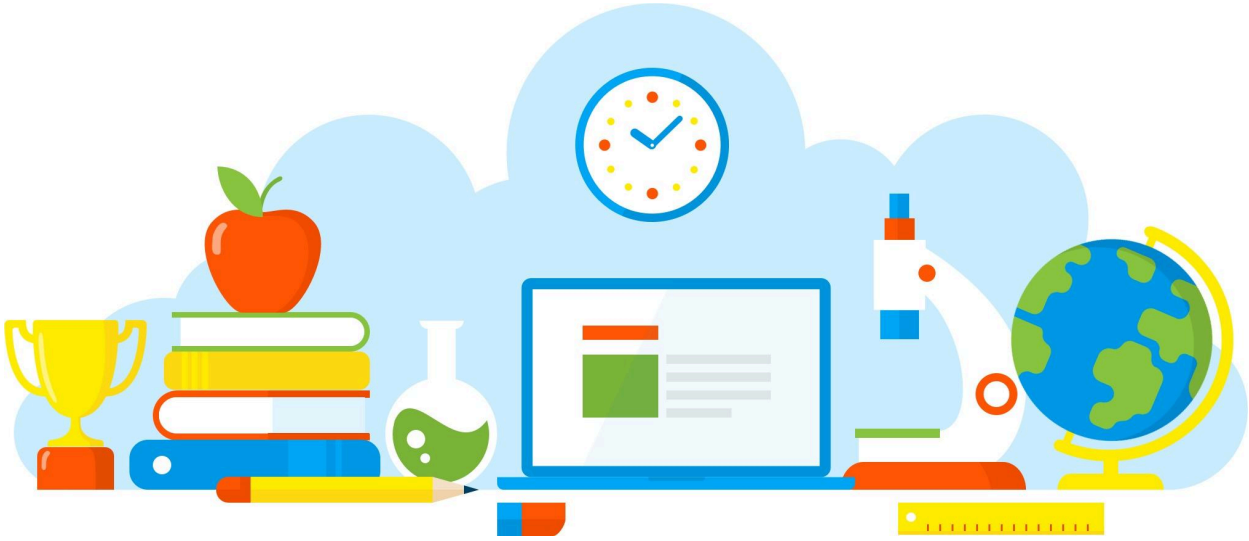


SQL Server Monitoring and Maintenance

20 Fundamental Techniques
for
SQL Server Database Health and Performance

Nayeem Islam

Nayeem Islam
12/08/2024

Introduction

Effective monitoring and maintenance of SQL Server are critical components of managing a healthy database environment. SQL Server, as a robust and complex database management system, requires ongoing attention to ensure optimal performance, security, and reliability. The 20 methods outlined in this guide provide a foundational approach to monitoring and maintaining SQL Server, helping database administrators (DBAs) and IT professionals proactively manage their environments.

These methods cover a wide range of topics, from monitoring storage and tracking database growth to detecting deadlocks and securing against unauthorized access. Each technique is designed to address common challenges encountered in SQL Server management, providing practical solutions to maintain database health, prevent downtime, and ensure data integrity.

While these methods serve as a starting point, it is important to recognize that every SQL Server environment is unique. Factors such as data volume, user behavior, application demands, and organizational policies can all influence how these techniques should be applied. As you implement these monitoring and maintenance strategies, you may need to adapt them to fit your specific scenario, ensuring that your SQL Server environment remains resilient and responsive to changing needs.

By mastering these fundamental practices, you lay the groundwork for more advanced database management, enabling you to keep your SQL Server instances running smoothly and securely. Whether you are new to SQL Server administration or looking to refine your existing practices, these 20 methods will help you build a strong foundation for ongoing success.

Let's Start!

Section 1: CPU Usage Monitoring

1. Description

CPU Usage Monitoring in SQL Server is a process that involves tracking the CPU utilization of SQL Server processes. This helps in identifying queries or processes that consume excessive CPU resources, which can degrade the performance of the SQL Server instance. By monitoring CPU usage, database administrators can pinpoint and optimize resource-intensive processes to maintain the server's efficiency.

2. How to Do

To monitor CPU usage in SQL Server:

1. **Open SQL Server Management Studio (SSMS)**: Connect to the SQL Server instance you want to monitor.
2. **Run the CPU Usage Monitoring Query**: Use a query that leverages Dynamic Management Views (DMVs) to retrieve details about CPU usage.
3. **Analyze the Results**: Identify sessions or queries that are using high CPU resources and take appropriate actions, such as optimizing queries or investigating further.

3. Code

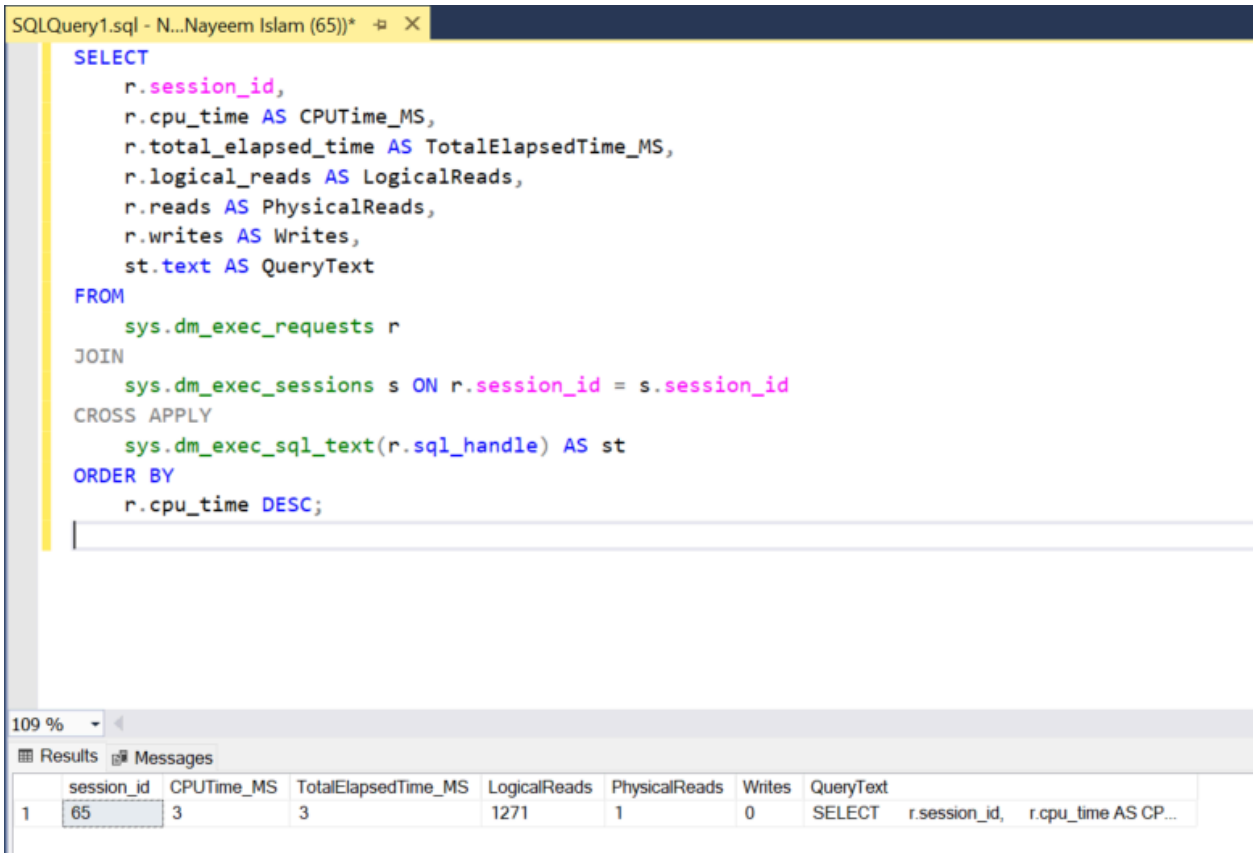
Use the following SQL query to monitor CPU usage by active sessions in SQL Server:

```
SELECT
    r.session_id,
    r.cpu_time AS CPUTime_MS,
    r.total_elapsed_time AS TotalElapsedTime_MS,
    r.logical_reads AS LogicalReads,
    r.reads AS PhysicalReads,
    r.writes AS Writes,
    st.text AS QueryText
FROM
    sys.dm_exec_requests r
JOIN
    sys.dm_exec_sessions s ON r.session_id = s.session_id
CROSS APPLY
    sys.dm_exec_sql_text(r.sql_handle) AS st
ORDER BY
    r.cpu_time DESC;
```

4. Explain Result of Code

- **session_id**: The unique identifier for each session currently connected to the SQL Server instance. It allows you to track which session is consuming CPU resources.

- **CPUTime_MS**: The total CPU time consumed by the session, in milliseconds. Higher values indicate more CPU usage.
- **TotalElapsedTime_MS**: The total time the query has been running, in milliseconds. This can help in understanding if a long-running query is contributing to high CPU usage.
- **LogicalReads**: The number of logical read operations performed by the session. High logical reads might indicate that the session is processing a large amount of data.
- **PhysicalReads**: The number of physical read operations performed by the session. High physical reads can indicate that the query is fetching data from disk, which might contribute to CPU usage.
- **Writes**: The number of write operations performed by the session. Monitoring writes can help in understanding how much data the session is modifying.
- **QueryText**: The actual SQL text being executed by the session. This helps in identifying which queries are responsible for the CPU load.



This structure provides a comprehensive view of how to monitor and interpret CPU usage in SQL Server, guiding users through each step from description to result interpretation.

Section 2: Memory Usage Monitoring

1. Description

Memory Usage Monitoring in SQL Server focuses on tracking the utilization of SQL Server's memory resources, including the buffer cache, procedure cache, and other memory allocations. Monitoring memory usage helps in detecting memory pressure, potential memory leaks, and understanding how efficiently SQL Server is using the available memory. Effective memory management is crucial for maintaining the performance and stability of SQL Server.

2. How to Do

To monitor memory usage in SQL Server:

1. **Access SQL Server Management Studio (SSMS)**: Connect to the SQL Server instance you wish to monitor.
2. **Run a Memory Usage Monitoring Query**: Use DMVs to retrieve detailed information about memory usage.
3. **Analyze the Results**: Identify areas where memory pressure might be occurring or where memory usage can be optimized.

3. Code

The following SQL query can be used to monitor memory usage in SQL Server:

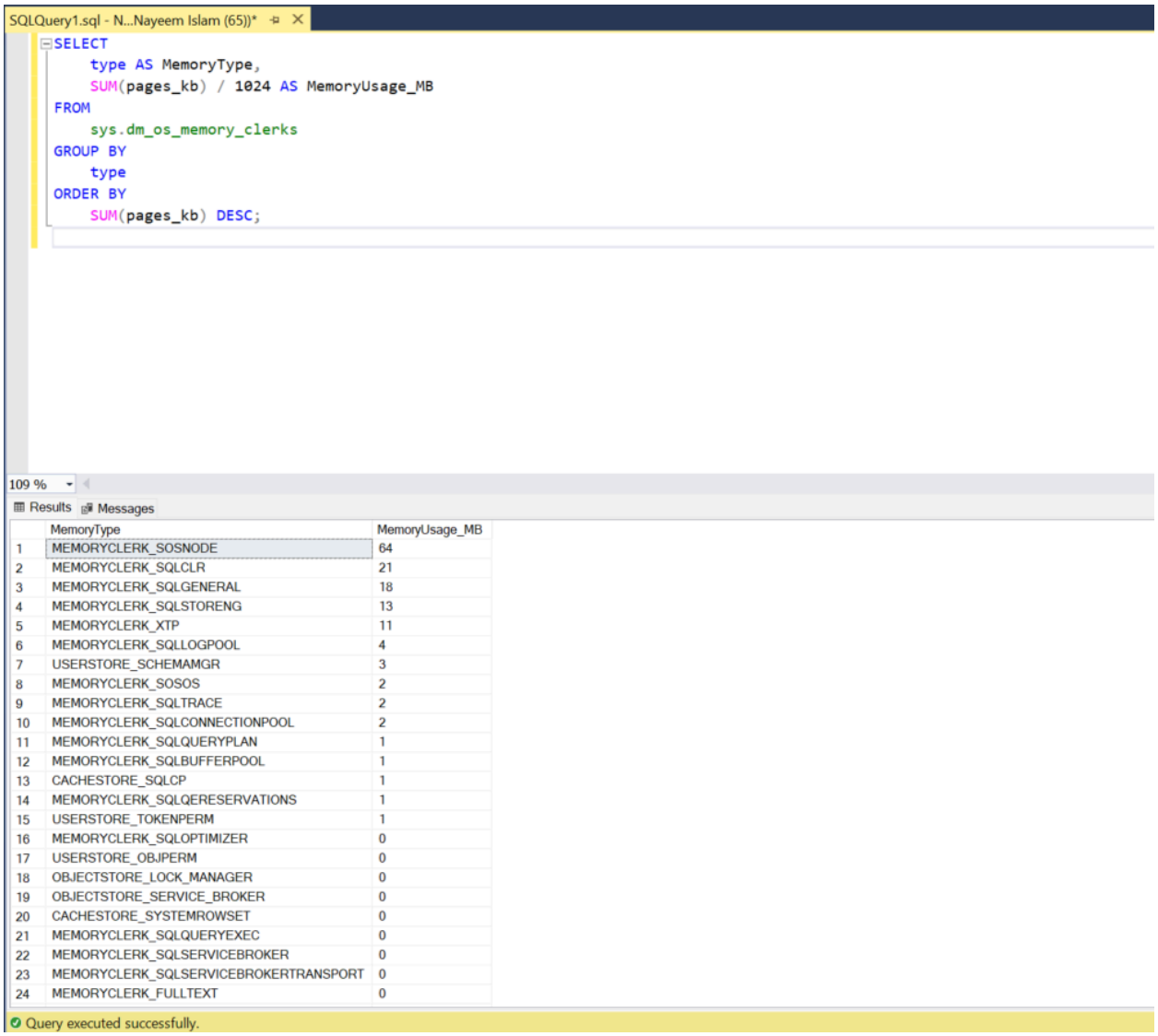
```
SELECT
    type AS MemoryType,
    SUM(pages_kb) / 1024 AS MemoryUsage_MB
FROM
    sys.dm_os_memory_clerks
GROUP BY
    type
ORDER BY
    SUM(pages_kb) DESC;
```

This query provides an overview of memory usage by different types of memory clerks, helping to identify where the most memory is being consumed.

4. Explain Result of Code

- **MemoryType**: Indicates the type of memory clerk, such as 'CACHESTORE_SQLCP' (for cached SQL plans), 'CACHESTORE_OBJCP' (for cached object plans), and others. These clerks are responsible for managing memory in different areas of SQL Server.
- **MemoryUsage_MB**: The total memory used by each type of clerk, in megabytes. This shows how much memory each component of SQL Server is consuming. A high value in certain memory types might indicate that SQL Server is holding onto more memory than necessary, potentially leading to memory pressure.

- **GROUP BY and ORDER BY**: The query groups the memory usage by clerk type and orders the results by the amount of memory used, making it easy to identify which clerks are using the most memory.



	MemoryType	MemoryUsage_MB
1	MEMORYCLERK_SOSNODE	64
2	MEMORYCLERK_SQLCLR	21
3	MEMORYCLERK_SQLGENERAL	18
4	MEMORYCLERK_SQLSTORENG	13
5	MEMORYCLERK_XTP	11
6	MEMORYCLERK_SQLLOGPOOL	4
7	USERSTORE_SCHEMAMGR	3
8	MEMORYCLERK_SOSOS	2
9	MEMORYCLERK_SQLTRACE	2
10	MEMORYCLERK_SQLCONNECTIONPOOL	2
11	MEMORYCLERK_SQLQUERYPLAN	1
12	MEMORYCLERK_SQLBUFFERPOOL	1
13	CACHESTORE_SQLCP	1
14	MEMORYCLERK_SQLQERESERVATIONS	1
15	USERSTORE_TOKENPERM	1
16	MEMORYCLERK_SQLOPTIMIZER	0
17	USERSTORE_OBJPERM	0
18	OBJECTSTORE_LOCK_MANAGER	0
19	OBJECTSTORE_SERVICE_BROKER	0
20	CACHESTORE_SYSTEMROWSET	0
21	MEMORYCLERK_SQLQUERYEXEC	0
22	MEMORYCLERK_SQLSERVICEBROKER	0
23	MEMORYCLERK_SQLSERVICEBROKERTRANSPORT	0
24	MEMORYCLERK_FULLTEXT	0

This structured approach helps you understand and manage memory usage within SQL Server, ensuring that memory resources are allocated efficiently to maintain optimal performance.

Section 3: Disk I/O Monitoring

1. Description

Disk I/O Monitoring in SQL Server involves tracking the read and write operations performed on databases. Monitoring disk I/O is crucial for identifying slow-performing disks, I/O bottlenecks, and understanding how efficiently data is being read from and written to the disk. High disk

latency or excessive I/O operations can significantly impact the performance of SQL Server, making it essential to monitor and optimize disk I/O.

2. How to Do

To monitor disk I/O in SQL Server:

1. **Open SQL Server Management Studio (SSMS)**: Connect to the SQL Server instance where you want to monitor disk I/O.
2. **Run a Disk I/O Monitoring Query**: Use the DMVs to gather detailed information about disk read/write activities and latency.
3. **Analyze the Results**: Identify any I/O bottlenecks or slow-performing disks that might be affecting the database performance.

3. Code

Here's a SQL query to monitor disk I/O performance in SQL Server:

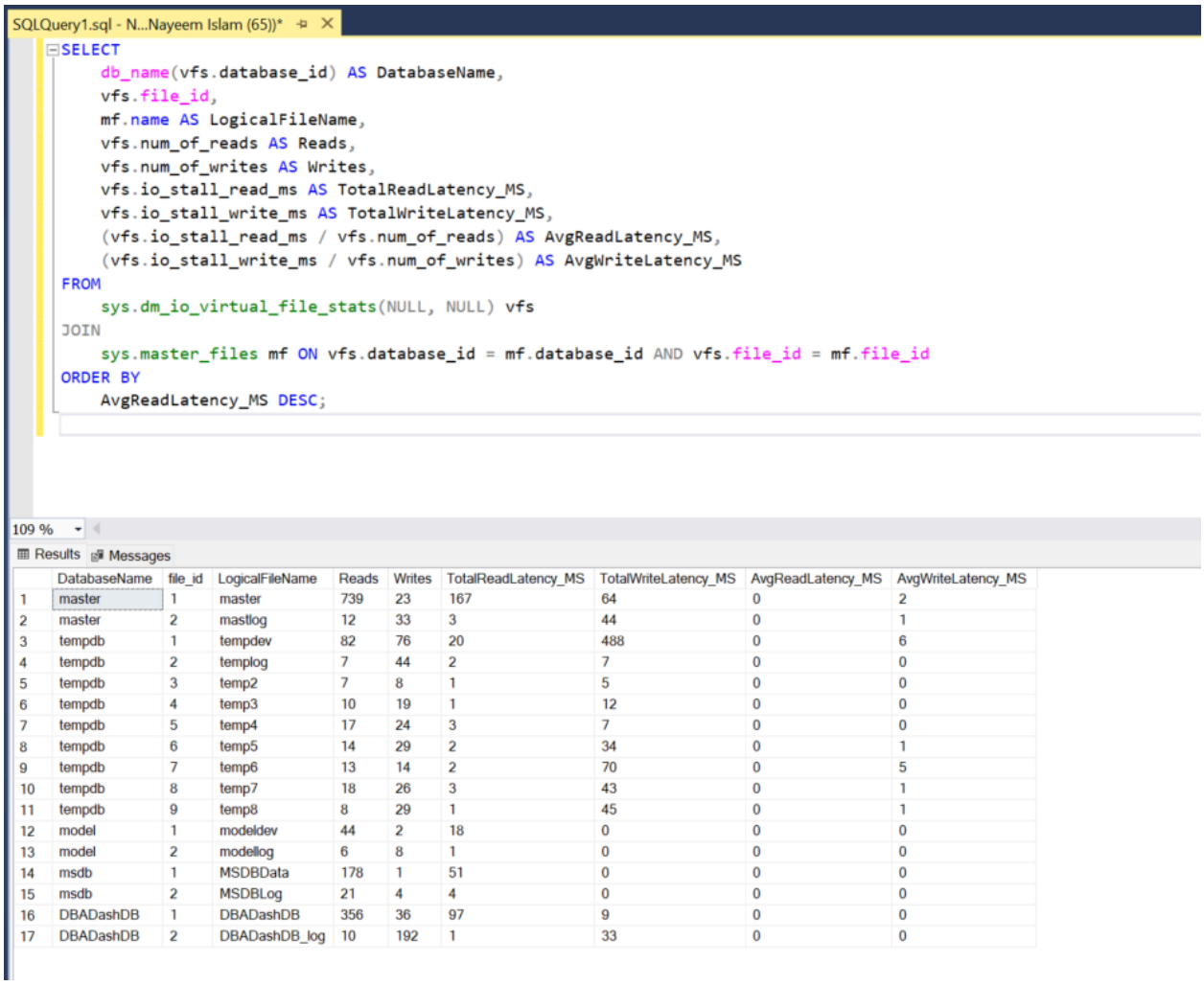
```
SELECT
    db_name(vfs.database_id) AS DatabaseName,
    vfs.file_id,
    mf.name AS LogicalFileName,
    vfs.num_of_reads AS Reads,
    vfs.num_of_writes AS Writes,
    vfs.io_stall_read_ms AS TotalReadLatency_MS,
    vfs.io_stall_write_ms AS TotalWriteLatency_MS,
    (vfs.io_stall_read_ms / vfs.num_of_reads) AS AvgReadLatency_MS,
    (vfs.io_stall_write_ms / vfs.num_of_writes) AS AvgWriteLatency_MS
FROM
    sys.dm_io_virtual_file_stats(NULL, NULL) vfs
JOIN
    sys.master_files mf ON vfs.database_id = mf.database_id AND vfs.file_id
= mf.file_id
ORDER BY
    AvgReadLatency_MS DESC;
```

This query provides insights into read and write operations, along with latency metrics, for each database file.

4. Explain Result of Code

- **DatabaseName**: The name of the database to which the file belongs.

- **file_id**: The ID of the file within the database. This helps in identifying specific files that might be experiencing high I/O activity.
- **LogicalFileName**: The logical name of the database file, providing a more understandable reference than the file ID.
- **Reads/Writes**: The total number of read and write operations performed on the file. A high number of operations can indicate a file that is heavily used, possibly leading to I/O contention.
- **TotalReadLatency_MS/TotalWriteLatency_MS**: The total time, in milliseconds, that SQL Server has waited for read and write operations to complete. High values suggest that I/O operations are slow, which can impact overall performance.
- **AvgReadLatency_MS/AvgWriteLatency_MS**: The average time taken per read or write operation, in milliseconds. This is a key metric for identifying slow I/O performance, with higher values indicating possible issues with the underlying disk subsystem.



	DatabaseName	file_id	LogicalFileName	Reads	Writes	TotalReadLatency_MS	TotalWriteLatency_MS	AvgReadLatency_MS	AvgWriteLatency_MS
1	master	1	master	739	23	167	64	0	2
2	master	2	mastlog	12	33	3	44	0	1
3	tempdb	1	tempdev	82	76	20	488	0	6
4	tempdb	2	templog	7	44	2	7	0	0
5	tempdb	3	temp2	7	8	1	5	0	0
6	tempdb	4	temp3	10	19	1	12	0	0
7	tempdb	5	temp4	17	24	3	7	0	0
8	tempdb	6	temp5	14	29	2	34	0	1
9	tempdb	7	temp6	13	14	2	70	0	5
10	tempdb	8	temp7	18	26	3	43	0	1
11	tempdb	9	temp8	8	29	1	45	0	1
12	model	1	modeldev	44	2	18	0	0	0
13	model	2	modellog	6	8	1	0	0	0
14	msdb	1	MSDBData	178	1	51	0	0	0
15	msdb	2	MSDBLog	21	4	4	0	0	0
16	DBADashDB	1	DBADashDB	356	36	97	9	0	0
17	DBADashDB	2	DBADashDB_log	10	192	1	33	0	0

This structure allows you to effectively monitor and troubleshoot disk I/O performance issues in SQL Server, ensuring that your database operations are not hindered by slow disk performance.

Section 4: Query Performance Monitoring

1. Description

Query Performance Monitoring in SQL Server is essential for tracking the execution of queries to identify slow-running or resource-intensive queries. Monitoring query performance helps in optimizing SQL statements, improving response times, and reducing the load on the server. By analyzing execution plans and performance metrics, database administrators can fine-tune queries and indexes to ensure efficient database operations.

2. How to Do

To monitor query performance in SQL Server:

1. **Access SQL Server Management Studio (SSMS)**: Connect to the SQL Server instance you wish to monitor.
2. **Run a Query Performance Monitoring Script**: Use DMVs to gather detailed information about the execution times and resource usage of queries.
3. **Analyze Execution Plans and Performance Metrics**: Identify slow or inefficient queries and optimize them by modifying the SQL code, updating statistics, or adjusting indexes.

3. Code

The following SQL query helps you monitor query performance by identifying the top resource-consuming queries:

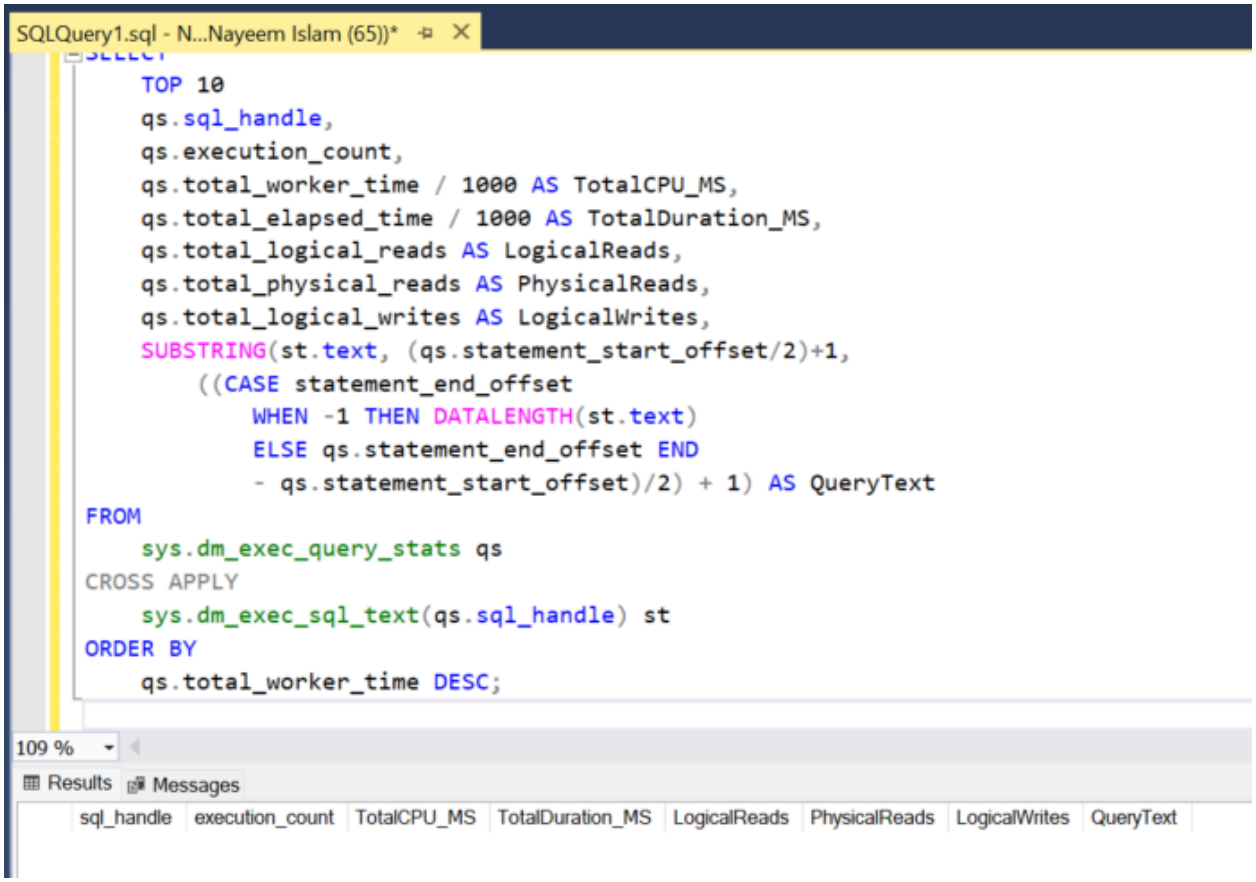
```
SELECT
    TOP 10
    qs.sql_handle,
    qs.execution_count,
    qs.total_worker_time / 1000 AS TotalCPU_MS,
    qs.total_elapsed_time / 1000 AS TotalDuration_MS,
    qs.total_logical_reads AS LogicalReads,
    qs.total_physical_reads AS PhysicalReads,
    qs.total_logical_writes AS LogicalWrites,
    SUBSTRING(st.text, (qs.statement_start_offset/2)+1,
        ((CASE statement_end_offset
            WHEN -1 THEN DATALENGTH(st.text)
            ELSE qs.statement_end_offset END
            - qs.statement_start_offset)/2) + 1) AS QueryText
FROM
    sys.dm_exec_query_stats qs
CROSS APPLY
    sys.dm_exec_sql_text(qs.sql_handle) st
```

```
ORDER BY
    qs.total_worker_time DESC;
```

This query retrieves information about the most CPU-intensive queries, including their execution count, total CPU time, duration, logical and physical reads, and the actual SQL text.

4. Explain Result of Code

- **sql_handle**: A unique identifier for the query. It can be used to retrieve additional details about the query from other DMVs.
- **execution_count**: The number of times the query has been executed. High execution counts combined with high resource usage can indicate that a frequently run query is a major contributor to overall load.
- **TotalCPU_MS**: The total CPU time consumed by the query, in milliseconds. Queries with high CPU usage can impact server performance and should be optimized.
- **TotalDuration_MS**: The total time taken to execute the query, in milliseconds. This helps in identifying long-running queries that might need optimization.
- **LogicalReads/PhysicalReads**: The number of logical and physical read operations performed by the query. High read operations may indicate that the query is processing a large amount of data, possibly leading to performance issues.
- **LogicalWrites**: The number of logical write operations. Monitoring write operations is important for understanding the impact of the query on data modification.
- **QueryText**: The actual SQL statement being executed. This is crucial for identifying and understanding the query that is consuming resources, allowing you to focus on optimizing it.



This section provides a comprehensive approach to monitoring and optimizing query performance, ensuring that SQL Server runs efficiently and queries are executed as quickly as possible.

Section 5: Lock and Wait Monitoring

1. Description

Lock and Wait Monitoring in SQL Server involves tracking and logging locks, blocking sessions, and deadlocks. These issues can significantly impact database performance by causing queries to wait unnecessarily, leading to delays and potential timeouts. Monitoring locks and waits helps database administrators identify problematic queries and processes, allowing them to take corrective actions to optimize database performance.

2. How to Do

To monitor locks and waits in SQL Server:

1. **Access SQL Server Management Studio (SSMS)**: Connect to your SQL Server instance.

2. **Run a Lock and Wait Monitoring Query**: Use DMVs to retrieve information about current locks, wait types, and blocking sessions.
3. **Analyze the Results**: Identify processes that are holding locks or are blocked, and take action to resolve or optimize the situation.

3. Code

Use the following SQL query to monitor locks and waits in SQL Server:

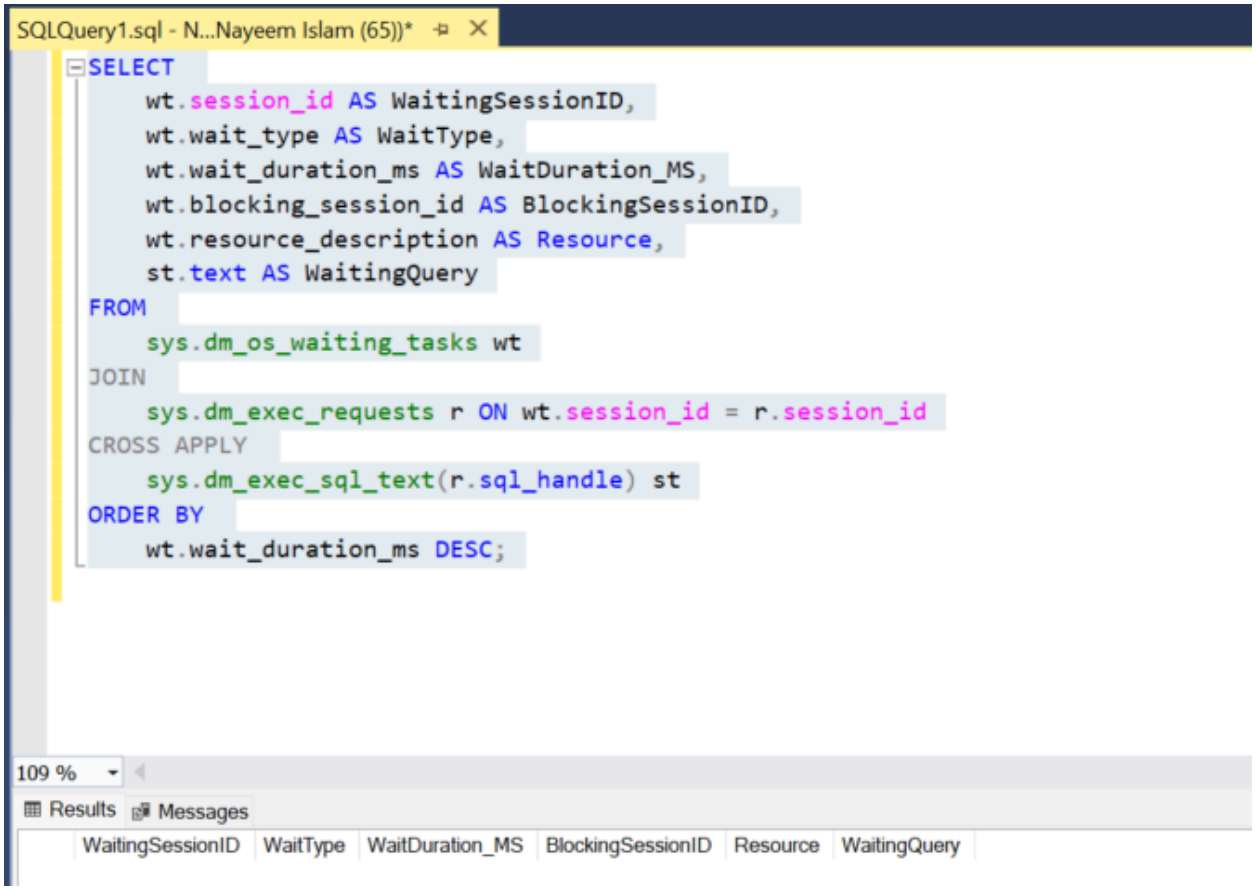
```
SELECT
    wt.session_id AS WaitingSessionID,
    wt.wait_type AS WaitType,
    wt.wait_duration_ms AS WaitDuration_MS,
    wt.blocking_session_id AS BlockingSessionID,
    wt.resource_description AS Resource,
    st.text AS WaitingQuery
FROM
    sys.dm_os_waiting_tasks wt
JOIN
    sys.dm_exec_requests r ON wt.session_id = r.session_id
CROSS APPLY
    sys.dm_exec_sql_text(r.sql_handle) st
ORDER BY
    wt.wait_duration_ms DESC;
```

This query retrieves information about sessions that are currently waiting, the type of wait, the session that is blocking them, and the SQL query that is being blocked.

4. Explain Result of Code

- **WaitingSessionID**: The ID of the session that is currently waiting. This helps in identifying which session is experiencing delays due to locks or other wait conditions.
- **WaitType**: The type of wait the session is experiencing (e.g., `LCK_M_S` for shared locks). Different wait types can indicate different performance issues, such as locking, I/O waits, or network delays.
- **WaitDuration_MS**: The duration, in milliseconds, that the session has been waiting. Longer wait durations can indicate more severe performance bottlenecks.
- **BlockingSessionID**: The ID of the session that is causing the wait by holding the required resource. This is crucial for identifying and resolving blocking issues.
- **Resource**: Describes the resource that the session is waiting for, such as a specific table or row. Understanding the resource helps in pinpointing the exact cause of the delay.

- **WaitingQuery**: The SQL query that is being blocked. This allows you to understand which operation is being delayed and helps in optimizing or restructuring the query to avoid such locks in the future.



This section enables database administrators to effectively monitor and manage locks and waits in SQL Server, ensuring that performance is not degraded by unnecessary delays.

Section 6: Database Size and Growth Monitoring

1. Description

Database Size and Growth Monitoring in SQL Server involves tracking the size of databases and their growth rates over time. This is crucial for capacity planning, ensuring that there is enough storage available to accommodate future growth, and identifying unexpected growth patterns that may indicate issues such as excessive data accumulation or inefficient storage use.

2. How to Do

To monitor database size and growth in SQL Server:

1. **Open SQL Server Management Studio (SSMS)**: Connect to the SQL Server instance where you want to monitor database size and growth.
2. **Run a Database Size Monitoring Query**: Use DMVs to gather information about the current size of databases and their historical growth patterns.
3. **Analyze the Results**: Review the data to identify trends in database growth, ensure that storage capacity is adequate, and detect any unusual growth that might require investigation.

3. Code

Use the following SQL query to monitor the size and growth of databases:

```
WITH GrowthData AS (
    SELECT
        database_id,
        type_desc AS FileType,
        size * 8 / 1024 AS CurrentSize_MB,
        growth * 8 / 1024 AS Growth_MB
    FROM
        sys.master_files
)
SELECT
    DB_NAME(database_id) AS DatabaseName,
    FileType,
    SUM(CurrentSize_MB) AS TotalSize_MB,
    SUM(Growth_MB) AS GrowthIncrement_MB
FROM
    GrowthData
GROUP BY
    database_id, FileType
ORDER BY
    TotalSize_MB DESC;
```

This query retrieves the current size and growth increments for each database file in megabytes, grouped by database and file type.

4. Explain Result of Code

- **DatabaseName**: The name of the database. This helps you identify which database's size and growth you are monitoring.
- **FileType**: Indicates whether the file is a data file (`ROWS`) or a log file (`LOG`). This distinction is important because data files and log files have different growth patterns and storage requirements.

- **TotalSize_MB**: The total current size of the database file, in megabytes. This provides a snapshot of how much space the database is currently using.
- **GrowthIncrement_MB**: The configured growth increment for the file, in megabytes. This shows how much the database file will grow by when it needs more space, helping in planning for future storage needs.

SQLQuery1.sql - N...Nayeem Islam (65))*

```sql
WITH GrowthData AS (
    SELECT
        database_id,
        type_desc AS FileType,
        size * 8 / 1024 AS CurrentSize_MB,
        growth * 8 / 1024 AS Growth_MB
    FROM
        sys.master_files
)
SELECT
    DB_NAME(database_id) AS DatabaseName,
    FileType,
    SUM(CurrentSize_MB) AS TotalSize_MB,
    SUM(Growth_MB) AS GrowthIncrement_MB
FROM
    GrowthData
GROUP BY
    database_id, FileType
ORDER BY
    TotalSize_MB DESC;
```

109 %

Results | Messages

	DatabaseName	FileType	TotalSize_MB	GrowthIncrement_MB
1	DBADashDB	LOG	136	64
2	DBADashDB	ROWS	72	64
3	tempdb	ROWS	64	512
4	msdb	ROWS	18	0
5	model	ROWS	8	64
6	tempdb	LOG	8	64
7	model	LOG	8	64
8	msdb	LOG	5	0
9	master	ROWS	5	0
10	master	LOG	2	0

This structure helps you monitor and manage the size and growth of your SQL Server databases effectively, ensuring that storage resources are used efficiently and that you can plan for future needs.

Section 7: Index Fragmentation Monitoring

1. Description

Index Fragmentation Monitoring in SQL Server focuses on tracking the fragmentation levels of indexes within your databases. Fragmentation occurs when the logical order of index pages does not match the physical order, leading to inefficient data access. Monitoring and managing index fragmentation is crucial for maintaining query performance, as fragmented indexes can slow down data retrieval and increase I/O operations.

2. How to Do

To monitor index fragmentation in SQL Server:

1. **Access SQL Server Management Studio (SSMS)**: Connect to the SQL Server instance where you want to monitor index fragmentation.
2. **Run an Index Fragmentation Monitoring Query**: Use DMVs to analyze the fragmentation levels of indexes in your databases.
3. **Analyze and Address Fragmentation**: Based on the results, decide whether to reorganize or rebuild the indexes to optimize performance.

3. Code

The following SQL query helps monitor the fragmentation of indexes:

```
SELECT
    dbschemas.[name] AS SchemaName,
    dbtables.[name] AS TableName,
    dbindexes.[name] AS IndexName,
    indexstats.avg_fragmentation_in_percent AS FragmentationPercentage,
    indexstats.page_count AS PageCount
FROM
    sys.dm_db_index_physical_stats(NULL, NULL, NULL, NULL, 'LIMITED') AS
indexstats
JOIN
    sys.tables dbtables ON dbtables.[object_id] = indexstats.[object_id]
JOIN
    sys.schemas dbschemas ON dbtables.[schema_id] = dbschemas.[schema_id]
JOIN
    sys.indexes AS dbindexes ON dbindexes.[object_id] =
indexstats.[object_id]
    AND indexstats.index_id = dbindexes.index_id
WHERE
    indexstats.database_id = DB_ID()
```

```
ORDER BY
    indexstats.avg_fragmentation_in_percent DESC;
```

This query retrieves information about the fragmentation level of indexes, organized by schema and table.

4. Explain Result of Code

- **SchemaName**: The schema to which the table belongs. This helps in identifying the scope of the fragmentation within different parts of the database.
- **TableName**: The name of the table that contains the index. This shows which tables are affected by fragmentation.
- **IndexName**: The name of the index being analyzed. This helps you identify specific indexes that may require maintenance.
- **FragmentationPercentage**: The percentage of fragmentation in the index. Higher percentages indicate more significant fragmentation, which may necessitate action. Generally, a fragmentation level above 30% suggests that the index should be rebuilt, while levels between 10% and 30% may benefit from reorganization.
- **PageCount**: The number of pages in the index. Larger indexes with high page counts are more likely to suffer performance degradation due to fragmentation.

SQLQuery1.sql - N...Nayeem Islam (65))*

```sql
SELECT
    dbschemas.[name] AS SchemaName,
    dbtables.[name] AS TableName,
    dbindexes.[name] AS IndexName,
    indexstats.avg_fragmentation_in_percent AS FragmentationPercentage,
    indexstats.page_count AS PageCount
FROM
    sys.dm_db_index_physical_stats(NULL, NULL, NULL, NULL, 'LIMITED') AS indexstats
JOIN
    sys.tables dbtables ON dbtables.[object_id] = indexstats.[object_id]
JOIN
    sys.schemas dbschemas ON dbtables.[schema_id] = dbschemas.[schema_id]
JOIN
    sys.indexes AS dbindexes ON dbindexes.[object_id] = indexstats.[object_id]
    AND indexstats.index_id = dbindexes.index_id
WHERE
    indexstats.database_id = DB_ID()
ORDER BY
    indexstats.avg_fragmentation_in_percent DESC;
```

109 %

Results | Messages

	SchemaName	TableName	IndexName	FragmentationPercentage	PageCount
1	dbo	spt_fallback_db	NULL	0	0
2	dbo	spt_fallback_dev	NULL	0	0
3	dbo	spt_fallback_usg	NULL	0	0
4	dbo	DiskUsage	PK__DiskUsag__3214EC2787D9DE46	0	0
5	dbo	spt_monitor	NULL	0	1
6	dbo	MyMonitorTable	NULL	0	0
7	dbo	MSreplication_options	NULL	0	1

By monitoring and addressing index fragmentation, you can ensure that your SQL Server databases perform efficiently, minimizing the time required for data retrieval and reducing unnecessary I/O operations.

Section 8: Integrity Checks (DBCC CHECKDB)

1. Description

Integrity checks in SQL Server involve verifying the consistency and integrity of the database structure. The `DBCC CHECKDB` command is the primary tool used for this purpose. It checks the physical and logical integrity of all the objects in the specified database, including tables, indexes, and more. Regular integrity checks are essential to ensure that the database remains free from corruption and to catch any issues early before they can cause significant problems.

2. How to Do

To perform integrity checks using `DBCC CHECKDB`:

1. **Open SQL Server Management Studio (SSMS)**: Connect to the SQL Server instance where you want to perform the integrity check.

2. **Run the DBCC CHECKDB Command**: Execute the `DBCC CHECKDB` command to verify the integrity of your database.
3. **Review the Results**: Analyze the results to determine if any corruption or integrity issues are detected. Take corrective actions if necessary.

3. Code

The following SQL command runs an integrity check on a specific database:

```
DBCC CHECKDB('YourDatabaseName')
WITH NO_INFOMSGS, ALL_ERRORMSGS;
```

- Replace `'YourDatabaseName'` with the name of the database you wish to check.
- The `WITH NO_INFOMSGS` option suppresses informational messages, and `ALL_ERRORMSGS` ensures that all error messages are displayed.

4. Explain Result of Code

- **DBCC CHECKDB**: This command performs a comprehensive check of the specified database's integrity.
- **NO_INFOMSGS**: Suppresses unnecessary informational messages, making the output easier to read.
- **ALL_ERRORMSGS**: Displays all error messages if any integrity issues are found.
- **Result Interpretation**: After running the command, SQL Server will return a message indicating whether any issues were found. If no issues are found, the result will confirm that the database integrity is intact. If issues are detected, SQL Server will provide details about the corruption, including affected objects, allowing you to take corrective actions such as restoring from a backup or repairing the database.

Regularly running `DBCC CHECKDB` helps ensure that your SQL Server databases remain healthy and free from corruption, protecting data integrity and minimizing the risk of data loss.

Section 9: Transaction Log Monitoring

1. Description

Transaction Log Monitoring in SQL Server is crucial for ensuring that the transaction log does not grow uncontrollably and that it is being properly maintained. The transaction log records all database modifications, making it essential for data recovery in case of a failure. Monitoring the transaction log helps in identifying when the log is growing excessively and ensures that log backups are being performed regularly to truncate the log and free up space.

2. How to Do

To effectively monitor transaction log usage in SQL Server:

1. **Access SQL Server Management Studio (SSMS)**: Connect to the SQL Server instance where you want to monitor the transaction logs.
2. **Run a Transaction Log Monitoring Query**: Use the provided SQL query to check the current size, used space, and free space of the transaction logs for each database.
3. **Analyze the Results**: Based on the output, determine if the transaction logs are growing too large or if there is insufficient free space, and take action such as performing log backups or shrinking the log if necessary.

3. Code

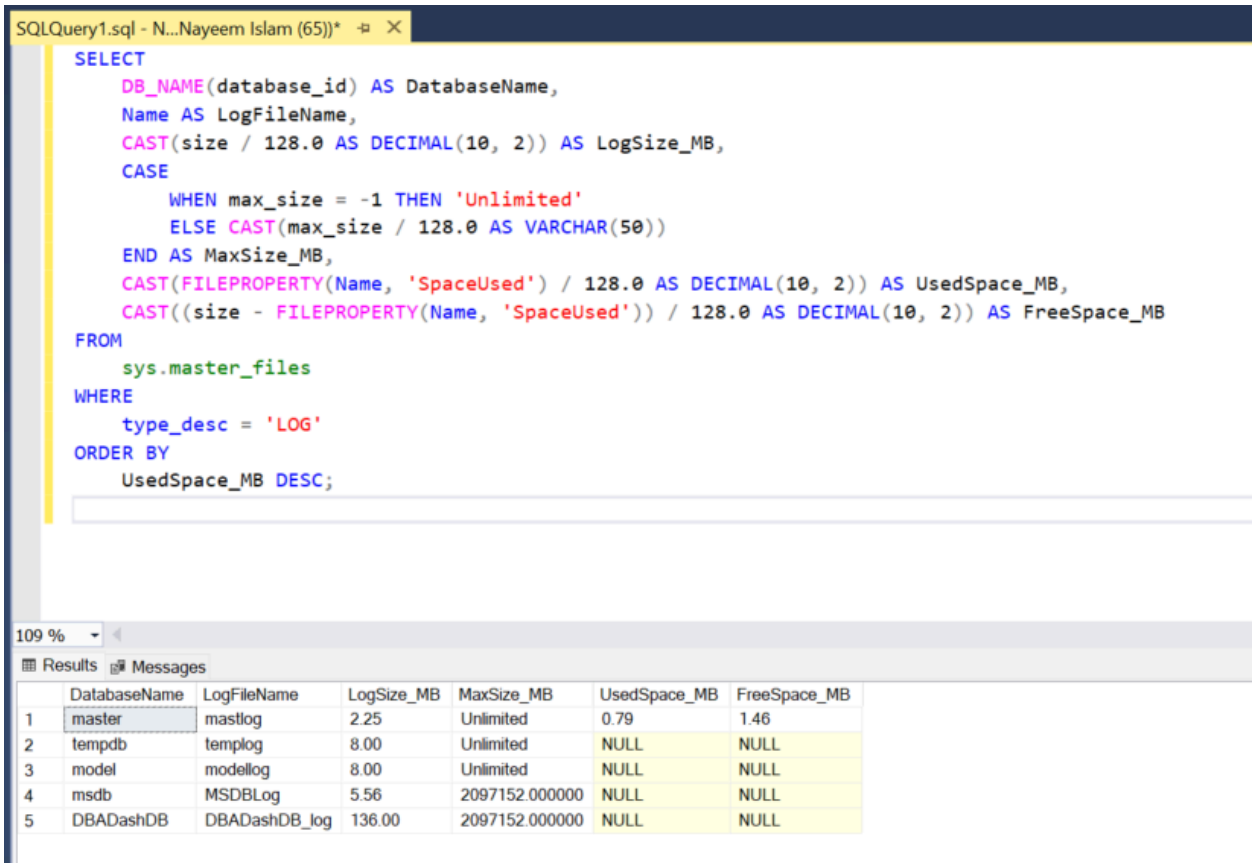
Use the following SQL query to monitor the size and usage of transaction logs:

```
SELECT
    DB_NAME(database_id) AS DatabaseName,
    Name AS LogFileName,
    CAST(size / 128.0 AS DECIMAL(10, 2)) AS LogSize_MB,
    CASE
        WHEN max_size = -1 THEN 'Unlimited'
        ELSE CAST(max_size / 128.0 AS VARCHAR(50))
    END AS MaxSize_MB,
    CAST(FILEPROPERTY(Name, 'SpaceUsed') / 128.0 AS DECIMAL(10, 2)) AS
UsedSpace_MB,
    CAST((size - FILEPROPERTY(Name, 'SpaceUsed')) / 128.0 AS DECIMAL(10,
2)) AS FreeSpace_MB
FROM
    sys.master_files
WHERE
    type_desc = 'LOG'
ORDER BY
    UsedSpace_MB DESC;
```

4. Explain Result of Code

- **DatabaseName**: The name of the database associated with the transaction log. This helps you identify which database's log you are monitoring.
- **LogFileName**: The name of the transaction log file. This provides a reference to the specific log file being analyzed.
- **LogSize_MB**: The total size of the transaction log file in megabytes. This shows how much space the log is currently consuming.

- **MaxSize_MB**: The maximum size the transaction log can grow to, in megabytes. If the log is set to "Unlimited," it can grow until it consumes all available disk space, which requires careful monitoring.
- **UsedSpace_MB**: The amount of space currently used in the transaction log, in megabytes. High used space can indicate that the log is not being truncated regularly, necessitating more frequent log backups.
- **FreeSpace_MB**: The amount of free space remaining in the transaction log, in megabytes. Monitoring free space helps in understanding when the log might need to grow or when it's time to perform a log backup to free up space.



	DatabaseName	LogFileName	LogSize_MB	MaxSize_MB	UsedSpace_MB	FreeSpace_MB
1	master	mastlog	2.25	Unlimited	0.79	1.46
2	tempdb	templog	8.00	Unlimited	NULL	NULL
3	model	modellog	8.00	Unlimited	NULL	NULL
4	msdb	MSDBLog	5.56	2097152.000000	NULL	NULL
5	DBADashDB	DBADashDB_log	136.00	2097152.000000	NULL	NULL

This approach ensures that you can effectively monitor the transaction log size and usage, prevent potential issues related to log growth, and maintain the overall health of your SQL Server databases.

Section 10: Login Monitoring (Revised)

1. Description

Login Monitoring in SQL Server involves tracking both successful and failed login attempts to ensure security and detect unauthorized access attempts. Monitoring login activity helps

database administrators keep track of who is accessing the SQL Server and from where, which is essential for maintaining a secure environment.

2. How to Do

To monitor login activity in SQL Server:

1. **Access SQL Server Management Studio (SSMS)**: Connect to the SQL Server instance.
2. **Enable Audit for Login Activity**: Consider creating server audits to capture detailed login activities.
3. **Run the Login Monitoring Query**: Use the provided SQL query to retrieve details about recent login attempts, focusing on failed logins to detect potential security threats.
4. **Analyze the Results**: Review the data to identify unusual login patterns or repeated failed login attempts, which might indicate unauthorized access attempts.

3. Code

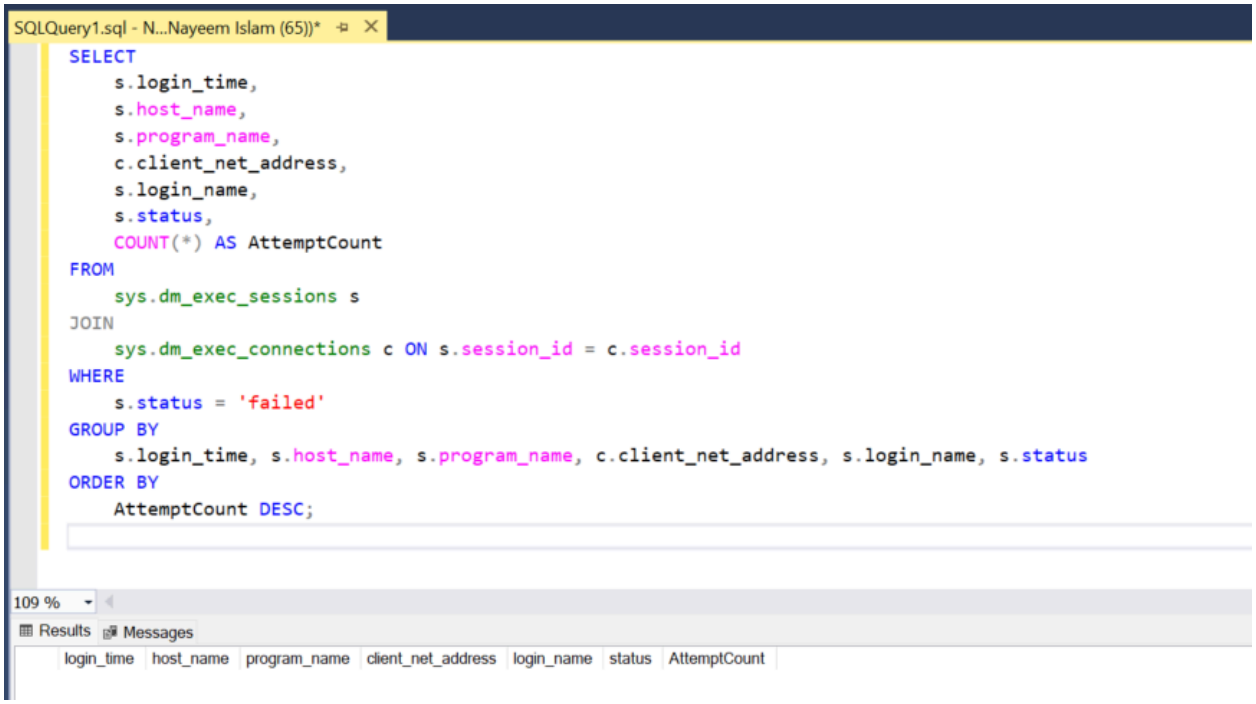
Use the following SQL query to monitor failed login attempts:

```
SELECT
    s.login_time,
    s.host_name,
    s.program_name,
    c.client_net_address,
    s.login_name,
    s.status,
    COUNT(*) AS AttemptCount
FROM
    sys.dm_exec_sessions s
JOIN
    sys.dm_exec_connections c ON s.session_id = c.session_id
WHERE
    s.status = 'failed'
GROUP BY
    s.login_time, s.host_name, s.program_name, c.client_net_address,
s.login_name, s.status
ORDER BY
    AttemptCount DESC;
```

4. Explain Result of Code

- **login_time**: The time at which the login attempt occurred.
- **host_name**: The name of the client machine that attempted to connect.

- **program_name**: The name of the application that attempted the login.
- **client_net_address**: The IP address of the client making the login attempt.
- **login_name**: The SQL Server login name used in the attempt.
- **status**: The status of the login attempt (focusing on 'failed' attempts).
- **AttemptCount**: The number of times the login attempt was made.



This approach allows database administrators to effectively monitor login activity, ensuring that any unauthorized access attempts are quickly detected and addressed to maintain the security of the SQL Server environment.

Section 11: Permission Changes Monitoring (Revised and Completed)

1. Description

Permission Changes Monitoring in SQL Server is essential for ensuring that only authorized personnel make changes to database permissions. By tracking and reviewing these changes, you can maintain the security and integrity of your database systems.

2. How to Do

1. **Set Up Auditing**: If not already done, create a server audit and audit specification to track permission changes.
2. **Run the Permission Changes Monitoring Query**: Use the corrected query to fetch and review the logs.

3. **Analyze the Results**: Ensure all permission changes comply with your organization's security policies.

3. Code

After setting up the audit, use this query to monitor permission changes:

```
SELECT
    event_time,
    session_server_principal_name AS ChangedBy,
    database_name,
    schema_name,
    object_name,
    statement AS ChangeStatement
FROM
    sys.fn_get_audit_file('C:\AuditLogs\*.sqlaudit', DEFAULT, DEFAULT)
WHERE
    action_id IN ('G', 'R', 'D')  -- G: GRANT, R: REVOKE, D: DENY
ORDER BY
    event_time DESC;
```

4. Explain Result of Code

- **event_time**: The timestamp of when the permission change occurred.
- **ChangedBy**: The principal (user or role) that executed the permission change.
- **database_name**: The database where the change occurred.
- **schema_name**: The schema within the database.
- **object_name**: The object (table, view, etc.) on which permissions were changed.
- **ChangeStatement**: The actual SQL statement executed for the permission change.

By regularly monitoring permission changes, you can ensure that your SQL Server environment remains secure, and any unauthorized changes can be promptly identified and addressed.

Section 12: SQL Injection Detection (Revised)

1. Description

SQL Injection Detection is critical for safeguarding your SQL Server from unauthorized data access and manipulation. SQL injection attacks involve the insertion of malicious SQL code into queries to manipulate or breach the database. Detecting these attempts in real-time helps in mitigating the risk and maintaining the security and integrity of the data.

2. How to Do

To detect potential SQL injection attacks:

1. **Access SQL Server Management Studio (SSMS)**: Connect to the SQL Server instance.
2. **Run Real-Time Query Monitoring**: Use the provided query to monitor active queries and detect suspicious patterns.
3. **Analyze Suspicious Queries**: Identify and respond to potential SQL injection attempts by analyzing the detected queries.

3. Code

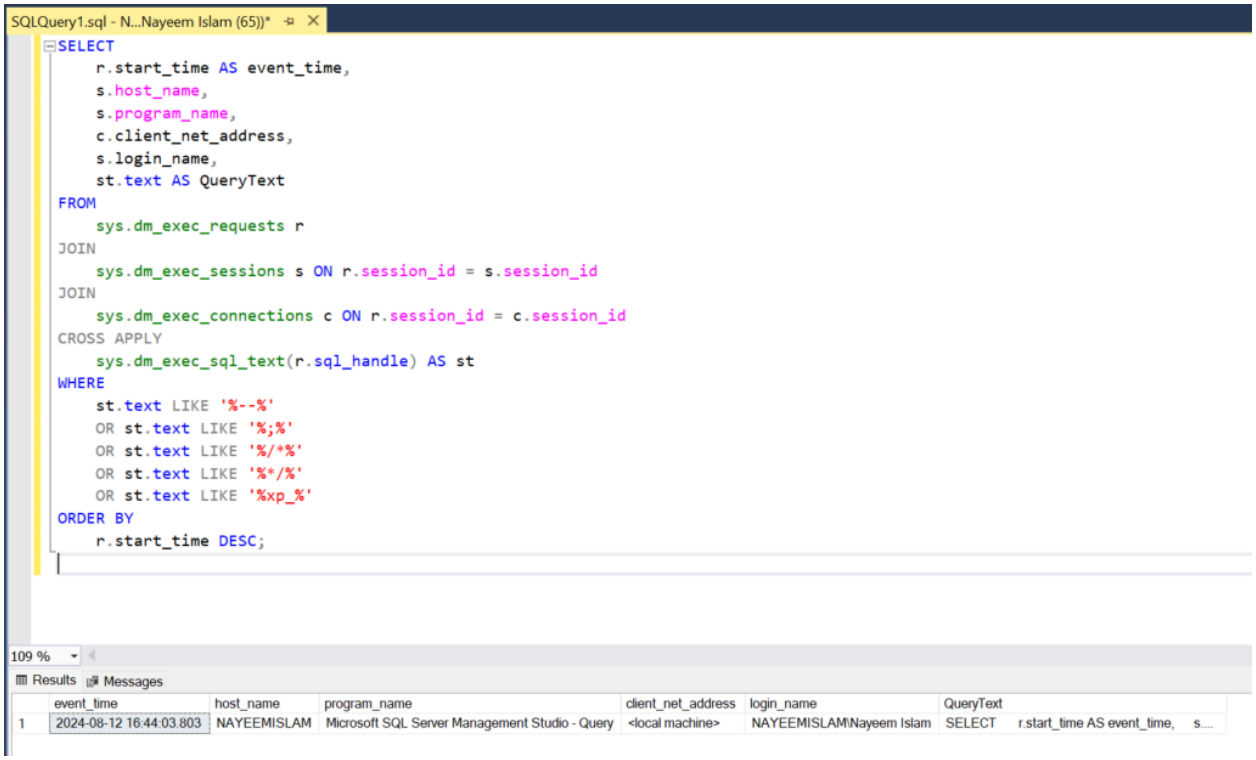
Use the following SQL query to detect potentially malicious queries indicative of SQL injection:

```
SELECT
    r.start_time AS event_time,
    s.host_name,
    s.program_name,
    c.client_net_address,
    s.login_name,
    st.text AS QueryText
FROM
    sys.dm_exec_requests r
JOIN
    sys.dm_exec_sessions s ON r.session_id = s.session_id
JOIN
    sys.dm_exec_connections c ON r.session_id = c.session_id
CROSS APPLY
    sys.dm_exec_sql_text(r.sql_handle) AS st
WHERE
    st.text LIKE '%--%'
    OR st.text LIKE '%;%'
    OR st.text LIKE '%/*%'
    OR st.text LIKE '%*/%'
    OR st.text LIKE '%xp_%'
ORDER BY
    r.start_time DESC;
```

4. Explain Result of Code

- **event_time**: The time when the query started execution.
- **host_name**: The client machine from which the query was executed.
- **program_name**: The application used to execute the query.
- **client_net_address**: The IP address of the client executing the query.
- **login_name**: The SQL Server login used for the session.

- **QueryText**: The full text of the executed query, highlighting potential SQL injection patterns.



This approach allows you to monitor and detect SQL injection attempts in real time, providing an immediate response to any potential threats.

Section 13: Audit Log Monitoring (Revised)

1. Description

Audit Log Monitoring involves tracking changes to audit settings and configurations. This ensures that auditing remains effective and that any unauthorized changes to audit mechanisms are promptly detected.

2. How to Do

1. **Ensure Auditing is Enabled**: Set up server audits if not already done.
2. **Run the Audit Log Monitoring Query**: Use the provided query to monitor changes to audit configurations.
3. **Review Logs**: Regularly review the logs to ensure compliance with security policies.

3. Code

Use the following SQL query to monitor changes to audit logs:

```
SELECT
    event_time,
    session_server_principal_name AS ChangedBy,
    action_id,
    succeeded,
    statement
FROM
    sys.fn_get_audit_file('C:\AuditLogs\*.sqlaudit', DEFAULT, DEFAULT)
WHERE
    action_id IN ('AUEC', 'AUGR', 'AUAF')
ORDER BY
    event_time DESC;
```

4. Explain Result of Code

- **event_time**: Timestamp of when the audit change occurred.
- **ChangedBy**: The user who made the change.
- **action_id**: Type of action performed.
- **succeeded**: Whether the action succeeded.
- **statement**: The SQL statement executed.

If you encounter any issues after setting up the audit, double-check the file paths and ensure that SQL Server has the necessary permissions to access the directory where the audit logs are stored.

Section 14: Uptime Monitoring

1. Description

Uptime Monitoring in SQL Server ensures that the database server is continuously available and running as expected. Monitoring uptime is crucial for maintaining the reliability of your database systems, as any unexpected downtime can impact business operations. By tracking uptime, database administrators can quickly detect and respond to any outages or unplanned downtime, ensuring that the SQL Server is always accessible when needed.

2. How to Do

To monitor uptime in SQL Server:

1. **Access SQL Server Management Studio (SSMS)**: Connect to the SQL Server instance you wish to monitor.
2. **Run an Uptime Monitoring Query**: Use a SQL query to retrieve information about the server's uptime, including when it was last restarted.

3. **Analyze the Results**: Regularly review the uptime information to ensure that the server is running continuously and to identify any patterns of unexpected restarts or downtime.

3. Code

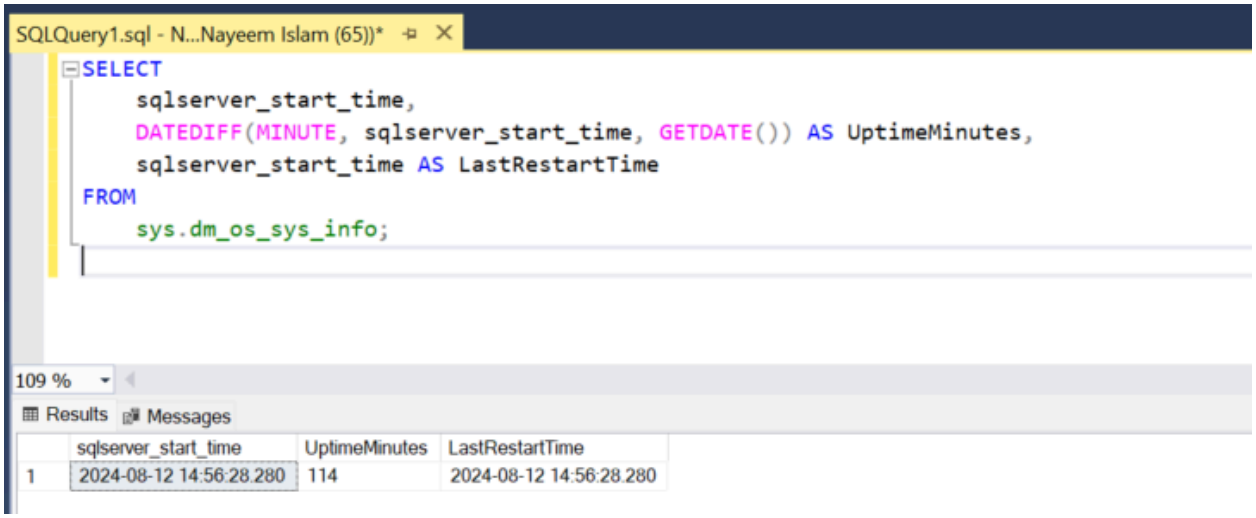
Here's a SQL query that helps monitor the uptime of your SQL Server instance:

```
SELECT
    sqlserver_start_time,
    DATEDIFF(MINUTE, sqlserver_start_time, GETDATE()) AS UptimeMinutes,
    sqlserver_start_time AS LastRestartTime
FROM
    sys.dm_os_sys_info;
```

This query retrieves the time when the SQL Server instance was last started and calculates the total uptime in minutes.

4. Explain Result of Code

- **sqlserver_start_time**: The timestamp when the SQL Server instance was last started. This indicates the last time the server was restarted, which can help identify any unplanned outages.
- **UptimeMinutes**: The total uptime in minutes since the last restart. This metric helps in understanding how long the server has been running without interruption.
- **LastRestartTime**: This is essentially the same as `sqlserver_start_time`, reiterating when the last restart occurred, which can be useful for confirming server availability.



By monitoring uptime, you ensure that your SQL Server instance is running reliably and that any unexpected downtime is quickly identified and addressed.

Section 15: Backup Status Monitoring (Revised)

1. Description

Backup Status Monitoring in SQL Server is essential for ensuring that your databases are properly backed up. Regular backups are critical for data protection and recovery, and monitoring their status ensures that they are performed on schedule and completed successfully.

2. How to Do

To monitor backup status effectively:

1. **Access SQL Server Management Studio (SSMS)**: Connect to the desired SQL Server instance.
2. **Run the Backup Status Monitoring Query**: Use the provided query to fetch the latest backup details, including the type and size of the backups.
3. **Review Backup Information**: Regularly review the data to confirm that backups are executed as planned and that there are no issues requiring attention.

3. Code

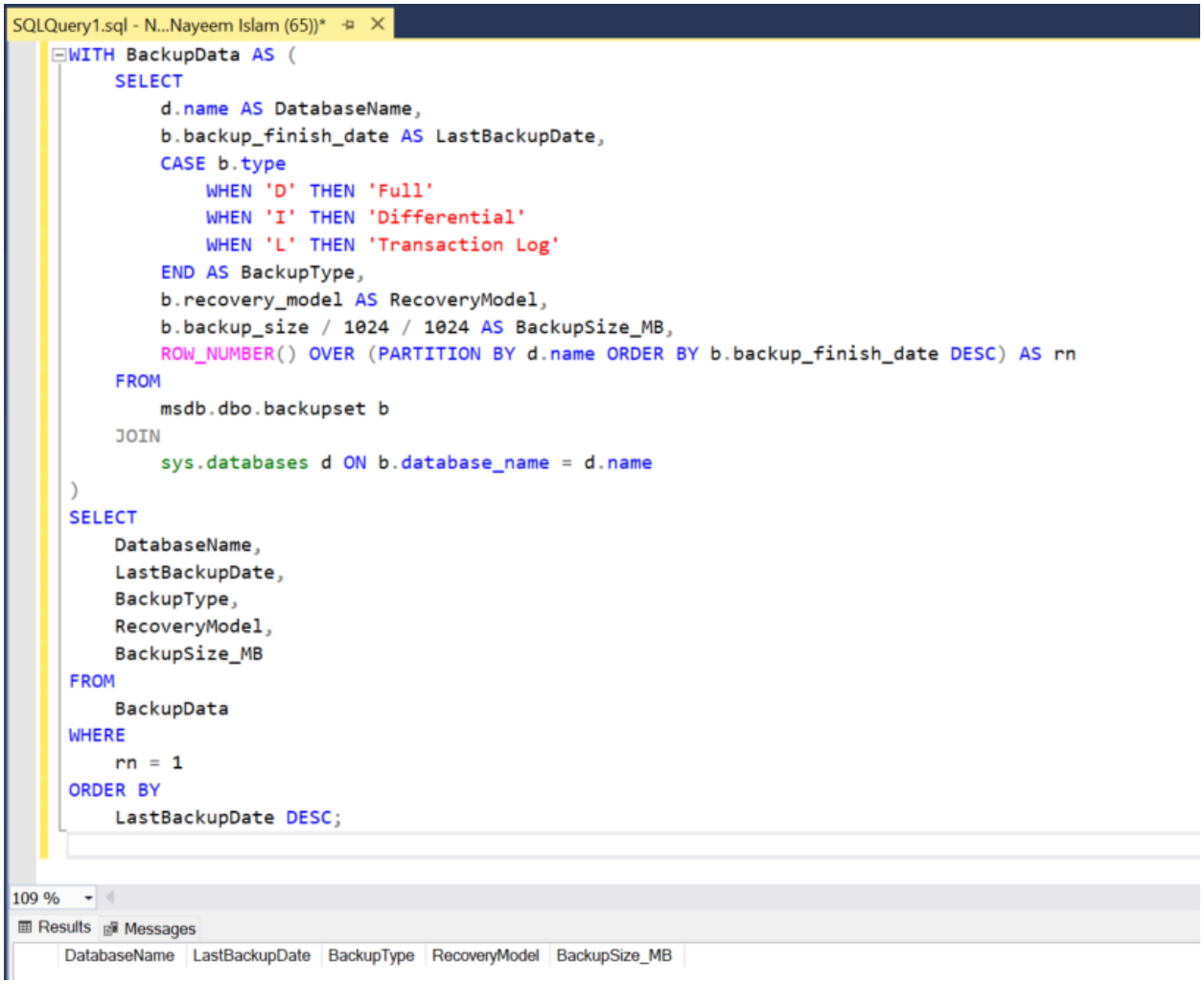
Use this query to monitor the latest backup status:

```
WITH BackupData AS (
    SELECT
        d.name AS DatabaseName,
        b.backup_finish_date AS LastBackupDate,
        CASE b.type
            WHEN 'D' THEN 'Full'
            WHEN 'I' THEN 'Differential'
            WHEN 'L' THEN 'Transaction Log'
        END AS BackupType,
        b.recovery_model AS RecoveryModel,
        b.backup_size / 1024 / 1024 AS BackupSize_MB,
        ROW_NUMBER() OVER (PARTITION BY d.name ORDER BY
b.backup_finish_date DESC) AS rn
    FROM
        msdb.dbo.backupset b
    JOIN
        sys.databases d ON b.database_name = d.name
)
SELECT
    DatabaseName,
    LastBackupDate,
    BackupType,
```

```
    RecoveryModel,
    BackupSize_MB
FROM
    BackupData
WHERE
    rn = 1
ORDER BY
    LastBackupDate DESC;
```

4. Explain Result of Code

- **DatabaseName**: The name of the database for which the backup was performed.
- **LastBackupDate**: The date and time of the most recent backup.
- **BackupType**: Indicates the type of the last backup (Full, Differential, or Transaction Log).
- **RecoveryModel**: The database's recovery model (e.g., Simple, Full).
- **BackupSize_MB**: The size of the last backup in megabytes.



This approach ensures that you are always informed of the latest backup status for each database, allowing you to maintain a reliable backup strategy.

Section 16: Replication Monitoring

1. Description

Replication Monitoring in SQL Server is essential for ensuring that data replication processes are running smoothly and efficiently. Monitoring the status of replication agents, such as the Merge Agent, Distribution Agent, and Log Reader Agent, helps ensure that data is accurately and consistently transferred between databases. Effective replication monitoring allows you to quickly identify and address any issues, preventing data discrepancies and maintaining database integrity.

2. How to Do

To monitor replication effectively in SQL Server:

1. **Access SQL Server Management Studio (SSMS)**: Connect to the SQL Server instance where replication is configured.
2. **Set Up Replication Monitoring**: Utilize built-in tools like Replication Monitor, or set up custom monitoring using SQL Server Agent jobs.
3. **Run the Replication Monitoring Procedure**: Use the provided stored procedure to check the status of replication-related jobs.
4. **Analyze the Results**: Regularly review the job statuses to ensure that replication is running as expected and to identify any issues that may require attention.

3. Code

Here's a stored procedure you can use to monitor the status of replication jobs:

```
CREATE PROCEDURE GetReplicationAgentStatus
AS
BEGIN
    SET NOCOUNT ON;
    SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

    -- Retrieve the replication jobs
    SELECT
        s.job_id,
        s.name AS JobName,
        s.enabled,
        c.name AS CategoryName
    INTO #JobList
    FROM
        msdb.dbo.sysjobs s
    INNER JOIN
        msdb.dbo.syscategories c ON s.category_id = c.category_id
    WHERE
        c.name IN ('REPL-Merge','REPL-Distribution','REPL-LogReader');

    -- Get the status of these jobs
    CREATE TABLE #xp_results
       (
        job_id                UNIQUEIDENTIFIER NOT NULL,
        last_run_date         INT              NOT NULL,
        last_run_time         INT              NOT NULL,
        next_run_date         INT              NOT NULL,
        next_run_time         INT              NOT NULL,
        next_run_schedule_id  INT              NOT NULL,
        requested_to_run      INT              NOT NULL,
        request_source        INT              NOT NULL,
```

```
        request_source_id     sysname          COLLATE database_default
NULL,
        running               INT              NOT NULL,
        current_step          INT              NOT NULL,
        current_retry_attempt INT              NOT NULL,
        job_state             INT              NOT NULL
       );

    INSERT INTO #xp_results
    EXEC master.dbo.xp_sqlagent_enum_jobs 1, '';

    -- Join and retrieve the job statuses
    SELECT
        j.name AS JobName,
        j.CategoryName,
        j.enabled AS IsEnabled,
        CASE WHEN r.running = 1 THEN 'Running' ELSE 'Stopped' END AS
AgentStatus
    FROM
        #JobList j
    INNER JOIN
        #xp_results r ON j.job_id = r.job_id;

    -- Drop temporary tables
    DROP TABLE #JobList, #xp_results;
END;
```

4. Explain Result of Code

- **JobName**: The name of the replication job, such as those related to merge, distribution, or log reading.
- **CategoryName**: The category of the job, indicating its role in replication.
- **IsEnabled**: Whether the job is currently enabled.
- **AgentStatus**: Indicates whether the replication agent is currently running or stopped. This is crucial for ensuring that all necessary replication processes are active and functioning correctly.

By running this stored procedure, you can monitor the status of your SQL Server replication jobs, helping ensure that your replication processes are running smoothly and efficiently.

Section 17: Cluster and Failover Monitoring (Final Revision)

1. Description

Cluster and Failover Monitoring is vital for maintaining high availability in SQL Server environments where databases are part of an availability group. Monitoring helps ensure that cluster nodes and replicas are operating correctly and that failover events are handled smoothly to prevent downtime.

2. How to Do

To effectively monitor clusters and failover events:

1. **Access SQL Server Management Studio (SSMS)**: Connect to the SQL Server instance within a cluster.
2. **Run Cluster and Failover Monitoring Queries**: Use the corrected queries to check the status of cluster nodes, availability groups, and failover events.
3. **Analyze the Results**: Regularly review the results to ensure the cluster nodes and availability groups are healthy and operational.

3. Code

Use these queries to monitor the cluster and failover status:

1. **Check Cluster Nodes Status:**

```
SELECT
    member_name AS NodeName,
    member_type_desc AS NodeType,
    member_state_desc AS NodeStatus
FROM
    sys.dm_hadr_cluster_members;
```

2. **Check Availability Groups and Replicas Status:**

```
SELECT
    ag.name AS AvailabilityGroupName,
    r.replica_server_name AS ReplicaServerName,
    r.availability_mode_desc AS AvailabilityMode,
    r.failover_mode_desc AS FailoverMode,
    rs.role_desc AS Role,
    rs.synchronization_health_desc AS SynchronizationHealth
FROM
    sys.availability_groups AS ag
JOIN
    sys.availability_replicas AS r ON ag.group_id = r.group_id
JOIN
    sys.dm_hadr_availability_replica_states AS rs ON r.replica_id =
```

```
rs.replica_id
WHERE
    rs.group_id = ag.group_id;
```

3. **Monitor Failover Events:**

```
SELECT
    ag.name AS AvailabilityGroupName,
    rs.replica_server_name AS ReplicaServerName,
    rs.last_commit_time AS LastCommitTime,
    rs.synchronization_health_desc AS SynchronizationHealth,
    rcs.last_redone_time AS LastRedoTime
FROM
    sys.dm_hadr_database_replica_states AS rs
JOIN
    sys.availability_groups AS ag ON rs.group_id = ag.group_id
JOIN
    sys.dm_hadr_availability_replica_cluster_states AS rcs ON rs.replica_id
= rcs.replica_id
ORDER BY
    rs.last_commit_time DESC;
```

By monitoring cluster and failover events, you ensure that your SQL Server environment remains highly available and can quickly respond to any issues that might lead to downtime.

Section 18: Storage Monitoring

1. Description

Monitoring storage is essential to ensure that there is enough disk space for your SQL Server databases to operate smoothly. This includes tracking the sizes and growth rates of data and log files, ensuring that they do not exceed available disk capacity.

2. How to Do

1. **Connect to SQL Server Management Studio (SSMS)**: Access the SQL Server instance.
2. **Execute Storage Monitoring Queries**: Use the provided SQL query to monitor disk usage.
3. **Review Results**: Regularly check the data to manage your disk space efficiently.

3. Code

Run this query to monitor the storage usage of your SQL Server database files:

```sql
SELECT
    DB_NAME(database_id) AS DatabaseName,
    name AS LogicalName,
    type_desc AS FileType,
    CAST(size AS BIGINT) * 8 / 1024 AS SizeMB,
    CASE
        WHEN max_size = -1 THEN 'Unlimited'
        ELSE CAST(CAST(max_size AS BIGINT) * 8 / 1024 AS VARCHAR(50))
    END AS MaxSizeMB,
    CAST(growth AS BIGINT) * 8 / 1024 AS GrowthMB,
    physical_name AS PhysicalFileName
FROM
    sys.master_files
WHERE
    type IN (0, 1) -- 0 = Data files, 1 = Log files
ORDER BY
    SizeMB DESC;
```

4. Explain Result of Code

- **DatabaseName**: Name of the database.
- **LogicalName**: The logical name of the file.
- **FileType**: Indicates whether it is a data file or a log file.
- **SizeMB**: The current size of the file in megabytes.
- **MaxSizeMB**: The maximum size the file can grow to, or 'Unlimited' if there is no limit.
- **GrowthMB**: The size by which the file will grow when more space is needed.
- **PhysicalFileName**: The full path to the physical file on disk.

SQLQuery1.sql - N...Nayeem Islam (65))*

```sql
SELECT
    DB_NAME(database_id) AS DatabaseName,
    name AS LogicalName,
    type_desc AS FileType,
    CAST(size AS BIGINT) * 8 / 1024 AS SizeMB,
    CASE
        WHEN max_size = -1 THEN 'Unlimited'
        ELSE CAST(CAST(max_size AS BIGINT) * 8 / 1024 AS VARCHAR(50))
    END AS MaxSizeMB,
    CAST(growth AS BIGINT) * 8 / 1024 AS GrowthMB,
    physical_name AS PhysicalFileName
FROM
    sys.master_files
WHERE
    type IN (0, 1) -- 0 = Data files, 1 = Log files
ORDER BY
    SizeMB DESC;
```

109 %

Results | Messages

	DatabaseName	LogicalName	FileType	SizeMB	MaxSizeMB	GrowthMB	PhysicalFileName
1	DBADashDB	DBADashDB_log	LOG	136	2097152	64	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...
2	DBADashDB	DBADashDB	ROWS	72	Unlimited	64	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...
3	msdb	MSDBData	ROWS	18	Unlimited	0	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...
4	tempdb	tempdev	ROWS	8	Unlimited	64	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...
5	tempdb	templog	LOG	8	Unlimited	64	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...
6	tempdb	temp2	ROWS	8	Unlimited	64	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...
7	tempdb	temp3	ROWS	8	Unlimited	64	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...
8	tempdb	temp4	ROWS	8	Unlimited	64	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...
9	tempdb	temp5	ROWS	8	Unlimited	64	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...
10	tempdb	temp6	ROWS	8	Unlimited	64	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...
11	tempdb	temp7	ROWS	8	Unlimited	64	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...
12	tempdb	temp8	ROWS	8	Unlimited	64	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...
13	model	modeldev	ROWS	8	Unlimited	64	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...
14	model	modellog	LOG	8	Unlimited	64	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...
15	msdb	MSDBLog	LOG	5	2097152	0	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...
16	master	master	ROWS	5	Unlimited	0	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...
17	master	mastlog	LOG	2	Unlimited	0	C:\Program Files\Microsoft SQL Server\MSSQL16.MSS...

This approach should now handle the data correctly without causing type conversion errors.

Section 19: Deadlock Monitoring

1. Description

Deadlock Monitoring in SQL Server is essential for detecting and resolving deadlocks, which occur when two or more processes block each other by holding locks on resources the others need. Deadlocks can cause significant performance issues, as affected transactions may be rolled back, leading to delays or loss of data consistency. By monitoring deadlocks, database administrators can identify and resolve these issues promptly, ensuring smooth and efficient database operations.

2. How to Do

To monitor deadlocks in SQL Server:

1. **Enable Deadlock Trace Flags**: Use trace flags to enable deadlock monitoring, such as `Trace Flag 1222`, which logs detailed deadlock information in the error log.
2. **Use Extended Events**: Set up an Extended Events session to capture deadlock events, providing insights into the queries and resources involved.
3. **Run Deadlock Monitoring Queries**: Use SQL queries to retrieve deadlock information from the system views and logs.
4. **Analyze the Results**: Regularly review the deadlock logs and captured events to identify and mitigate deadlock occurrences.

3. Code

Here's a SQL query that helps monitor deadlocks using Extended Events:

```
-- Create an Extended Events session to capture deadlock information
CREATE EVENT SESSION [DeadlockMonitor] ON SERVER
ADD EVENT sqlserver.xml_deadlock_report
ADD TARGET package0.event_file (SET filename =
'C:\DeadlockMonitoring\DeadlockReport.xel', max_file_size = 5)
WITH (MAX_MEMORY = 4096 KB, EVENT_RETENTION_MODE = ALLOW_SINGLE_EVENT_LOSS,
MAX_DISPATCH_LATENCY = 30 SECONDS, MAX_EVENT_SIZE = 0 KB,
MEMORY_PARTITION_MODE = NONE, TRACK_CAUSALITY = ON, STARTUP_STATE = OFF);
GO

-- Start the Extended Events session
ALTER EVENT SESSION [DeadlockMonitor] ON SERVER STATE = START;
GO
```

This query sets up an Extended Events session that captures deadlock reports and saves them to a file for later analysis.

4. Explain Result of Code

- **Extended Events Session**: The session named `DeadlockMonitor` is created to capture deadlock events using the `sqlserver.xml_deadlock_report` event, which logs detailed information about deadlocks in an XML format.
- **Event File Target**: The captured events are saved to a file (`DeadlockReport.xel`) located in `C:\DeadlockMonitoring\`, with each file having a maximum size of 5 MB.
- **Start the Session**: The session is then started, allowing SQL Server to begin capturing deadlock events immediately.

By monitoring deadlocks using this method, you can capture detailed information about deadlocks as they occur, allowing you to diagnose and resolve the underlying causes efficiently.

Section 20: TempDB Monitoring

1. Description

TempDB is a critical system database in SQL Server that is used to store temporary objects such as temporary tables, table variables, and intermediate results of query processing. Monitoring TempDB is essential for ensuring that it has sufficient space and is not becoming a performance bottleneck. TempDB is shared among all databases on the SQL Server instance, so issues with TempDB can affect the performance of the entire server.

2. How to Do

To effectively monitor TempDB in SQL Server:

1. **Monitor Space Usage**: Regularly check the space usage of TempDB to ensure that it is not running out of space.
2. **Track File Growth**: Monitor the growth of TempDB data and log files to prevent them from consuming too much disk space.
3. **Monitor Performance**: Track wait types and bottlenecks related to TempDB to identify and resolve performance issues.

3. Code

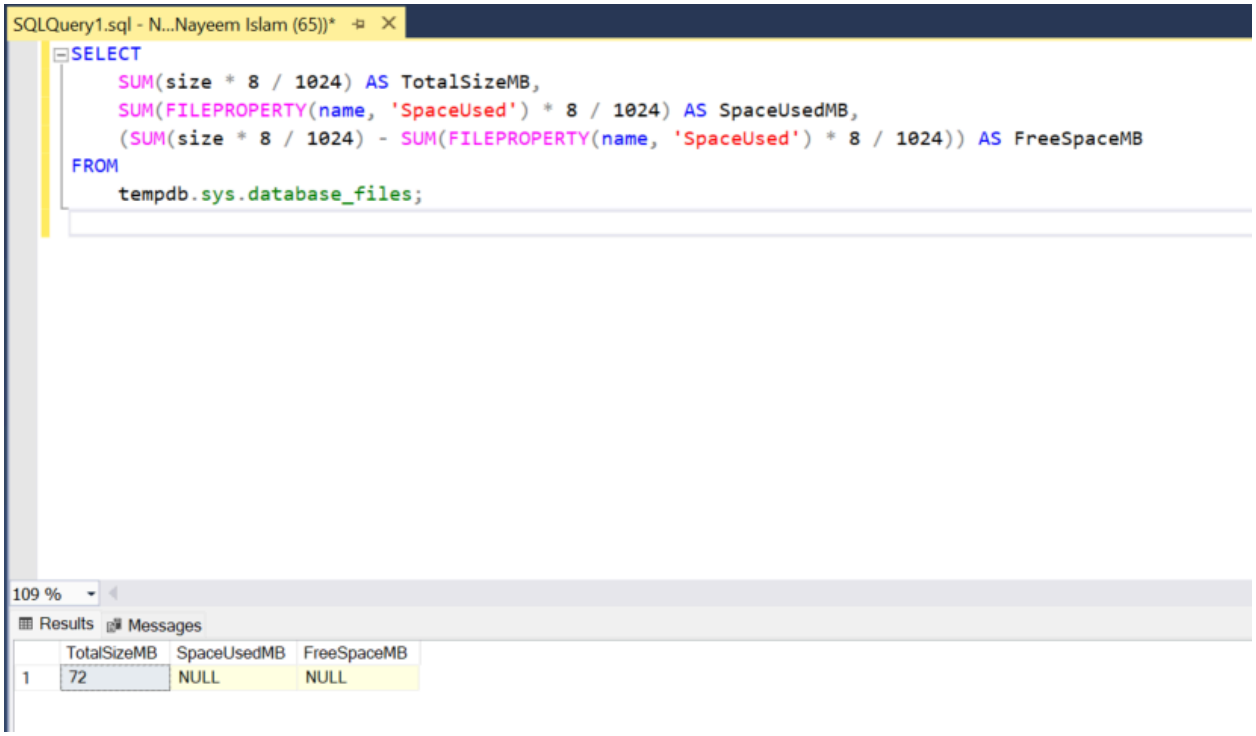
Here's a SQL query that helps monitor TempDB usage:

```
SELECT
    SUM(size * 8 / 1024) AS TotalSizeMB,
    SUM(FILEPROPERTY(name, 'SpaceUsed') * 8 / 1024) AS SpaceUsedMB,
    (SUM(size * 8 / 1024) - SUM(FILEPROPERTY(name, 'SpaceUsed') * 8 /
1024)) AS FreeSpaceMB
FROM
    tempdb.sys.database_files;
```

This query calculates the total size, used space, and free space of TempDB in megabytes.

4. Explain Result of Code

- **TotalSizeMB**: The total size of all TempDB files combined, in megabytes.
- **SpaceUsedMB**: The amount of space currently used by TempDB files, in megabytes.
- **FreeSpaceMB**: The amount of free space remaining in TempDB, in megabytes. This is calculated as the difference between the total size and the used space.



By regularly monitoring these metrics, you can ensure that TempDB has enough space to handle its workload and that it is not becoming a performance bottleneck.

Section 21: Security Monitoring

1. Description

Security Monitoring in SQL Server is essential for safeguarding your databases against unauthorized access, data breaches, and other security threats. It involves tracking login attempts, monitoring permission changes, detecting potential SQL injection attempts, and ensuring that the security configurations are in compliance with your organization's policies. By proactively monitoring security-related events, you can detect and respond to threats before they compromise your data.

2. How to Do

To effectively monitor security in SQL Server:

1. **Monitor Login Attempts**: Track successful and failed login attempts to identify potential unauthorized access.
2. **Track Permission Changes**: Monitor changes to user permissions to ensure that they are authorized and comply with security policies.
3. **Detect SQL Injection Attempts**: Set up alerts for queries that exhibit characteristics of SQL injection attacks.

4. **Review Security Logs**: Regularly review SQL Server logs for any security-related events or anomalies.

3. Code

Here's a SQL query that helps monitor failed login attempts:

```
SELECT
    login_name,
    COUNT(*) AS FailedLoginCount,
    MAX(login_time) AS LastFailedLoginTime
FROM
    sys.dm_exec_sessions
WHERE
    is_user_process = 1
    AND login_name IS NOT NULL
    AND status = 'failed'
GROUP BY
    login_name
ORDER BY
    FailedLoginCount DESC;
```

This query retrieves the number of failed login attempts per user, helping identify accounts that may be under attack.

4. Explain Result of Code

- **login_name**: The name of the user attempting to log in. This helps identify which accounts are being targeted for unauthorized access.
- **FailedLoginCount**: The total number of failed login attempts for each user. A high count may indicate a brute-force attack or a compromised account.
- **LastFailedLoginTime**: The timestamp of the last failed login attempt. This helps determine when the most recent unauthorized access attempt occurred.

By monitoring these security metrics, you can quickly detect and respond to potential security threats, ensuring that your SQL Server environment remains secure.

Conclusion

While the 20 methods of monitoring and maintenance presented in this guide offer a solid foundation for managing your SQL Server environment, it's important to recognize that real-life scenarios can vary significantly depending on factors such as data volume, usage patterns, user behavior, and the specific requirements of your organization. The strategies discussed are designed to help you get started with effective monitoring and maintenance, but they should be adapted and expanded based on the unique characteristics of your SQL Server setup.

In practice, you may encounter challenges that require more advanced techniques or customized solutions. Continuous learning and adaptation are key to maintaining a healthy and efficient SQL Server environment. Regularly reviewing and updating your monitoring and maintenance practices will help you address the evolving needs of your databases and ensure their ongoing performance and security.

By building on these foundational practices and tailoring them to your specific circumstances, you can create a robust framework for SQL Server management that meets the demands of your business and protects your critical data assets.