on the secondary database as well. Kill the long-running sessions so that it can clear the versioning and reclaim space in the TempDB database.

#### 

#### **Conclusion**

In this article, we learned about the SQL Server database TempDB database best practice and different methods to detect, prevent unexpected growth. You should monitor TempDB regularly and configure different alerts to be proactive.

* TempDB size monitoring
* Drive space monitoring
* Long-running transactions