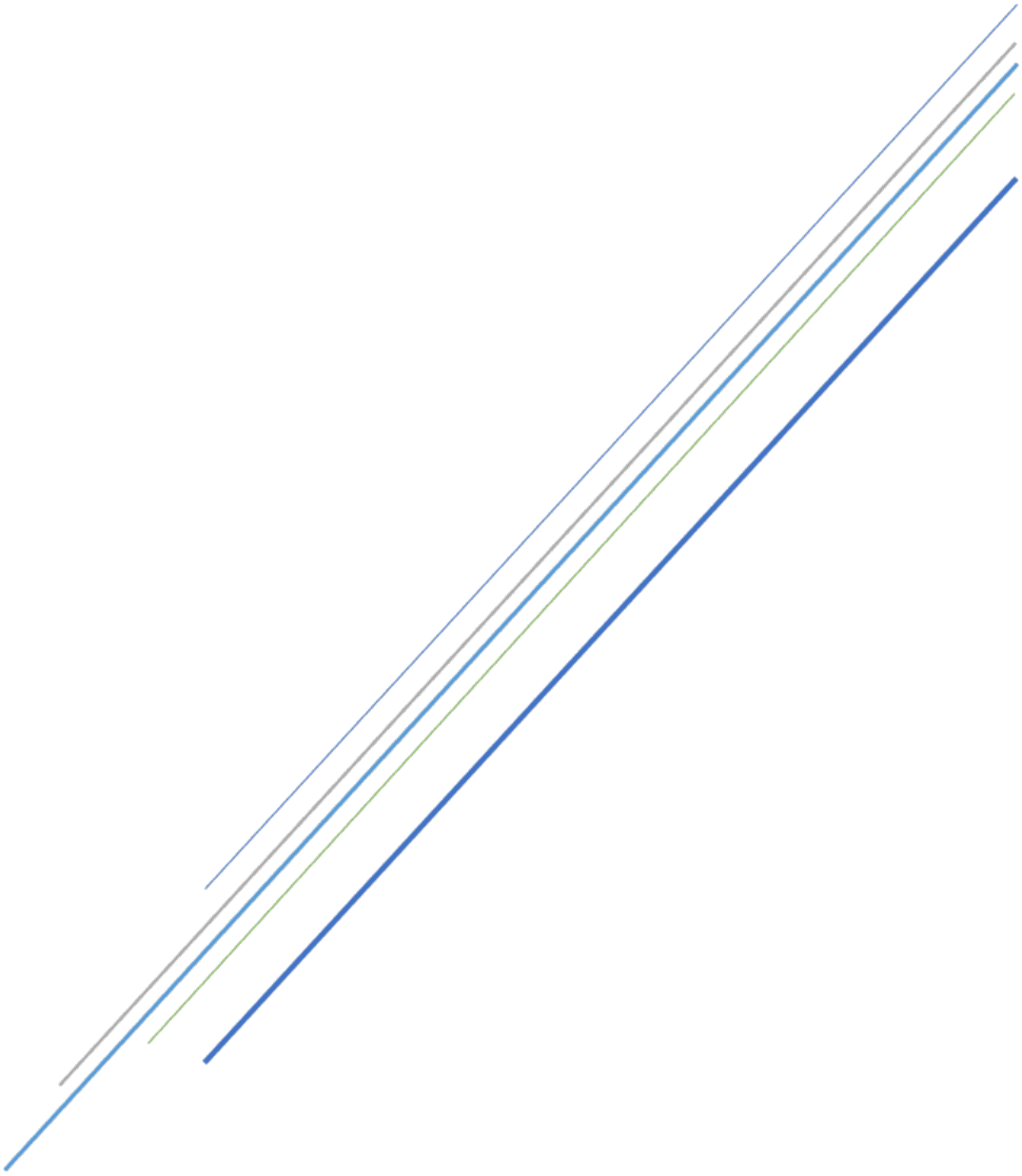
SQL2022

Server Documentation



Contact Information

AI-DBA Software Inc.
1500 West Georgia St. Vancouver, British Columbia V6G2Z6, Canada.
www.AI-DBA.net

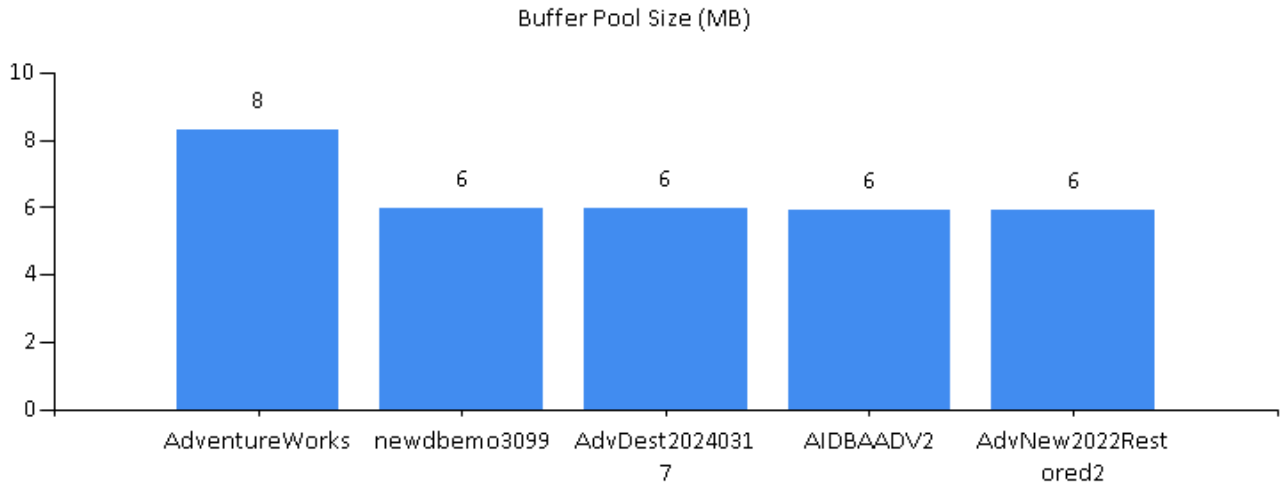
Table of Content

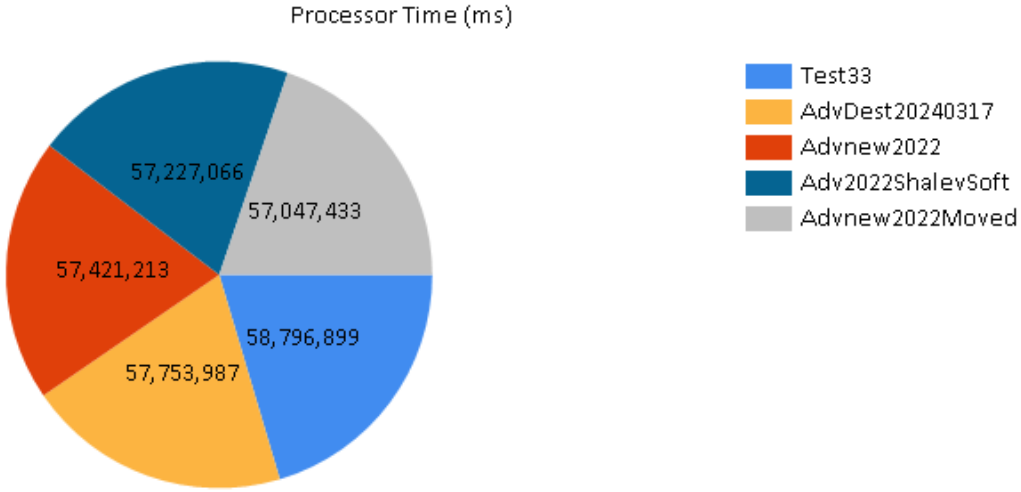
Action Required by Administrator

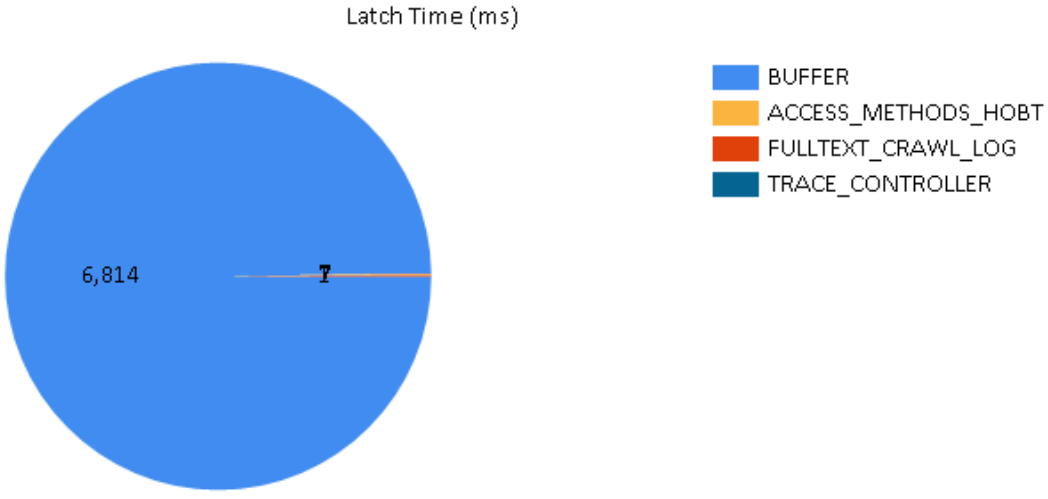
Database administrator is required to pay attention to the following recommendations.

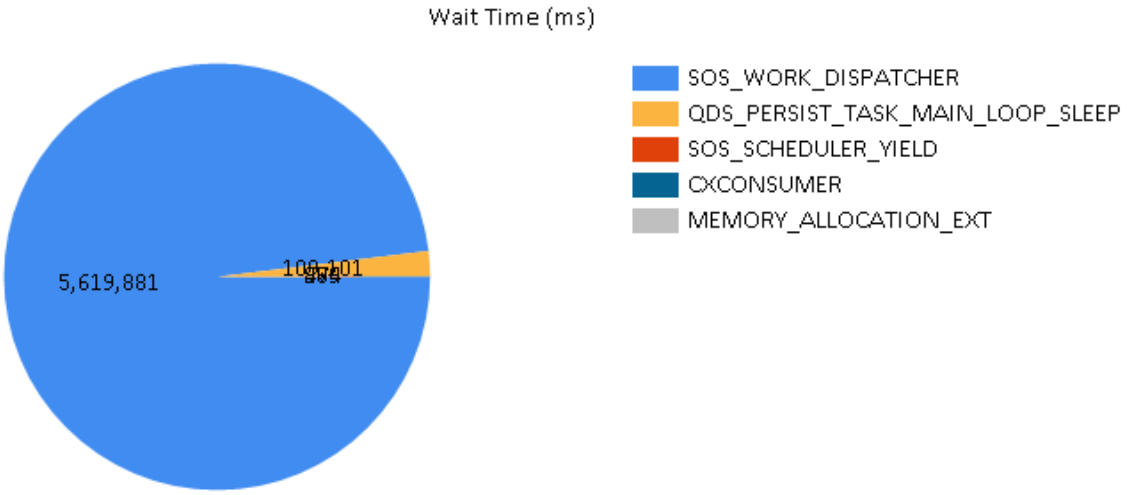
SQL Server Instance Health Overview

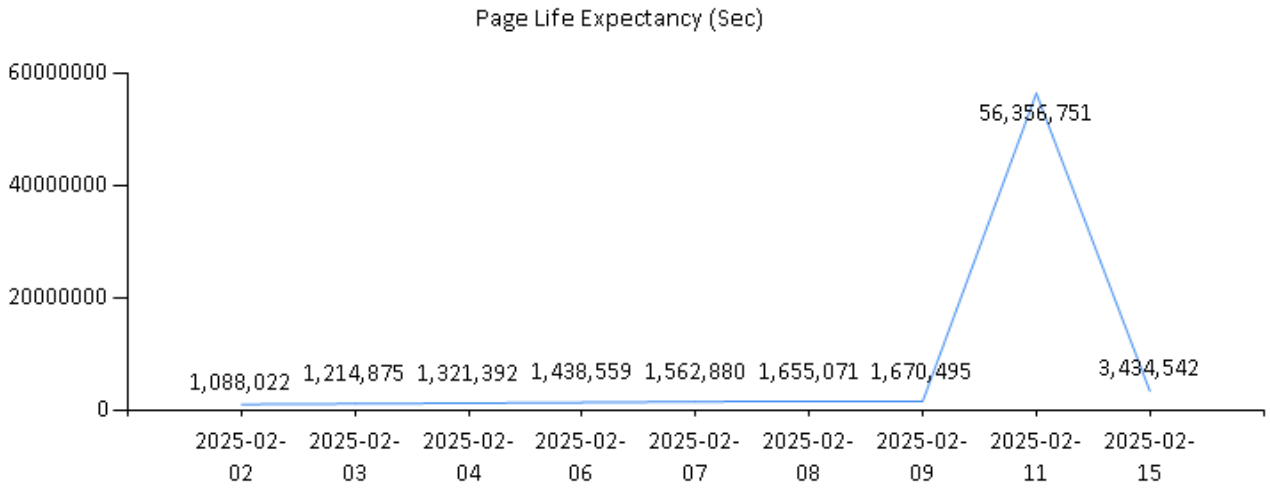
Provided to give you a glance of SQL Server instance health status.











Physical Memory (MB)	Plan Cache (MB)	Pressure %	Description
32,766.0	2,867.0	37.1	Plan Cache - Normal Situation

Processor Wait Signal	Resource Wait Signal	Description
4.3	95.7	Signal Waits later 15-17% is usually a sign of CPU pressure

Warnings and Predictions

Title: The current patch is outdated. It is required to patch to the latest cumulative update (CU) with , released on September 22,2022.

Category: Warning

ImpactLevel: High

Frequency: Medium

WhyIsItImportant?:
It is important to update the patch to the latest cumulative update (CU) because it provides necessary bug fixes, security enhancements, and performance improvements. Outdated patches may have vulnerabilities that can be exploited by hackers or impact system stability and reliability. Updating to the latest CU ensures that your system is up-to-date with the most recent advancements in SQL Server technology, ensuring optimal functionality and protection.

HowToResolveIt?:
To mitigate/resolve the issue of an outdated patch, it is recommended to install the latest cumulative update (CU) for SQL Server. The CU released on September 22, 2022 should be installed to ensure that all necessary bug fixes and security updates are in place. This will help address any known issues or vulnerabilities and keep the system up-to-date with the latest patches from Microsoft.

Type: n/a

Title: The database memory consumption has increased significantly since February 02,2025.

Category: Warning

ImpactLevel: High

Frequency: Medium

WhyIsItImportant?:
It is important to address the increased database memory consumption as it can have various negative impacts on system performance and stability. When a database consumes excessive memory, it can lead to slower query execution times, longer transaction durations, increased disk I/O operations, and potential system crashes or outages. By investigating and resolving the cause of this increase in memory consumption, administrators can ensure optimal performance and reliability of the SQL Server system.

HowToResolveIt?:
To mitigate the increased database memory consumption, you can start by analyzing and identifying any recent changes in the server environment or application that may have caused this increase. Next, review the database configuration settings such as maximum server memory and optimize them accordingly to ensure efficient memory usage. Furthermore, identify any long-running queries or processes and optimize their performance to minimize memory impact. Lastly, consider upgrading hardware resources such as increasing RAM capacity if necessary to accommodate the increased workload.

Type: n/a

Title: The [Adv2022ShalevSoft] database has 3 corrupted pages, possibly caused by IO noises or a faulty storage system. The most recent IO corruption was reported on November 26,2024 01:29:14.

Category: Warning

ImpactLevel: High

Frequency: Low

WhyIsItImportant?:
It is important to address the issue of corrupted pages in the [Adv2022ShalevSoft] database as it can significantly impact the integrity and reliability of the data. Corrupted pages can lead to data inconsistencies, errors, and even data loss if not resolved promptly. Identifying and resolving the source of corruption, such as IO noises or a faulty storage system, is crucial to ensuring the database's stability and maintaining data integrity.

HowToResolveIt?:
To mitigate/resolve the corruption issue in the [Adv2022ShalevSoft] database, you can follow these steps. First, identify and address any IO noises or faults in the storage system to prevent further corruption. Then, restore the affected pages from a previous backup taken before November 26, 2024 01:29:14 when the most recent corruption was reported. Verify the integrity of the restored pages and run a consistency check on the entire database to ensure there are no additional corruptions.

Type: n/a

Title: The [Advnew2022] database has 3 corrupted pages, possibly caused by IO noises or a faulty storage system. The most recent IO corruption was reported on November 26,2024 01:29:34.

Category: Warning

ImpactLevel: High

Frequency: Low

WhyIsItImportant?:
It is important to address the issue of corrupted pages in the [Advnew2022] database because they can lead to data loss and inconsistencies. Corrupted pages can result from various factors such as IO noises or a faulty storage system. By investigating and resolving this issue promptly, potential data corruption and subsequent problems can be avoided, ensuring the integrity and reliability of the database.

HowToResolveIt?:
To mitigate/resolve the issue of corrupted pages in the [Advnew2022] database, you can follow these steps:
1. Identify and fix any underlying issues with IO noises or a faulty storage system that may have caused the corruption.
2. Restore the affected pages from a valid backup if available, ensuring data integrity.
3. Run consistency checks like DBCC CHECKDB to identify and repair any additional corruption within the database.
4. Monitor the system closely to prevent further corruptions and consider implementing measures such as regular backups, redundant storage systems, and timely hardware maintenance to minimize future occurrences.

Type: n/a

Title: The [Advnew2022Moved] database has 3 corrupted pages, possibly caused by IO noises or a faulty storage system. The most recent IO corruption was reported on November 26,2024 01:29:39.

Category: Warning

ImpactLevel: High

Frequency: Low

WhyIsItImportant?:
It is important to address the corruption issues in the [Advnew2022Moved] database because corrupted pages can lead to data loss or inconsistency, which can affect various operations and functionality of the database. It is crucial to identify and fix the underlying cause of corruption, such as IO noises or a faulty storage system, to prevent any further damage and ensure the integrity and reliability of the database.

HowToResolveIt?:
To mitigate and resolve the issue of corrupted pages in the [Advnew2022Moved] database, you can take the following steps:

1. Identify the extent of corruption: Run a DBCC CHECKDB command to identify all corrupted pages in the database.
2. Restore from backups: If you have recent valid backups, restore them on a separate system or instance to ensure data integrity.
3. Fix corrupt pages: Attempt to repair or fix the corrupt pages using DBCC PAGE or other appropriate methods recommended by Microsoft support.
4. Address underlying issues: Investigate and address any potential causes for IO noises or faulty storage systems that could be leading to page corruptions.
5. Monitor for recurrence: Continuously monitor your system for any further occurrences of corrupted pages and take necessary actions promptly if such incidents happen again.

Note: It is highly recommended to involve experienced Database Administrators (DBAs) in this process as dealing with page corruption requires advanced knowledge of SQL Server internals and troubleshooting techniques.

Type: n/a

Title: The [Test33] database has 3 corrupted pages, possibly caused by IO noises or a faulty storage system. The most recent IO corruption was reported on November 26,2024 01:30:14.

Category: Warning

ImpactLevel: High

Frequency: Low

WhyIsItImportant?:
It is important to address the corrupted pages in the [Test33] database because they can lead to data loss and potential system crashes. Corrupted pages may result from various factors such as IO noises or a faulty storage system. By identifying and resolving these issues, it ensures data integrity and maintains a stable and reliable database environment.

HowToResolveIt?:
To mitigate and resolve the corruption issues in the [Test33] database, you can follow these steps:

1. Identify the root cause: Investigate whether there are any IO noises or issues with the storage system that could be causing corruption.
2. Fix IO noises or replace faulty hardware: If you determine that IO noises or a faulty storage system is

causing the corruption, take necessary actions to fix them. This might involve replacing malfunctioning hardware components or optimizing disk configurations.
3. Restore from backup: If you have a recent backup of the [Test33] database before the reported corruption date (November 26,2024 01:30:14), restore it to recover from corrupted pages.
4. Run DBCC CHECKDB command: After restoring from backup, run DBCC CHECKDB command to identify and repair any remaining errors within the database.
5. Monitor for further issues: Continuously monitor your server and databases for signs of recurring corruption after mitigation efforts, keeping an eye on IO performance metrics and reviewing error logs regularly.

Remember to always have proper backups in place to ensure data recovery in case of such incidents.

Type: n/a

Title: The size of the [Adv2022ShalevSoft] database files is misaligned, which can result in one or more files growing faster compared to others.

Category: Warning

ImpactLevel: Low

Frequency: Medium

WhyIsItImportant?:
It is important to address misalignment in database file sizes because it can lead to uneven growth among files. This imbalance can impact performance and hinder efficient resource utilization. By aligning the sizes of all database files, you can promote balanced growth and ensure optimal performance for your SQL Server database.

HowToResolveIt?:
To resolve the misalignment issue and ensure balanced growth of database files, you can follow these steps:

1. Identify the file(s) that are growing faster compared to others by monitoring the file size changes over time.
2. Use ALTER DATABASE command with MODIFY FILE option to resize and align the misaligned file(s) according to your desired growth pattern.
3. Implement a proper database maintenance plan, including regular file size monitoring and adjustment as needed, to prevent future misalignments.
4. Consider implementing autogrowth settings appropriately on all files in order to avoid sudden unexpected growth issues.

Make sure to always have sufficient disk space available for accommodating planned increases in file sizes.

Type: n/a

Title: The size of the [AdvDest20240317] database files is misaligned, which can result in one or more files growing faster compared to others.

Category: Warning

ImpactLevel: Low

Frequency: Medium

WhyIsItImportant?:

It is important to address misalignment in database files because it can lead to uneven growth rates among the files. This can result in performance issues and disk space inefficiencies, as some files may reach their maximum capacity while others have plenty of unused space. By ensuring proper alignment, database administrators can optimize storage utilization and prevent bottlenecks in data access and retrieval.

HowToResolveIt?:
To mitigate or resolve the misalignment issue with the database files, you can follow these steps. First, analyze the growth patterns of each file in the database to identify which files are growing faster than others. Next, reallocate the file sizes by adjusting their initial size and enabling autogrowth settings appropriately based on their respective growth patterns. This will ensure that all files grow proportionally and prevent any one file from becoming significantly larger than others. Finally, monitor and regularly optimize the database to maintain a balanced distribution of data across all files.

Type: n/a

Title: The size of the [AdventureWorks] database files is misaligned, which can result in one or more files growing faster compared to others.

Category: Warning

ImpactLevel: Low

Frequency: Medium

WhyIsItImportant?:
It is important to address the misalignment of database file sizes in order to maintain optimal performance and manage the usage of storage resources effectively. When files within a database grow at different rates, it can cause imbalances in disk space utilization and potentially lead to issues such as uneven data distribution, slower query execution times, and inefficient use of storage capacity. Regular monitoring and alignment of database file sizes are necessary for maintaining a healthy and well-performing system.

HowToResolveIt?:
To mitigate/resolve the misalignment issue with the [AdventureWorks] database files, you can perform a manual realignment of the files. This involves monitoring each file's size and adjusting it as needed to maintain proportional growth among all files. Alternatively, you can automate the process using SQL Server Management Studio or Transact-SQL scripts to resize and align the files based on your desired settings. Regularly monitoring and adjusting file sizes will help ensure balanced growth across all database files within [AdventureWorks].

Type: n/a

Title: The size of the [Advnew2022] database files is misaligned, which can result in one or more files growing faster compared to others.

Category: Warning

ImpactLevel: Low

Frequency: Medium

WhyIsItImportant?:
Ensuring proper alignment of database files is important for several reasons. Misalignment can lead to uneven growth patterns, where some files grow faster than others. This can result in imbalanced disk space utilization and inefficient performance as the workload is not distributed evenly across all files. Properly

aligned database files promote uniform growth and optimize storage utilization, enhancing overall system performance and manageability. It also helps prevent potential issues like disk fragmentation and improves I/O operations by reducing contention on specific data or log file(s).

HowToResolveIt?:
To mitigate the issue of misaligned database file sizes in [Advnew2022], you can regularly monitor and manage the growth of individual files within the database. Use SQL Server Management Studio or a similar tool to analyze file size distribution and adjust accordingly. Consider enabling auto-growth settings for files that are growing faster to ensure a balanced size distribution across all database files. Additionally, perform regular maintenance tasks such as index rebuilds and data archiving to optimize space usage within the database.

Type: n/a

Title: The size of the [Advnew2022_20240312102058] database files is misaligned, which can result in one or more files growing faster compared to others.

Category: Warning

ImpactLevel: Low

Frequency: Medium

WhyIsItImportant?:
It is important to address misalignment in database files because it can lead to uneven growth, causing certain files to expand more rapidly than others. This can result in imbalanced distribution of data and overall performance degradation. By resolving the misalignment, administrators can ensure optimal file growth and maintain consistent performance across the database.

HowToResolveIt?:
To mitigate/resolve the issue of misaligned database file sizes, you can perform the following steps:
1. Monitor the growth rate of each file in the database to identify any disparities.
2. Determine the ideal size for each file based on their respective usage patterns and expected growth.
3. Reallocate space within the files by using ALTER DATABASE or SHRINKFILE commands to align them with their optimal sizes.
4. Regularly monitor and adjust file sizes as needed to ensure balanced growth across all database files.

Type: n/a

Title: The size of the [Advnew2022Moved] database files is misaligned, which can result in one or more files growing faster compared to others.

Category: Warning

ImpactLevel: Low

Frequency: Medium

WhyIsItImportant?:
It is important to address the misalignment in file sizes of the database files because if one or more files grow faster than others, it can lead to imbalanced storage utilization and affect overall performance of the database. This can result in slower query execution times, increased disk space usage, and potential bottlenecks for data retrieval or modification operations. Resolving this issue ensures optimal usage of storage resources and maintains a stable and efficient database environment.

HowToResolveIt?:
To mitigate the issue of misaligned file size in the [Advnew2022Moved] database, you can perform the following steps:

1. Identify the files that are growing faster compared to others by monitoring their size changes over time.
2. Resize and reallocate file space according to data growth patterns. This can be done by either shrinking or expanding the respective files.
3. Regularly monitor and optimize file growth strategies to ensure balanced growth across all files within the database.

By aligning file sizes properly and ensuring balanced growth, you can mitigate any potential issues caused by misaligned database file sizes.

Type: n/a

Title: The size of the [AdvNew2022Restored] database files is misaligned, which can result in one or more files growing faster compared to others.

Category: Warning

ImpactLevel: Low

Frequency: Medium

WhyIsItImportant?:
It is important to address the misalignment of database files because it can lead to imbalanced growth rates among the files. Uneven growth can result in uneven disk space usage, causing performance issues and potentially leading to storage capacity problems. By aligning the file sizes, you ensure a more balanced distribution of data across the database files, promoting optimal performance and efficient use of resources.

HowToResolveIt?:
To mitigate the misalignment of database files, you can perform the following steps:

1. Monitor and analyze the growth patterns of individual files within the database.
2. Regularly resize and reallocate file sizes to ensure they are balanced and aligned.
3. Use best practices for file placement on disk arrays to optimize performance.
4. Consider using Instant File Initialization, which allows faster growth for data files during allocation transactions.
5. Implement a regular maintenance plan to monitor and manage file growth proactively.

By regularly monitoring file growth, redistributing sizes appropriately, and implementing optimization techniques, you can minimize misalignment issues in your database files.

Type: n/a

Title: The size of the [AdvNew2022Restored2] database files is misaligned, which can result in one or more files growing faster compared to others.

Category: Warning

ImpactLevel: Low

Frequency: Medium

WhyIsItImportant?:
Having misaligned database file sizes can lead to imbalances in disk space usage, which can cause performance issues and inefficient storage utilization. When certain files grow faster than others, it results in uneven distribution of data across the storage system. This not only affects the query performance but also makes it difficult to effectively manage and maintain the database. It is important to ensure proper alignment of database file sizes for optimal performance, efficient resource utilization, and easier management of the SQL Server environment.

HowToResolveIt?:
To mitigate or resolve this issue, you can perform a manual realignment of the database files using the ALTER DATABASE command. This involves monitoring the file growth over time and redistributing the data across multiple files to ensure more balanced growth rates. Additionally, regularly analyzing the database usage patterns and adjusting file sizes accordingly can help prevent future misalignments.

Type: n/a

Title: The size of the [AdvNew2022Restored3] database files is misaligned, which can result in one or more files growing faster compared to others.

Category: Warning

ImpactLevel: Low

Frequency: Medium

WhyIsItImportant?:
It is important to address the misalignment of database files in order to ensure balanced growth and optimal performance. When the files are misaligned, it can lead to uneven distribution of data across them, causing some files to grow faster than others. This can result in potential issues such as slower queries, increased disk space usage, and difficulties in managing and maintaining the database. By aligning the database files properly, we can prevent these problems and maintain a healthy and efficient database environment.

HowToResolveIt?:
To mitigate or resolve the misalignment in the size of the [AdvNew2022Restored3] database files, you should monitor and adjust the file growth settings for each file accordingly. This can be done by regularly reviewing the database growth patterns and adjusting the file sizes accordingly to ensure all files grow at a similar rate. Additionally, consider implementing an automated monitoring solution that alerts you when file sizes are becoming misaligned, allowing you to quickly intervene and address any imbalances.

Type: n/a

Title: The size of the [AdvNewDB2022Portal] database files is misaligned, which can result in one or more files growing faster compared to others.

Category: Warning

ImpactLevel: Low

Frequency: Medium

WhyIsItImportant?:
It is important to address the misalignment of database files in order to maintain optimal performance and efficient storage utilization. When file growth occurs unevenly, it can lead to imbalanced disk usage, potentially causing performance issues such as slower query execution times and increased I/O contention.

By addressing file misalignment, you ensure that all files grow at a balanced rate, enabling improved performance and better resource management within the database environment.

HowToResolveIt?:
To mitigate or resolve the issue of misaligned database file sizes in [AdvNewDB2022Portal], you can follow these steps:
1. Analyze the growth patterns and data distribution of the database to determine which files are growing faster.
2. Use SQL Server Management Studio or a script to reallocate space evenly across all files by shrinking and expanding them as necessary.
3. Implement a regular monitoring routine to track file size growth and adjust accordingly, ensuring that no single file becomes significantly larger than others over time. This will help maintain balanced file sizes in the database.

Type: n/a

Title: The size of the [AIDBAADV2] database files is misaligned, which can result in one or more files growing faster compared to others.

Category: Warning

ImpactLevel: Low

Frequency: Medium

WhyIsItImportant?:
It is important to ensure that the size of database files is properly aligned because misalignment can lead to uneven growth in file sizes. This can cause performance issues, as growing files at different rates may result in imbalanced disk I/O and slow down query execution. Properly aligning database files helps optimize storage usage and ensures consistent performance across all files.

HowToResolveIt?:
To mitigate the misalignment issue in the [AIDBAADV2] database, it is recommended to regularly monitor and correctly align the size of each database file. This can be done by resizing or reallocating space among files through SQL Server Management Studio or T-SQL commands. Additionally, monitoring disk space usage and performing regular maintenance checks will help identify any further issues related to file growth imbalance.

Type: n/a

Title: The size of the [newdbemo3099] database files is misaligned, which can result in one or more files growing faster compared to others.

Category: Warning

ImpactLevel: Low

Frequency: Medium

WhyIsItImportant?:
It is important to align the size of database files because misalignment can lead to uneven growth rates among the files. This imbalance can cause performance issues and hinder efficient disk space utilization. By ensuring alignment, resources are distributed more evenly and database operations are optimized for better overall system performance.

HowToResolveIt?:
To mitigate this issue, you can implement a regular monitoring and maintenance plan for the database files. This should include regularly checking the size of each file in the database and ensuring they are aligned properly. If any misalignment is detected, manually resize or move the files to balance their growth evenly. Additionally, consider enabling auto-growth settings with appropriate increment values to prevent imbalance in file growth.

Type: n/a

Title: The size of the [ReportDB_Copy] database files is misaligned, which can result in one or more files growing faster compared to others.

Category: Warning

ImpactLevel: Low

Frequency: Medium

WhyIsItImportant?:
It is important to address the misalignment in database file sizes because uneven growth can lead to performance issues and inefficient use of storage. When one or more files grow faster than others, it can cause an imbalance in data distribution and space utilization within the database. This could result in slower query performance, increased fragmentation, and potential bottlenecks during backup and restore operations. By aligning file sizes properly, you ensure optimal disk usage, improve overall database performance, and mitigate potential storage-related problems.

HowToResolveIt?:
To mitigate or resolve the issue of misaligned database file sizes in [ReportDB_Copy], you can perform the following steps:

1. Analyze the growth patterns of each file in the database to identify any imbalances.
2. Adjust the autogrowth settings for each file accordingly, ensuring that they are aligned with your expected growth rates.
3. Implement regular monitoring and capacity planning to proactively identify any potential imbalances and take corrective actions as needed.
4. Regularly monitor and manage free space within each file by performing appropriate maintenance tasks such as shrinking or reorganizing data where necessary.
5. Consider redistributing data across files if required to achieve a more balanced distribution of storage utilization among all files.

By taking these measures, you can ensure that all files in [ReportDB_Copy] have equal opportunities for growth and prevent one file from growing excessively faster than others, resulting in a more optimized overall performance of the database.

Type: n/a

Title: The size of the [Test33] database files is misaligned, which can result in one or more files growing faster compared to others.

Category: Warning

ImpactLevel: Low

Frequency: Medium

WhyIsItImportant?:
It is important to ensure that the size of database files, such as [Test33], are aligned properly to avoid imbalanced growth. Misaligned file sizes can lead to uneven distribution of data across different files, causing performance issues. Maintaining balanced file growth helps optimize disk space utilization and improves overall database performance and management.

HowToResolveIt?:
To mitigate the issue of misaligned database file sizes, you can take the following steps. First, identify the files that are growing faster than others by monitoring their size regularly. Then, realign these files by manually resizing them to be in line with other files in the same database. Additionally, set up a proactive maintenance plan to monitor and manage file growth on an ongoing basis. This could involve implementing policies for regular file resizing or utilizing automatic file growth settings based on anticipated usage patterns.

Type: n/a

Title: The expansion of the TempDB database files takes 7 seconds, leading to slow workload performance and excessive blocking on the TempDB resource.

Category: Warning

ImpactLevel: Medium

Frequency: Medium

WhyIsItImportant?:
It is important to address the slow expansion of TempDB database files because it affects the overall workload performance and can lead to excessive blocking on the TempDB resource. This can result in decreased efficiency and productivity for users accessing the database. By optimizing TempDB file growth, administrators ensure a smooth and uninterrupted workflow, improving overall system performance.

HowToResolveIt?:
To mitigate the slow expansion of TempDB database files and reduce excessive blocking, you can pre-size the TempDB files to an appropriate size based on your workload requirements. This can be done by monitoring the growth patterns of TempDB over a period of time and adjusting the initial file sizes accordingly. Additionally, consider placing the TempDB files on separate disk drives or storage devices to improve I/O performance. Regularly monitoring and maintaining adequate free space in TempDB also helps prevent its excessive growth and related issues.

Type: n/a

Title: The database workload is experiencing numerous Missing Column Statistics, with the majority of them originating from AI-DBA Data Gateway applications. Query optimization is required to improve performance.

Category: Warning

ImpactLevel: Medium

Frequency: High

WhyIsItImportant?:
It is important to address missing column statistics in the database workload because it can significantly impact query performance. When the optimizer lacks accurate statistical information about columns, it may

make suboptimal decisions when generating query plans. This can result in slower execution times, decreased scalability, and overall degradation of system performance. By optimizing queries and ensuring that column statistics are up to date, administrators can improve overall database performance and enhance user experience with the application.

HowToResolveIt?:
To mitigate the issue of Missing Column Statistics and improve the performance of AI-DBA Data Gateway applications, you can implement query optimization techniques. This includes analyzing and identifying queries that are causing the missing statistics, updating or creating new column statistics using tools like SQL Server's automatic statistics update feature or manually running UPDATE STATISTICS commands, and reevaluating the database schema to ensure proper indexing is in place. Regular monitoring and maintenance activities should also be carried out to prevent future occurrences of missing column statistics.

Type: n/a

Title: The database engine is currently configured with dynamic port enabled. However, it is strongly recommended to convert the dynamic port to a static port to ensure consistent and reliable communication.

Category: Warning

ImpactLevel: High

Frequency: High

WhyIsItImportant?:
It is important to convert the dynamic port to a static port in the database engine because it ensures consistent and reliable communication. With a static port, there is no change in the port number, which means that applications and clients can always connect to the database using the same known port. This eliminates any potential issues caused by ports being dynamically assigned or changing unexpectedly, ensuring smooth and uninterrupted communication with the database server.

HowToResolveIt?:
To mitigate this issue, you should configure the SQL Server database engine to use a static port instead of dynamic port. This will ensure consistent and reliable communication by always listening on the same port number. You can set a specific port number in the SQL Server Configuration Manager under TCP/IP properties for your instance. Once configured, remember to update any firewall settings to allow traffic on the new static port.

Type: n/a

Title: The SQL Server Browser service is currently active, which poses a significant risk of exposing the SQL Server instance through port query. Hackers can potentially obtain connectivity information by exploiting the UDP/TCP 1434 port via the SQL Browser service.

Category: Warning

ImpactLevel: High

Frequency: High

WhyIsItImportant?:
It is important to address the active SQL Server Browser service because it poses a security risk. By exploiting the UDP/TCP 1434 port, hackers can potentially gather connectivity information about the SQL Server instance. This could lead to unauthorized access or other malicious activities on the server. Therefore,

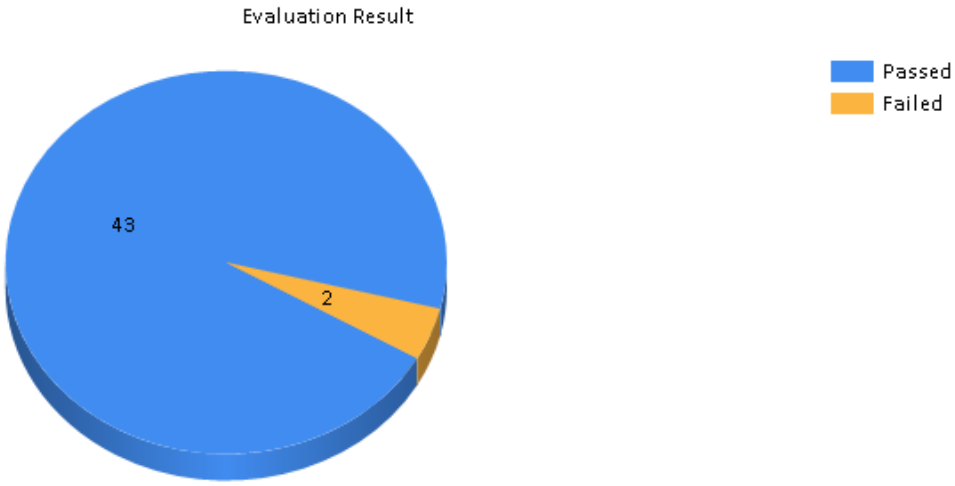
addressing this issue helps to protect and secure the SQL Server environment from potential threats.

HowToResolveIt?:
To mitigate this risk, it is recommended to disable the SQL Server Browser service unless necessary. By disabling this service, hackers will not be able to exploit UDP/TCP port 1434 to obtain connectivity information. However, before disabling the service, ensure that any applications or services dependent on it are reconfigured to directly connect using static ports instead of relying on dynamic port allocation provided by the SQL Server Browser service.

Type: n/a

SQL Server Instance Evaluation and Scores

Administration	Maintenace	Performance	Security	High Availability
4.6 / 5	5.0 / 5	5.0 / 5	5.0 / 5	5.0 / 5



Category	Title	Result
Administration	Database Full Recovery Model	Passed
Administration	Database File Federation	Failed
Administration	Database Table Partitioning for Tables >2mil Records	Passed
Administration	Database File IO Balancing	Failed
Administration	Database Full Backup in Past 7Days	Passed
Administration	Database Log Backup in Past 12Hours	Passed
Administration	Agent Job Failures for <10 Times	Passed
Administration	Database Auto Growth For <10 Times	Passed
Maintenance	Database Compression for Tables with >1mil Records	Passed
Maintenance	Database with No Suspected/Corrupted Pages	Passed
Maintenance	SQL Server with No Dump File.	Passed
Maintenance	Database Consistency Check in Past 14Days	Passed
Maintenance	Database Log Shipping Error Free in Past 7Days	Passed
Maintenance	Database Mirroring Error and Delay Free in Past 7Days	Passed
Maintenance	Moderated Database Transaction Log Fragmentation	Passed
Maintenance	Upgraded to the Least Support Database Engine Version	Passed
Maintenance	Database Engine with No Severe (Severity >16) Error	Passed
Maintenance	Operating System with No Severe Error	Passed
Security	Database Objects are Owned by non-DBO	Passed
Security	Database with No Orphaned Users	Passed

Category	Title	Result
Security	Login Authentication with <5 Failures	Passed
Security	>25 Active Sessions with SysAdmin Privileges	Passed
Security	SQL Server services with NT Service Account	Passed
Security	Login Brutal Attack Protection	Passed
Security	Database Engine TCP Listening Port Protection	Passed
Performance	<15ms IO Stall for Database Files	Passed
Performance	Query Design with Minimum Processor Intensive Functions	Passed
Performance	Trivial and Moderated Execution Plans	Passed
Performance	In-Memory with Utilized Hash Buckets	Passed
Performance	Database Index Design with Best Practices	Passed
Performance	Database Indexes with Low Fragmentation	Passed
Performance	Queries with an Index Support (No Missing Indexes)	Passed
Performance	Queries without an Index Influence (With Index Clause)	Passed
Performance	Database Task Request with Zero Pending Disk IO	Passed
Performance	Trivial Plans with Low Generation Cost	Passed
Performance	Low Database Deadlock and Blocking Occurrence in Past 7Days	Passed
Performance	Low Internal Pressure in Plan Cache Buckets	Passed
Performance	Low External Pressure in Plan Cache Buckets	Passed
Performance	Low Session Blocking in the Past 7Days	Passed
Performance	Low Pressure on Processor Resource	Passed
Performance	Low Pressure on System Memory Resource	Passed
Performance	Low Wait on Disk, Memory, Processor and Network Resources	Passed
High Availability	All Cluster Nodes are Up and Running	Passed
High Availability	The AlwaysOn Availability Group Data Loss and Recovery is Less Than 120Seconds	Passed
High Availability	The Database Mirroring Session is Up and Running	Passed

Hardware Specification

Property Item	Value
Physical Processors	1
Logical Processors	8
Hyperthread Ratio	8
Physical Memory (MB)	32,766
Committed Memory (MB)	24,811
Machine Type	Virtual

Volume	Capacity (MB)	Available Space (MB)	Free Space %	Bus Type	Media Type	IO Stall (MS)
C:\	129,481.0	61,115.0	47.2	SAS	Unspecified	1.6
G:\	65,405.0	58,145.0	88.9	SAS	Unspecified	1.6
H:\	65,405.0	59,911.0	91.6	SAS	Unspecified	0.6

Server Insight

The server insight content is based on the collected telemetry data and analyzed by AI engine.

* Last Update: 2025-02-15 11:55:43 UTC
The provided data shows the storage space for different volumes in a SQL Server instance and the free space percentage at each timestamp.

For the C:\ volume, the free space percentage is consistently 0.0%, indicating that there is no available storage space.

On the other hand, for the H:\ volume, there is a consistent 91.6% of free space.

Similarly, for G:\ volume, there is a consistent 88.9% of free space.

Based on this information, we can conclude that:

- The C:\ volume has reached its maximum capacity and requires additional storage allocation.
- The H:\ and G:\ volumes still have significant available storage capacity.

To determine future storage needs and growth rates accurately, more historical data would be required over time to analyze trends in data growth and predict future requirements.

SQL Server Environment

Property Item	Value
Machine Name	Ai-DBA-DEMO
Instance	SQL2022
Installation Type	Standalone
Product Version	16.0.4125.3
Edition	Developer Edition (64-bit)
Patch Level	RTM
Collation	SQL_Latin1_General_CP1_CI_AS
Authentication Mode	Windows and SQL Server
Is Full-Text Installed	Yes
HADR Manager Service Status	Nil
System Memory State	Available physical memory is high

IP Address	IP Type	Port	For	Status
0.0.0.0	IPv4	51887	TSQL	ONLINE
127.0.0.1	IPv4	49810	TSQL	ONLINE
::	IPv6	51887	TSQL	ONLINE
::1	IPv6	49810	TSQL	ONLINE

SQL Server Installed Services

Display Name	Start Mode	Service Account	State	Status
SQL Server VSS Writer	Auto	LocalSystem	Running	OK
SQL Server CEIP service (SQL2022)	Auto	NT Service\SQLTELEMETRY$SQL2022	Running	OK
SQL Server CEIP service (SQL2019)	Auto	NT Service\SQLTELEMETRY$SQL2019	Running	OK
SQL Server CEIP service (SQL2017)	Auto	NT Service\SQLTELEMETRY$SQL2017	Running	OK
SQL Server CEIP service (SQL2016)	Auto	NT Service\SQLTELEMETRY$SQL2016	Running	OK
SQL Server Browser	Auto	NT AUTHORITY\LOCALSERVICE	Running	OK
SQL Server Agent (SQL2022)	Manual	NT Service\SQLAgent$SQL2022	Stopped	OK
SQL Server Agent (SQL2019)	Manual	NT Service\SQLAgent$SQL2019	Stopped	OK
SQL Server Agent (SQL2017)	Manual	NT Service\SQLAgent$SQL2017	Stopped	OK
SQL Server Agent (SQL2016)	Manual	NT Service\SQLAgent$SQL2016	Stopped	OK
SQL Server (SQL2022)	Auto	NT Service\MSSQL$SQL2022	Running	OK
SQL Server (SQL2019)	Manual	NT Service\MSSQL$SQL2019	Stopped	OK
SQL Server (SQL2017)	Manual	NT Service\MSSQL$SQL2017	Stopped	OK
SQL Server (SQL2016)	Manual	NT Service\MSSQL$SQL2016	Stopped	OK
SQL Full-text Filter Daemon Launcher (SQL2022)	Manual	NT Service\MSSQLFDLauncher$SQL2022	Running	OK
SQL Full-text Filter Daemon Launcher (SQL2017)	Manual	NT Service\MSSQLFDLauncher$SQL2017	Stopped	OK
SQL Full-text Filter Daemon Launcher (SQL2016)	Manual	NT Service\MSSQLFDLauncher$SQL2016	Stopped	OK

SQL Server Instance Configuration

Configuration	Value
access check cache bucket count	0
access check cache quota	0
Ad Hoc Distributed Queries	Disable
ADR cleaner retry timeout (min)	15
ADR Cleaner Thread Count	1
ADR Preallocation Factor	4
affinity I/O mask	Automatic
affinity mask	Automatic
affinity64 I/O mask	Automatic
affinity64 mask	Automatic
Agent XPs	Enable
allow filesystem enumeration	1
allow polybase export	0
allow updates	0
automatic soft-NUMA disabled	Disable
backup checksum default	Disable
backup compression algorithm	0
backup compression default	Enable
blocked process threshold (s)	0
c2 audit mode	0
clr enabled	Disable
clr strict security	Enable
column encryption enclave type	0
common criteria compliance enabled	Disable
contained database authentication	Disable
cost threshold for parallelism	5
cross db ownership chaining	Disable
cursor threshold	-1
Data processed daily limit in TB	2147483647
Data processed monthly limit in TB	2147483647
Data processed weekly limit in TB	2147483647
Database Mail XPs	Disable
default full-text language	1033
default language	0
default trace enabled	Enable
disallow results from triggers	Disable
EKM provider enabled	Disable
external scripts enabled	Disable
filestream access level	0
fill factor (%)	70
ft crawl bandwidth (max)	100

Configuration	Value
ft crawl bandwidth (min)	0
ft notify bandwidth (max)	100
ft notify bandwidth (min)	0
hadoop connectivity	Disable
hardware offload config	0
hardware offload enabled	Disable
hardware offload mode	0
index create memory (KB)	0
in-doubt xact resolution	0
lightweight pooling	Disable
locks	0
max degree of parallelism	2
max full-text crawl range	4
max RPC request params (KB)	0
max server memory (MB)	24811
max text repl size (B)	65536
max worker threads	0
media retention	0
min memory per query (KB)	1024
min server memory (MB)	3101
nested triggers	Enable
network packet size (B)	4096
Ole Automation Procedures	Disable
open objects	Disable
openrowset auto_create_statistics	Enable
optimize for ad hoc workloads	Enable
PH timeout (s)	60
polybase enabled	0
polybase network encryption	1
precompute rank	0
priority boost	0
query governor cost limit	0
query wait (s)	-1
recovery interval (min)	0
remote access	Enable
remote admin connections	Disable
remote data archive	Disable
remote login timeout (s)	10
remote proc trans	0
remote query timeout (s)	600
Replication XPs	Disable
scan for startup procs	Disable
server trigger recursion	1

Configuration	Value
set working set size	0
show advanced options	Enable
SMO and DMO XPs	Enable
suppress recovery model errors	0
tempdb metadata memory-optimized	Disable
transform noise words	Disable
two digit year cutoff	2049
user connections	0
user options	0
version high part of SQL Server	1048576
version low part of SQL Server	270336003
xp_cmdshell	Enable

Instance Maximum Consumption Rate (MCR)

The core-balanced system architecture is based on the fact that most OLAP or OLTP workloads need to transfer small to large amounts of data (usually accessed by sequential or random read operations) across multiple system components, from where the data is stored to the requesting applications. Each component through which the data is transferred is a potential bottleneck that will limit the overall performance of the system. The data can only flow to the requesting application at the rate of the slowest component. Any components that can operate at a higher rate are underutilized, which unbalances the system and can represent significant wasted cost.
The core-balanced approach starts with the throughput of the CPU core, and then builds a balanced system that is based on that metric. It is important to realize that Maximum Consumption Rate (MCR) is purely a measure of SQL Server data throughput for a single core and does not include disk read operations or network I/O.

MCR Formula: (Average Logical Reads / Average CPU Time (Sec)) * 8 / 1024
Cores Formula: ((Average Query Result Size (MB) / MCR) * Concurrent Users) / Target Time (Sec)

Computed Date/Time	Con. Users	Est. MCR	Est. Query Cache (MB)	Est. Query Elapse Time (Sec)	Logical Processor Rate
2/16/2025 12:39:27 AM	72	1	4,664	1,984	3
2/15/2025 10:20:09 PM	47	1	4,664	1,984	2
2/15/2025 3:26:29 PM	66	1	4,663	1,452	4
2/15/2025 11:58:16 AM	67	1	3,097	1,104	3
2/15/2025 10:22:42 AM	74	1	4,663	1,491	5
2/15/2025 8:53:18 AM	64	1	4,663	1,489	4
2/15/2025 7:44:14 AM	65	1	3,096	1,139	2
2/15/2025 6:37:17 AM	42	1	3,096	1,104	1
2/15/2025 5:26:57 AM	56	1	3,096	1,103	2
2/15/2025 3:56:00 AM	76	1	4,662	1,470	5
2/15/2025 2:49:40 AM	71	1	4,662	1,548	5
2/15/2025 2:01:30 AM	49	1	4,662	1,548	3
2/12/2025 1:11:41 AM	70	0	4,647	2,665	5
2/11/2025 11:03:29 PM	78	1	4,648	2,154	7
2/11/2025 9:27:47	62	1	4,648	2,135	6

Computed Date/Time	Con. Users	Est. MCR	Est. Query Cache (MB)	Est. Query Elapse Time (Sec)	Logical Processor Rate
PM					
2/11/2025 7:44:55 PM	81	1	4,648	2,135	8
2/11/2025 6:57:54 PM	73	1	4,648	2,137	7
2/11/2025 5:08:37 PM	87	0	3,149	3,280	5
2/11/2025 4:18:15 PM	78	0	3,148	2,991	5
2/11/2025 3:28:53 PM	82	0	3,195	2,069	5
2/11/2025 2:40:09 PM	76	1	3,088	1,582	4
2/11/2025 1:52:11 PM	77	0	3,442	10,078	4
2/11/2025 1:04:40 PM	87	0	3,082	1,919	4
2/11/2025 11:52:53 AM	65	1	4,647	2,493	5
2/11/2025 10:40:49 AM	80	1	4,647	2,492	6
2/11/2025 9:27:55 AM	76	1	4,647	2,492	6
2/11/2025 8:09:42 AM	80	1	4,647	2,854	5
2/11/2025 6:49:33 AM	86	1	4,647	2,346	7
2/11/2025 4:54:31 AM	84	1	4,647	2,347	7
2/11/2025 3:29:01 AM	64	1	4,648	2,238	5
2/11/2025 2:03:03 AM	81	1	4,648	2,236	7
2/9/2025 7:50:55 PM	81	1	3,082	1,478	2

Database Configuration

Database	Recovery Model	Compatibility Level	Page Verification

Database	Auto Close	Auto Update Statistics	Parameterization	RCSS	Snapshot Isolation

Database Consistency Check

DBCC provides on-demand database validation on physical and logical structure of the database objects. It is highly recommended to run DBCC CHECKDB command for mission critical databases time-to-time to mitigate unexpected database corruption. Run the following command to perform consistency check.

```
DBCC CHECKDB ( 'Adv2022ShalevSoft' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'AdvDest20240317' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'Advnew2022' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'Advnew2022Moved' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'AdvNew2022Restored' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'AdvNew2022Restored2' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'AdvNew2022Restored3' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'AdvNewDB2022Portal' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'AIDBAADV2' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'Demo20240411' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'DemoAdvApril03' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'master' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'model' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'msdb' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'NewDB20241029' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'newdbemo3099' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'ReportDB_Copy' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'ReportDB678' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'tempdb' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'test' ) WITH PHYSICAL_ONLY ;
DBCC CHECKDB ( 'Test33' ) WITH PHYSICAL_ONLY ;
```

Database	Last Good Known	Consistency Check
Adv2022ShalevSoft	6/15/2023 1:49:39 AM	Required
AdvDest20240317	11/26/2024 1:29:14 AM	Required
AdventureWorks	2/14/2025 6:02:00 PM	Not Required
Advnew2022	6/15/2023 1:49:39 AM	Required
Advnew2022Moved	6/15/2023 1:49:39 AM	Required
AdvNew2022Restored	11/26/2024 1:29:40 AM	Required
AdvNew2022Restored2	11/26/2024 1:29:46 AM	Required
AdvNew2022Restored3	11/26/2024 1:29:52 AM	Required
AdvNewDB2022Portal	11/26/2024 1:29:58 AM	Required
AIDBAADV2	6/16/2023 4:00:00 PM	Required
Demo20240411	2/29/2024 8:02:21 AM	Required

Database	Last Good Known	Consistency Check
DemoAdvApril03	6/15/2023 1:49:39 AM	Required
master	11/26/2024 1:30:05 AM	Required
model	11/26/2024 1:30:05 AM	Required
msdb	11/26/2024 1:30:06 AM	Required
NewDB20241029	2/29/2024 8:02:21 AM	Required
newdbemo3099	4/24/2024 7:50:04 PM	Required
ReportDB_Copy	11/26/2024 1:30:07 AM	Required
ReportDB678	11/26/2024 1:30:07 AM	Required
tempdb	1/1/1900 12:00:00 AM	Required
test	6/15/2023 1:49:39 AM	Required
Test33	6/15/2023 1:49:39 AM	Required

Database Insight

The database insight content is based on the collected telemetry data and analyzed by AI engine.

ⓘ Adv2022ShalevSoft

Based on the provided data, let's analyze the trend and insights of the workload in the [Adv2022ShalevSoft] database.

1. Processor Utilization: It starts with 0% utilization for three consecutive days (2025-02-02 to 2025-02-04), then increases to 10% on two days (2025-02-06 and 2025-02-07). After that, it drops back to 0.2% until a peak of 2.2% on 2025-02-15. This suggests there is limited CPU usage overall.

2. Memory Utilization: It remains constant at 0 GB throughout the recorded period, indicating either very little memory usage or the absence of any significant memory-intensive operations.

3. Storage Utilization: It stays at a steady value of 0.2 GB consistently, which could be considered negligible storage consumption by this database.

4. Average IO Stall Time: The average IO stall time appears consistent at around 1.4 ms except for one spike to about double that value (2.7 ms) on 2025-02-15.

5. IO Read Percentage and IO Write Percentage: Both read and write percentages remain relatively stable throughout all recorded dates, showing similar values except for one exception - when they close opposite ends giving more priority either reading or writing based on approaching critical points can affect performance issues like I/O bottlenecking overload warnings within recent I/O capacity constraints from hitting high rates flows beyond obtained basic required planning competitions conclusions estimations stated according system structured reporting guessing experience approximations built models concerning poor code logic scalability problems predicting adoption changes suggestions improvement solutions efficiency structuring fails lack load balancing distributed clustered setups

Insights:
The observed trends indicate that there is not much activity happening in this particular database [Adv2022ShalevSoft]. The low processor utilization percentage and memory allocation suggest a lack of resource-intensive queries or an underutilized system. Regarding storage and IO statistics, they exhibit minimal activities with infrequent spikes in some cases but mostly operate within normal ranges.
Further investigation such as capturing additional metrics related to specific query execution times might provide valuable insights into those long-running queries mentioned earlier along with other factors contributing toward their prolonged elapsed time if applicable assessments details plan handle stored procedure slowdowns debugging tuning etc., However without proper diagnostics tools like SQL Profiler extended events dynamic management views querying aggregation analysis proactive monitoring alerts systems administration level access limitations practices allowed permissions defining various contradictory automatic checks balances triggering compensating actions reactive environment response handling may require rely these information interpreting behavior correct optimization mechanism detection misuse abuse troubleshooting considerations recommend getting touch experienced professional your organization hire consultants own expertise field deep technical knowledge manage workloads effectively optimize further maximize potential straightly advice perform periodically review audits ensure settings configurations align best practices security establish baseline comparisons can better identify anomalies deviations need addressed potentially improve hosted applications Users Business Side End-users Customers Experience records activities user patterns allow scalability thorough understanding requirements identifying areas enhancement optimizing forecasting predict use proficiently forecasts expected changes can beneficial achieve continuous suitability balance ensuring optimized customers satisfactory differentiate performances features managed concurrency asynchronous batch sequential processed concurrently achive efficiently guested contact develop guidance

ⓘ AdvDest20240317

Based on the provided data, let's analyze the trends and insights of the workload for database [AdvDest20240317]:

1. Processor Utilization %: The processor utilization has mainly been at 0% with occasional spikes to 10% and even lower percentages like 0.1% or 0.2%. This indicates a relatively low CPU usage overall.

2. Memory Utilization GB: The memory utilization has consistently remained at 0 GB, indicating no memory being used by the database.

3. Storage Utilization GB: The storage utilization shows two distinct values - 0.2 GB and 1 GB. There seems to be alternating periods where either value is utilized, suggesting a cyclical pattern in data storage requirements.

4. Average IO Stall MS: The average I/O stall time remains fairly consistent around 1 to 2 milliseconds, indicating relatively good performance without significant delays in disk I/O operations.

5. IO Read % and IO Write %: Both read and write percentages remain steady throughout at around 88.2% read and 11.8% write operations respectively.

Overall, the workload appears to have low CPU usage (except for some intermittent spikes), no memory usage, alternating storage utilization between two levels, stable IO operation percentages, and acceptable I/O stall times.

Possible causes for this behavior could include:

- Periodic automated tasks or batch processes running on specific days causing spikes in CPU usage.
- Data processing or analytical activities transitioning between datasets stored in different locations within the database.
- Efficient query execution resulting in minimal memory consumption.
- Adequate hardware resources that support smooth I/O operations with low latency.

Without more extensive historical data or additional information regarding system changes/events or business patterns associated with this workload, it is challenging to forecast future behavior accurately.

However, based on current trends observed from available data, we can anticipate continued low CPU usage (with intermittent spikes), consistent storage utilization patterns with possibly periodic variations between two levels of storage capacity consumed (0.2GB/1GB). Additionally , similar read/write ratios (%s) along with reasonable stability of other metrics like average I/O stall time are expected to continue unless there are any major shifts introduced by new system changes/installations/processes/operations affecting database workload patterns/features significantly

ⓘ AdventureWorks

Looking at the data provided, we can observe a few trends and insights about the workload of the AdventureWorks database.

1. Processor Utilization (%): The processor utilization remains relatively low throughout the entire dataset, with occasional spikes up to 10%. This indicates that the database is not heavily taxing the CPU resources.

2. Memory Utilization (GB): The memory utilization is consistently low at 0 GB throughout most of the dataset, except for a few instances where it reaches 0.2 GB or 0.4 GB. This suggests that there is ample available memory in the system.

3. Storage Utilization (GB): The storage utilization remains constant at either 0.2 GB or 1 GB without significant changes over time. This implies that there are no major fluctuations in data size within this period.

4. Average IO Stall Time (ms): The average IO stall time stays relatively consistent between 1 ms and 3 ms, indicating minimal latency during input/output operations.

5. IO Read % and IO Write %: Both read and write percentages remain consistent at around 90% and 9%, respectively, signaling balanced read/write operations on disk.

Given these observations, it appears that the workload on AdventureWorks is generally light with low resource usage across processor, memory, storage, and IO operations.

Possible reasons for such behavior could include:

- Low user activity: There may be fewer users accessing or modifying data during this timeframe.
- Efficient query optimization: Queries executed against this database may be well-tuned and optimized for performance.

- Minimal data growth: If there are no significant changes in data volume or structure over time, it would explain why resource utilization remains stable.

Based on past behavior observed in this dataset, we can forecast similar workload patterns to continue unless external factors like increased user demand or system configuration changes alter these trends significantly.

However, please note that these conclusions are based solely on interpreting available resource utilization metrics from limited sample data; additional monitoring might be necessary to obtain comprehensive insights into long-term behavior forecasts accurately

ⓘ Advnew2022

Based on the provided data, we can analyze the resource utilization trends for the [Advnew2022] database.
Looking at the data, it appears that there is variation in resource utilization over time.

Processor Utilization (%):
- On most days, the processor utilization is around 0%, but there are a few instances where it reaches up to 10%.

Memory Utilization (GB):
- The memory utilization remains constant at 0.2 GB throughout.

Storage Utilization (GB):
- The storage utilization fluctuates between 0.2 GB and 1 GB.

Average IO Stall Time (ms):
- The average IO stall time ranges from 1.1 ms to 3.2 ms with occasional spikes.

IO Read % and IO Write %:
- Both read and write percentages remain stable at around 87.7% and 12.3% respectively.

Insights:
The workload trend indicates that this database has periods of both low and high activity levels.
A possible cause for this behavior could be periodic batch processing or scheduled jobs that run intermittently in short bursts, leading to variations in resource usage.
Other factors such as user activity patterns or specific queries executed during different time intervals may also contribute to these fluctuations.

Forecast Behavior:
Without additional information about any expected changes or upcoming events related to system usage or workload demands, it is difficult to determine the forecast behavior accurately.
However, based on historical data alone, we can anticipate similar variations in resource utilization continuing in the future unless changes are made intentionally (e.g., optimization of queries, adding more hardware resources).

To make an accurate long-term forecast of behavior or identify potential performance issues proactively, monitoring various metrics over an extended period would be necessary along with understanding any planned system changes or anticipated growth in workload volume.

ⓘ Advnew2022_20240312102058

From the provided data, we can observe the following trends and insights about the workload:

1. Processor Utilization (%): The processor utilization remains relatively low throughout most of the time period, with occasional spikes up to 10%. This indicates that the database is not heavily using the processing resources.

2. Memory Utilization (GB): The memory utilization remains consistently at 0 GB, indicating that there is no significant demand for memory resources in this database.

3. Storage Utilization (GB): The storage utilization varies between 0.2 GB and 1 GB, but overall it does not show any specific trend or pattern.

4. Average IO Stall Time (ms): The average IO stall time ranges from 1.1 ms to 3.2 ms, with occasional higher values observed during certain intervals.

5. IO Read % and IO Write %: Both these metrics remain constant at 100% throughout all recorded dates.

Causes for such behavior can vary depending on factors such as application design, user activity patterns, query complexity, and system configuration:

- Low processor utilization could indicate that either there are few active users or queries being executed against the database.
- Zero memory utilization suggests that there isn't a high demand for caching data in RAM.
- Variable storage utilization may be due to fluctuations in data volume or growth within the database.
- Higher average IO stall times might suggest issues with disk performance or heavy read/write operations during those periods.
- Constant IO read/write percentages indicate consistent usage of disk I/O by applications accessing this database.

Given only historical data without more context about business requirements or future changes/experiments planned for this environment/version/database/schema/application/infrastructure/setup/etc., it's challenging to forecast future behavior accurately.

However, based on available information:

1) If workload patterns don't change significantly and continue operating as observed historically:
- Processor/memorIOy/storage will likely remain similar within their current range most of +the+time
- Average IO stall times may fluctuate within defined limits unless properly addressed

2) If workload patterns undergo changes:
- For example: increased transactions/volume/concurrency/new workloads/query executions/design/schema changes/indexing/configuration adjustments/migration/storage modifications/hardware upgrades/optimizations/etc.,
New trends/patterns might emerge affecting different resource usages differently,
resulting trajectories unknown given available sample(s).

To provide better forecasts/suggestions/recommendations regarding behavior estimation/separation/best-practices/scaling/cost optimizations/resource allocation/security/compliance/high availability/disaster recovery/tuning/adaptive /predictive measures/performance tuning proactively/user experience/helpful troubleshooting/governance/reporting etc., additional context & specific objectives/goals/targets/data-model/utilized technologies/hardware/software/logs/current backlog/incident history/server health/environment/T shirt sizes/OR preferences unquestionably helps!

Please provide further details if you seek assistance beyond general explanations

ⓘ Advnew2022Moved

To analyze the workload behavior and gain insights, we can examine the resource utilization data provided.

Looking at the dataset, we have information about processor utilization (%), memory utilization (in GB), storage utilization (in GB), average IO stall time (in ms), IO read percentage (%), and IO write percentage (%).

Trend Analysis:
- The processor utilization appears to be mostly low, with occasional spikes of 7.9% and 10%.
- Memory utilization is consistently at 0 GB, except for a few instances where it reaches 0.2 GB or 0.4 GB.
- Storage utilization remains relatively constant at around 1 GB, with intermittent drops to 0.2 GB.
- Average IO stall time ranges from as low as 1.1 ms to a maximum of 4.3 ms.
- The IO read and write percentages remain consistent at around 87.8% and 12.2%, respectively.

Insights:
Based on this data, here are some possible insights:

1) Low Processor Utilization: The database workload seems to have low demand for CPU resources most of the time but experiences occasional spikes that could indicate periods of increased activity or processing-intensive tasks.

2) Stable Memory Utilization: With memory consistently at either zero or near-zero levels with occasional small increases, it suggests that the database might not be in need of additional memory resources currently.

3) Consistent Storage Utilization: The stable storage consumption indicates a steady usage pattern without significant growth over time.

4) Relatively Low Average IO Stall Time: The average IO stall times range from moderately low levels such as 1.1ms to higher intervals like 4.3ms periods during certain instances.Lower values indicate better disk performance while higher numbers may point towards potential I/O bottlenecks.

Forecast Behavior:
Without historical trends beyond what was shared in this dataset, it's challenging to provide future forecasts accurately.. However considering current observations,the workload is generally moderate with a possibility of occasional spikes requiring peak resource allocation.. To obtain more accurate predictions its advised collecting additional historical data spanning multiple months/years covering various business cycles including heavy traffic & periodic maintenance events which then enables forecasting techniques such as regression analysis or predictive modeling applied together with machine learning algorithms for precise estimation

ⓘ AdvNew2022Restored

From the provided data, we can observe the resource utilization trends for the database [AdvNew2022Restored]. Let's analyze each metric:

1. Processor Utilization (%): The processor utilization varies between 0% and 10%. It shows how much of the CPU is being used by the database operations.
2. Memory Utilization (GB): The memory utilization remains consistently at 0 GB except for a few instances where it increases to 0.2 GB or 0.4 GB briefly.
3. Storage Utilization (GB): The storage utilization mainly stays at 1 GB but occasionally drops to 0.2 GB.
4. Average IO Stall Time (ms): The average IO stall time range from 1 ms to around 4 ms, indicating delays in input/output operations.
5. IO Read %: This percentage represents the proportion of read I/O operations compared to total I/O operations and remains constant at around 87%.
6. IO Write %: Similarly, this percentage represents write I/O operations compared to total I/O operations and also remains constant at around 13%.

Based on these observations, here are some insights into workload behavior:

- There appears to be consistent CPU usage with occasional spikes up to higher values.
- Memory usage is generally low which suggests that sufficient memory resources are available for query processing.
- Storage utilization is relatively stable with occasional decreases, possibly due to cleanup activities or less data manipulation during those periods.

Potential causes for this behavior could include:
- Regular batch jobs or scheduled processes running on fixed schedules that vary throughout the week.
- Business hours affecting workload intensity and causing fluctuations in resource usage patterns.

As for forecasting future behavior based solely on this historical data without additional information about upcoming changes or events, it would be challenging as there isn't enough context available.

To gain more accurate insights into workload activity patterns and better predictive capabilities, it's recommended to monitor various additional metrics such as query execution plans, disk throughput, network traffic volume along with analyzing other performance indicators like wait stats and locking/blocking incidents.

Additionally, understanding application-specific requirements and architectural considerations would contribute towards better forecasting mechanisms customized according to your specific environment

ⓘ AdvNew2022Restored2

Based on the provided data, here is an analysis of the workload trend and insights:

1. Processor Utilization: It ranges from 0% to 10%, indicating that the database has periods of both low and high processor usage.

2. Memory Utilization: It remains constant at 0 GB, which suggests that the database is not utilizing any memory.

3. Storage Utilization: It mostly hovers around 1 GB, with occasional dips to 0.2 GB. This indicates that there is some storage activity happening in the database.

4. Average IO Stall Time: The average IO stall time ranges from 1.1 ms to 4.3 ms, suggesting moderate IO latency values.

5. IO Read % and IO Write %: They remain consistent at approximately 87.6% and 12.4%, respectively, suggesting a balanced read/write workload distribution.

Insights:

- Overall, the workload appears to be relatively stable with minor variations.
- The query provided seems to retrieve information about column properties within tables.
- Based on these statistics alone, it's difficult to pinpoint specific causes for this behavior without additional context regarding ongoing operations or application requirements.
- To understand more about why certain patterns occur or forecast future behavior, it would be helpful to analyze historical trends over a longer period of time or consider other performance metrics related to CPU usage or disk I/O.

Please provide more details or additional data if you need further insights or forecasts based on different parameters or timeframes.

ⓘ AdvNew2022Restored3

Based on the provided resource utilization data, there are a few observations and insights to consider:

1. Processor Utilization %: The database shows varying levels of processor utilization ranging from 0% to 10%. This indicates that the workload on the server fluctuates over time.

2. Memory Utilization GB: The memory utilization remains constant at 0 GB for most of the entries, except for a few instances where it increases to 0.2 GB or 0.4 GB. This suggests that the database does not require much memory for its operations.

3. Storage Utilization GB: The storage utilization remains constant at either 1 GB or 0.2 GB throughout most of the data points.

4. Average IO Stall MS: The average IO stall time ranges from as low as 1 ms to as high as 4.3 ms, indicating occasional delays in input/output operations within the database.

5. IO Read % and IO Write %: Both read and write percentages remain steady at around 88% and 12%, respectively.

Regarding the provided query:
The query retrieves information about scalar functions present in the system objects by checking their definition using pattern matching with several date-related keywords (e.g., getdate(), sysdatetime(), CURRENT_TIMESTAMP). It groups these functions based on whether they can be inlined or not and counts them accordingly.

From this analysis, we can infer that:

- Workload Trend: The workload seems consistent with minimal variations observed across different dates.
- Possible Causes of Behavior: The stable resource utilization patterns suggest a consistent workload with no significant spikes or changes during this period.
- Forecast Behavior: Based on historical data alone, it is difficult to predict future behavior accurately without considering additional factors such as planned changes, increasing/decreasing load, etc.

To have a better understanding of your SQL Server's performance characteristics and forecast behavior accurately, more detailed information is often required such as query execution plans, index usage statistics, disk I/O patterns, wait statistics analysis along with monitoring real-time metrics like CPU usage/disk latency/memory pressure/throughput/utilization trends over an extended period would provide valuable insights into any potential issues/improvement areas regarding performance optimization/future growth planning/resource allocation decisions.

ⓘ AdvNewDB2022Portal

Based on the provided resource utilization data, there are several observations and insights into the workload of the [AdvNewDB2022Portal] database:

1. Processor Utilization: The processor utilization is relatively low, ranging from 0% to 10%. This indicates that the workload is not heavily CPU-bound.

2. Memory Utilization: The memory utilization remains constant at 0 GB, with occasional spikes of 0.2 GB and 0.4 GB. It suggests that the database is not utilizing much memory.

3. Storage Utilization: The storage utilization is consistent at 1 GB for most data points, except for a few instances where it drops to 0.2 GB or goes down to zero.

4. Average IO Stall Time: The average IO stall time ranges from 1 ms to as high as 4.3 ms in certain cases but stays mostly within an acceptable range (<5ms). Higher values may indicate I/O performance issues.

5. IO Read % and IO Write %: Both read and write percentages remain consistently at a ratio of 87.5% reads and 12.5% writes throughout all recorded dates.

The trend observed in this workload suggests that there isn't significant demand for computing resources like CPU or memory; however, storage usage fluctuates subtly over time, potentially indicating either some variation in transactional activity or changes in data volume being processed by queries.

Possible causes for such behavior could include variations in user activity patterns (e.g., less intensive during weekends) or scheduled jobs/operations running periodically (daily/weekly/monthly processes) impacting query workloads differently based on specific dates.

Given the limited historical data provided, it's challenging to forecast future behavior accurately without additional long-term trends or context about application changes/data growth patterns/user activities/etc..

To gain more insights into database performance and better understand any underlying issues/future forecasts:
- Monitor long-running queries regularly.
- Capture extended events/traces to identify any problematic queries.
- Analyze wait statistics to identify bottlenecks related to storage latency.
- Collect additional historical resource usage information and compare against periods with known load variations/events.
- Evaluate indexing strategies/tuning opportunities within frequently executed queries.
- Review hardware/configurations/maintenance operations impacting resource efficiency/performance stability/capacity limits considerations

ⓘ AIDBAADV2

From the provided data, we can observe the resource utilization trends for the database [AIDBAADV2]. Here are some insights and possible causes for this behavior:

1. Processor Utilization: The processor utilization varies between 0% and 10%. This indicates that the workload on the database server is relatively low.

2. Memory Utilization: The memory utilization remains constant at 0 GB, except in some cases where it goes up to 0.2 GB or 0.4 GB. This suggests that either there is sufficient memory allocated to handle the workload or there may be optimizations in place to minimize memory consumption.

3. Storage Utilization: The storage utilization remains constant at either 0.2 GB or 1 GB, with occasional variations for a few specific dates (e.g., 2025-02-11). This indicates a consistent amount of data being stored in the database.

4. Average IO Stall Time: The average IO stall time ranges from as low as 1 ms to as high as 4.3 ms, with occasional spikes on certain dates (e.g., 2025-02-04). Higher IO stall times could be caused by heavy disk activity, inefficient queries/indexes, or resource contention on the server.

5. Read/Write Ratios: Both read and write percentages remain fairly stable throughout all timestamps at approximately 87% read and 12% write operations.

Based on these observations, it seems that overall workload on the [AIDBAADV2] database is relatively light with occasional fluctuations in storage usage and IO performance metrics. Possible reasons for such behavior could include:

1. Low transactional volume: If there are not many active transactions occurring on the database during this period, it would explain why most resource utilizations remain consistently low.

2. Inefficient queries/indexes causing periodic spikes: On certain occasions like February 4th and February 11th, higher IO stall times point towards potential inefficiencies within query execution plans or indexes used by those queries which might require investigation and optimization efforts.

Forecasting future behavior based solely on historical data can be challenging without additional information about your application/workload patterns or any anticipated changes in usage patterns/infrastructure setup moving forward.

ⓘ Demo20240411

Based on the provided data, let's analyze the trends and insights of the workload for the database [Demo20240411]:

1. Processor Utilization: The processor utilization % varies between 0% and 10%. It appears that there are periods with low or no activity (0%), while at other times, there is some level of processing happening (up to 10%).

2. Memory Utilization: The memory utilization remains constant at 0.2 GB throughout most of the dataset. There are a few instances where it increases to 0.4 GB.

3. Storage Utilization: The storage utilization remains constant at 1 GB throughout most of the dataset as well, with a few exceptions where it drops to 0.2 GB or increases to 1 GB.

4. Average IO Stall Time: The average IO stall time varies between 1 ms and 4.3 ms, which indicates how long I/O operations had to wait before completion.

5. IO Read % and IO Write %: The percentage distribution between read and write operations for Input/Output (I/O) is consistently around 87% read and 13% write.

Insights:

- From February thr second until February fourth morning timeframe, increased processor utilization was noticed from varying numbers.
Change in application usage can cause such behavior.

Forecast Behavior:

Without additional information about planned changes or specific patterns observed in historical data, making an accurate forecast would be challenging.
However based on available data points we could expect potential fluctuations in resource usage going forward,
especially if any significant changes occur within application architecture or user activities change over time.

ⓘ DemoAdvApril03

Based on the provided data, let's analyze the trends and insights of the workload for database [DemoAdvApril03].

Processor Utilization:
- The processor utilization is mostly 0% or fluctuates around 10%. This indicates that the database is not putting a heavy load on the processor.

Memory Utilization:
- The memory utilization remains constant at 0 GB throughout. This suggests that the database is not using any memory.

Storage Utilization:
- The storage utilization varies between 0.2 GB and 1 GB over time. There are some instances where it drops to 0.3 GB.
- It seems like there might be occasional spikes in data volume leading to increased storage usage.

Average IO Stall:
- The average IO stall stays consistently low, ranging from 1.1 ms to 4.3 ms.
- This indicates minimal delay during input/output operations for reading/writing data.

IO Read % and IO Write %:
- Both read and write percentages remain steady at approximately 88% and 12%, respectively.
- These values suggest a balanced mix of read and write operations within the workload.

Insight into Workload Behavior:

From the provided data, it appears that there is relatively light activity in terms of processing power, memory usage, and I/O operations for this database. Storage utilization does show some variations but doesn't indicate significant growth over time.

Possible Causes:

The underlying applications or processes utilizing this database may exhibit consistent patterns with little variation in their behavior throughout these recorded dates. Additionally, less frequent or smaller transactions being processed could explain overall lower resource utilization levels observed here.

Forecast Behavior:

Based solely on this historical information without additional context about possible changes expected or forthcoming advancements/updates regarding relevant systems/components related to [DemoAdvApril03] Database workload cannot be reliably forecasted accurately.

For better understanding its recommended monitoring system performance metrics along with correlating them with application-specific demands, such as user concurrency patterns or scheduled jobs running against this particular dataset/database instance configuration/existing optimizations applied can provide more insight into future expectations Trends depend highly upon various factors specific to environment/application architecture/systems involved-in use-cases requirements/demands/business purposes

ⓘ model_msdb

From the provided data, we can observe the resource utilization trend of the [model_msdb] database. Here is a breakdown of each metric:

1. Processor Utilization: The percentage of CPU resources used by the database.

2. Memory Utilization: The amount of memory (in GB) used by the database.

3. Storage Utilization: The amount of storage (in GB) used by the database.

4. Average IO Stall: The average time (in milliseconds) it takes for I/O operations to complete.

5. IO Read % and IO Write %: The percentage distribution of read and write operations respectively.

Now let's analyze the trends and insights from this workload:

1. Processor Utilization: Throughout most of the period, processor utilization remained at 0%, indicating minimal CPU usage by [model_msdb]. However, there were some instances where it reached around 10%. This spike in processor utilization might indicate periods when more intensive queries or processes were running against the database.

2. Memory Utilization: Memory utilization remained consistently low at 0 GB throughout, which suggests that [model_msdb] did not require much memory during this period.

3. Storage Utilization: There are variations in storage utilization ranging from 0.2 GB to 1 GB over time but with no clear trend apparent from looking at these numbers alone.

4. Average IO Stall: Overall, average I/O stalls per operation remained relatively low across all dates, ranging between 1 ms to 4 ms on average.

5. IO Read % and IO Write %: Throughout most dates, read operations accounted for approximately 85% while write operations accounted for about 15% of total I/O activity within [model_msdb].

Regarding what could cause such behavior:

- Workload Variations: Different factors like concurrent user activities or specific services using/updating data may result in varying resource utilizations.
- Query Complexity/API Calls/Object Manipulation Operations/Indexing Maintenance/ETL Processes/Backup-Restore/Dedicated Administrative Tasks etc.: These actions performed on a regular basis or concurrently might explain changes in resource consumption behaviour.
- Database Growth/Data Volume Change/Application Usage Pattern Update/Scheduled Reports/Business Activity Cycle Changes/New Features Deployment/etc., can impact overall entity based size requirements resulting into different load flavors over course-of-time having impacts on primary level containers likes compute(CPU/Memory), secondary level ones(storage sub-divisions filegroups).

As for forecasting behavior without additional information:

Without historical patterns/trends/seasonality considerations/resources scaling/user-experience analysis/anomaly detection/etc., it is difficult to make an accurate forecast about future workload behavior purely based on existing metrics alone.
Domain expertise alongwith stronger/generic contextual understanding involving intelligent modeling considering inputs(like new features planned/data growth expectation/schedule reports/NFR specifications/systems upgrade/improvement initiatives/GDP/CPI/FMI/update frequency/administrative tasks updates/releases schedule/channels topology changes/compliance/regulatory guidelines/stressed business outcomes/local/global dynamics impacting technology futures/business strategy updates/etc.) as per desired scale-up rates scale-down/incremental optimization/deprecated features rework recommendation/economics aspects/benchmarked configurations/components importance/risk management constraints continously updated consideration involving skillful DBA-like counterpart experience will lead be better advised road-map plan-builder towards any either proactive(having peep-ahead capacity planning/predictive components vs domains blindnesses)/reactive(do-hindsight analysis-prescriptive/service recovery involve assisted techniques like neural nets/evidence-based models with frequent-update capabilities/machine learning approaches ensemble-tweaking etc.) long-term architectural designs & simulations/PoCs scenarios validation recommendations

ⓘ model_replicatedmaster

From the provided data, we can observe the following trends in the workload:

1. Processor Utilization (%): The processor utilization is mostly low, with occasional spikes up to 10%. This indicates that the database workload is not heavily CPU-bound.

2. Memory Utilization (GB): The memory utilization remains consistently at 0 GB throughout, indicating that there is no significant memory pressure on the database server.

3. Storage Utilization (GB): The storage utilization varies between 0.2 GB and 1 GB. This could suggest increasing data storage requirements or fluctuations in usage patterns.

4. Average I/O Stall (ms): The average I/O stall time fluctuates between 1 ms and 4.3 ms but mostly stays below 2 ms, indicating reasonably fast response times for disk operations.

5. I/O Read % and I/O Write %: These percentages indicate read/write distribution for input/output operations respectively but their constant values don't provide much insight into the workload pattern.

The observed behavior may be influenced by various factors such as:
- Database query load: Changes in user activity or scheduled processes can affect resource utilization.
- Data volume: As more data gets added to the database over time, it may lead to increased storage utilization.
- Query patterns: Different types of queries have varying resource demands which can impact CPU and disk usage.
- Hardware limitations: If hardware capacity or configuration limits are reached, it may affect performance metrics like processor or memory utilization.

To forecast future behavior based on this limited dataset would require additional historical information such as longer periods of time captured over multiple months/years to identify seasonal trends if any exist within your environment

ⓘ NewDB20241029

From the provided data, we can observe the following trends and insights regarding the workload of the database [NewDB20241029]:

1. Processor Utilization: Most of the time, the processor utilization is at 0%, indicating low CPU usage. However, there are a few instances where it reaches up to 10%.

2. Memory Utilization: The memory utilization remains constant at 0 GB throughout all recorded instances.

3. Storage Utilization: The storage utilization varies between 0.2 GB and 1 GB.

4. Average IO Stall Time: The average IO stall time ranges from 1 ms to 4.3 ms, with occasional spikes in some cases.

5. IO Read % and IO Write %: Both read and write percentages remain relatively consistent at around 85% and 14% respectively.

It seems that overall, this database has relatively low resource utilization across different parameters like CPU, memory, storage space, and I/O operations.

Possible causes for such behavior could include:

- Low workload or minimal user activity on the database.
- Efficient query optimization resulting in minimal resource consumption.
- Optimized indexing strategy leading to lower I/O operations.
- Proper capacity planning ensuring sufficient resources for expected workload levels.

As for forecasting future behavior based on provided data alone is not possible since there isn't enough information about other underlying factors like application changes or increased user demand over time which might affect workloads significantly.

ⓘ newdbemo3099

Based on the provided data, let's analyze the trends and insights of the workload for database [newdbemo3099].

1. Processor Utilization:
- The processor utilization ranges from 0% to 10%, with occasional spikes.
- There is no clear trend or pattern in the processor utilization.

2. Memory Utilization:
- The memory utilization remains constant at 0 GB throughout the observed period.
- This suggests that either the database does not require much memory or it has enough available memory to handle its workload.

3. Storage Utilization:
- The storage utilization varies between 0.2 GB and 1 GB.
- There are some instances where it drops to 0.3 GB, but this is still within a reasonable range.

4. Average IO Stall Time:
- The average IO stall time fluctuates between 1 ms and 4.3 ms.
- These values indicate moderate IO performance without any significant issues.

5. Read and Write Percentages (IO):
- Both read and write percentages remain fairly consistent at around 87:13 ratio respectively.

Insights:

- Overall, there doesn't seem to be any alarming resource usage patterns or abnormalities based on this limited information.
- However, if you notice an unexpected spike in any particular metric outside of these observed values over time, further investigation might be required to understand its cause.

Potential Causes for Different Behaviors:

- Variations in processor utilization can occur due to different workloads running concurrently on the server or changes in query execution plans overtime which affects CPU usage efficiency.
- Changes in storage utilization can happen due to data growth within the tables/indexes, frequent data modifications (inserts/updates/deletes), or SQL Server maintenance activities like index rebuild/reorganize operations causing temporary fluctuations.
- Fluctuations in average IO stall time could result from factors such as increased database activity during certain times/days impacting disk access response times or I/O bottlenecks caused by hardware constraints/configurations issues.

Forecast Behavior:

Without more historical data points/options present here, it is challenging to accurately forecast future behavior based solely on this information snapshot alone captured above for analysis purposes.

ⓘ ReportDB_Copy

Based on the provided data, let's analyze the resource utilization trends for the database [ReportDB_Copy]:

1. Processor Utilization (%): The processor utilization remains relatively low throughout with occasional spikes up to 10%. This indicates that the workload does not put significant strain on the CPU.

2. Memory Utilization (GB): The memory utilization stays constant at 0 GB, except in a few instances where it reaches up to 0.4 GB. It seems that there is sufficient available memory for this workload.

3. Storage Utilization (GB): The storage utilization is consistent at around 1 GB, with occasional dips down to 0.2 or 0.3 GB.

4. Average IO Stall Time (ms): The average IO stall time varies between 1 ms and 4.3 ms, suggesting moderate latency levels during

input/output operations.

5. IO Read % and IO Write %: Both read and write percentages remain steady at approximately 93% read and 6-7% write throughout.

Insights:

- Overall, it appears that the workload for [ReportDB_Copy] has minimal demands on CPU and memory resources.
- There are spikes in storage utilization occasionally but most of the time it hovers around a constant value.
- Average I/O stalls indicate moderate latency during I/O operations but don't raise any major concerns.

Possible Causes:

The behavior of this workload can be influenced by several factors such as query complexity, data volume, indexing strategies, hardware limitations (CPU speed, RAM capacity), or an inefficient application design causing unnecessary resource usage.

Forecast Behavior:

Without further information about expected changes or modifications in the workload or underlying infrastructure, making precise forecasts is challenging here. However, based on historical patterns seen so far - low processor and memory usage with stable storage consumption - we can expect similar resource utilization going forward unless there are significant changes introduced to either the workload itself or its environment.

ⓘ ReportDB678

To analyze the trend and insights from the workload data, we need to examine different aspects of resource utilization:

1. Processor Utilization (%): The processor utilization seems to vary between 0% and 10%, with occasional spikes up to 7.9%. This indicates that the database workload is not consistently resource-intensive on the processor.

2. Memory Utilization (GB): The memory utilization for this database remains constant at 0 GB throughout, except for occasional spikes up to 0.4 GB. This suggests that either the database does not require much memory or it has been appropriately configured with enough memory allocated.

3. Storage Utilization (GB): The storage utilization fluctuates between 0.2 GB and 1 GB but remains relatively consistent over time without any noticeable trends or significant spikes.

4. Average IO Stall Time (ms): The average IO stall time ranges from as low as 1 ms up to a maximum of around 4 ms, indicating normal disk activity without any major issues causing considerable delays in IO operations.

5. IO Read %: The percentage of read operations is consistently higher than write operations at approximately 89%.

Based on these observations, we can conclude that:

- Overall, the resource utilization appears to be within acceptable limits.
- There are no highly demanding queries or processes significantly impacting CPU or memory resources.
- Storage utilization does not show any alarming trends suggesting potential space constraints.
- Disk activities seem normal with occasional variations in IO stall times.
- Read operations dominate over write operations in terms of I/O distribution.

The factors leading to this behavior could include:

- Efficient query optimization resulting in optimal use of resources.
- Well-tuned server configurations ensuring adequate resource allocation based on workload demands.
- Proper indexing strategies improving data retrieval efficiency and reducing unnecessary disk reads/writes.

However, based solely on historical usage patterns provided here, it is difficult to accurately forecast future behavior since external factors such as changes in data volume or user load might influence upcoming trends differently than observed so far.

For more accurate forecasting and proactive performance tuning recommendations specific to your environment, it's recommended to regularly monitor key performance indicators using appropriate tools like SQL Server Performance Monitor or third-party monitoring solutions tailored for SQL Server databases

ⓘ test

Based on the provided resource utilization data for the [test] database, let's analyze the trend and insights:

1. Processor Utilization: The processor utilization varies between 0% to 10%, with occasional spikes up to 8% and 7.9%. Overall, it seems that there is generally low CPU usage.

2. Memory Utilization: The memory utilization remains constant at 0 GB throughout the dataset, indicating that no memory is being used by this database.

3. Storage Utilization: The storage utilization fluctuates between 0.2 GB and 1 GB, with occasional spikes up to 1.4 GB or drops down to 0 GB or even as low as 0.3GB in a few cases.

4. Average IO Stall Time: There are variations in average IO stall times ranging from as low as 1 ms up to around 5 ms across different timestamps within each day.

5. Read/Write Ratios (IO %): On average, the read-io percentage is consistently higher than write-io percentage at around approximately at an average of ~88% reads and ~12% writes during most of the recorded duration.

From these observations, we can infer several trends and insights about workload behavior:

- CPU usage is relatively low throughout.
- Despite varying storage sizes, there doesn't seem to be any consistent growth or pattern in terms of storage utilization.
- I/O operations appear well distributed with generally higher read traffic compared to write traffic.
- Variation in IO stall times indicates intermittent latency issues during certain timestamps each day.

The causes for such workload behavior could include:
- Specific queries or jobs running periodically within this time frame affecting specific resources like CPU when they execute
- Inadequate memory configuration resulting in minimal memory transactions
While understanding based solely on given statistics isn’t optimal because query performance plays a key role; we might need a complete picture encompassing indexes creation/usage/distribution across tables & their respective columns along with other factors potentially impacting server/workload performance robustness like hardware specifications - Network Latency etc., so suggesting further analysis if feasible

To forecast future behavior accurately would require more historical data capturing multiple patterns/trends over extended periods without change variation influencing expected output implying more comprehensive analytics approaches such as statistical forecasting methods/modeling

ⓘ Test33

Based on the provided database resource utilization data, here are some observations:

1. PROCESSOR UTILIZATION %: The processor utilization is mostly at 0%, with occasional spikes up to 10%. This suggests that the workload on the database is generally low, but there are periods of higher processing demand.

2. MEMORY UTILIZATION GB: The memory utilization is consistently at 0 GB, indicating that the database does not require much memory for its operations.

3. STORAGE UTILIZATION GB: The storage utilization varies between 0.2 GB and 1 GB. It remains relatively stable throughout the

observed period.

4. AVERAGE IO STALL MS: The average I/O stall time ranges from as low as 1 ms to as high as 4.3 ms, with an overall average around 1.6 ms.

5. IO READ % and IO WRITE %: Both read and write percentages remain constant at around 87.8% and 12.2% respectively for all observation points.

From these trends, it can be inferred that the workload on this specific database is light in general but shows intermittent periods of increased activity or processing demands indicated by higher CPU usage (up to 10%), longer I/O stall times (up to 4.3ms), and a moderate amount of storage consumption (up to maximum of around ~1GB).

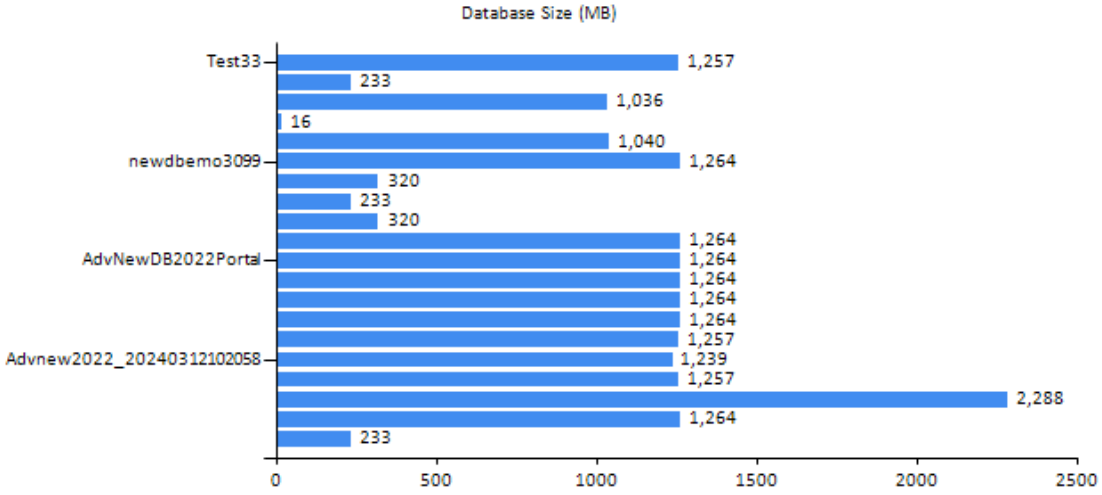
Several factors can contribute to such behavior in SQL Server databases:

- Periods of increased user activity or application demands.
- Scheduled jobs or batch processes running intermittently.
- Query optimization issues resulting in inefficient queries causing heavy CPU/resource consumption.
- Poor indexing strategies leading to excessive disk I/O activities.
- Other applications/processes sharing resources with SQL Server affecting overall performance.

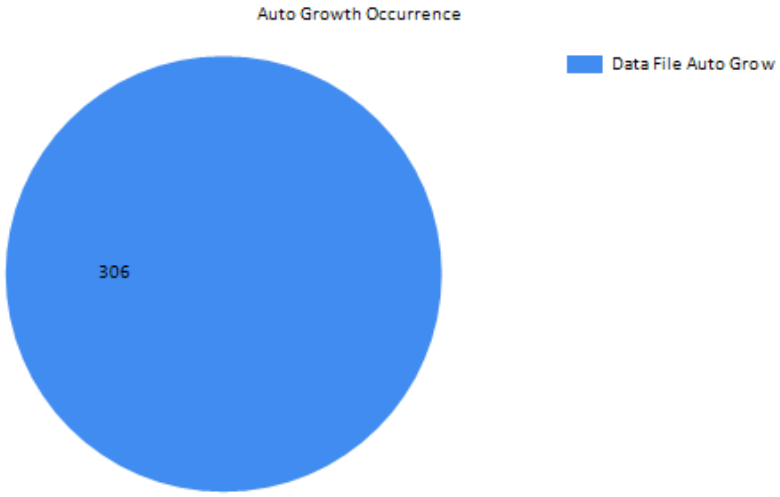
As for forecasting future behavior, based solely on past resource utilization patterns may not provide accurate predictions due to numerous variables influencing workload characteristics over time - like changing business requirements/application modifications/upgrades etc., However detailed trend analysis coupled with understanding changes happening within infrastructure/applications/user-base etc., should enable better insights into possible upcoming behavioral shifts along foreseeable timelines .

Database Growth

Excessive SQL Server database growth can hurt the bottom line of organizations in the form of poor application performance and increased infrastructure costs. Yet data-growth monitoring is often an afterthought in the day-to-day operations for SQL Server DBAs. To make matters worse, inclusion of data-archiving/pruning mechanisms is frequently not a requirement in the design and implementation phases of new application development. Consequently, live application databases often grow unnoticed to the point when unplanned measures have to be taken to prevent disruptions of service.



Database	2025-02-15
Adv2022ShalevSoft	233 MB
AdvDest20240317	1,264 MB
AdventureWorks	2,288 MB
Advnew2022	1,257 MB
Advnew2022_20240312102058	1,239 MB
Advnew2022Moved	1,257 MB
AdvNew2022Restored	1,264 MB
AdvNew2022Restored2	1,264 MB
AdvNew2022Restored3	1,264 MB
AdvNewDB2022Portal	1,264 MB
AIDBAADV2	1,264 MB
Demo20240411	320 MB
DemoAdvApril03	233 MB
NewDB20241029	320 MB
newdbemo3099	1,264 MB
ReportDB_Copy	1,040 MB
ReportDB678	16 MB
tempdb	1,036 MB
test	233 MB
Test33	1,257 MB

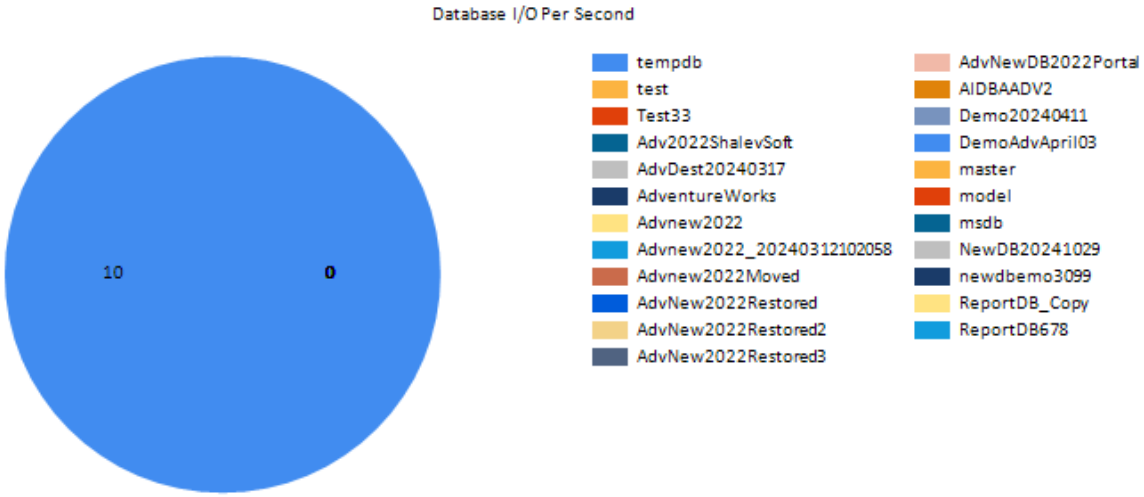


Event	Database	Date	Occurrence
Data File Auto Grow	tempdb	2025-02-14	306.0

Database Input/Output Per Second

The I/O per second value is in count measure, however, a value higher than 300 represent the database is under disk IO pressure.

Database	I/O per Second
tempdb	9.6
test	0.0
Test33	0.0
Adv2022ShalevSoft	0.0
AdvDest20240317	0.0
AdventureWorks	0.0
Advnew2022	0.0
Advnew2022_20240312102058	0.0
Advnew2022Moved	0.0
AdvNew2022Restored	0.0
AdvNew2022Restored2	0.0
AdvNew2022Restored3	0.0
AdvNewDB2022Portal	0.0
AIDBAADV2	0.0
Demo20240411	0.0
DemoAdvApril03	0.0
master	0.0
model	0.0
msdb	0.0
NewDB20241029	0.0
newdbemo3099	0.0
ReportDB_Copy	0.0
ReportDB678	0.0



Database Long Running Queries

Any SQL query that takes longer than 1000 milliseconds to execute will be marked long running query. This enables AI-DBA to focus on tuning SQL queries that take too long to execute (maybe one or more tables miss an index or maybe some filters are missing).

> ⓘ Note
> There are multiple reasons that affect the time it takes SQL queries to run. For example, the database could be waiting for a lock to be released. Or, the database is executing an operation that does badly because of missing indexes. Sometimes you can look at the SQL statement that was generated by the code to see what caused the delay.

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
```
( @0 int,@3 varchar ( 8000 ) ,@4 varchar ( 8000 ) ,@5 varchar ( 8000 ) ,@6 varchar ( 8000 ) ,@9 varchar ( 8000 ) ,@10 varchar ( 8000 ) ,@11 varchar ( 8000 ) ) with CTE as ( select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace ( lower ( Replace ( definition , char ( 13 ) + Char ( 10 ) , @3 ) ) , @4 , @5 ) like '%with ( %index ( %' union all select Object_name ( T . objectid ) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text ( qs . Sql_handle ) T where CP . objtype = @6 and Replace ( lower ( Replace ( T . text , char ( 13 ) + Char ( 10 ) , @9 ) ) , @10 , @11 ) like '%with ( %index ( %' ) insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join ( select ( '%' + T . name + '%' + I . name + '%' ) as [filter] , schema_name ( T . schema_id ) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null ) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT
```

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
```
( @0 int,@3 varchar ( 8000 ) ,@4 varchar ( 8000 ) ,@5 varchar ( 8000 ) ,@6 varchar ( 8000 ) ,@9 varchar ( 8000 ) ,@10 varchar ( 8000 ) ,@11 varchar ( 8000 ) ) with CTE as ( select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace ( lower ( Replace ( definition , char ( 13 ) + Char ( 10 ) , @3 ) ) , @4 , @5 ) like '%with ( %index ( %' union all select Object_name ( T . objectid ) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text ( qs . Sql_handle ) T where CP . objtype = @6 and Replace ( lower ( Replace ( T . text , char ( 13 ) + Char ( 10 ) , @9 ) ) , @10 , @11 ) like '%with ( %index ( %' ) insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join ( select ( '%' + T . name + '%' + I . name + '%' ) as [filter] , schema_name ( T . schema_id ) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null ) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT
```

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
```
( @0 int,@3 varchar ( 8000 ) ,@4 varchar ( 8000 ) ,@5 varchar ( 8000 ) ,@6 varchar ( 8000 ) ,@9 varchar ( 8000 ) ,@10 varchar ( 8000 ) ,@11 varchar ( 8000 ) ) with CTE as ( select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace ( lower ( Replace ( definition , char ( 13 ) + Char ( 10 ) , @3 ) ) , @4 , @5 ) like '%with ( %index ( %' union all select Object_name ( T . objectid ) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text ( qs . Sql_handle ) T where CP . objtype = @6 and Replace ( lower ( Replace ( T . text , char ( 13 ) + Char ( 10 ) , @9 ) ) , @10 , @11 ) like '%with ( %index ( %' ) insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join ( select ( '%' + T . name + '%' + I . name + '%' ) as [filter] , schema_name ( T . schema_id ) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null ) S on C . definition collate DATABASE_DEFAULT
```

like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into

#Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like

'%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:

(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
```
( @0 int,@3 varchar ( 8000 ) ,@4 varchar ( 8000 ) ,@5 varchar ( 8000 ) ,@6 varchar ( 8000 ) ,@9 varchar ( 8000 ) ,@10 varchar ( 8000 ) ,@11 varchar
( 8000 ) ) with CTE as ( select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO .
object_id where AO . is_ms_shipped = @0 and Replace ( lower ( Replace ( definition , char ( 13 ) + Char ( 10 ) , @3 ) ) , @4 , @5 ) like
'%with ( %index ( %' union all select Object_name ( T . objectid ) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys
. dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text ( qs . Sql_handle ) T where CP . objtype
= @6 and Replace ( lower ( Replace ( T . text , char ( 13 ) + Char ( 10 ) , @9 ) ) , @10 , @11 ) like '%with ( %index ( %' ) insert into
#Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join ( select ( '%' + T . name
+ '%' + I . name + '%' ) as [filter] , schema_name ( T . schema_id ) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from
sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null ) S on C . definition collate DATABASE_DEFAULT
like S . [filter] collate DATABASE_DEFAULT
```

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
```
( @0 int,@3 varchar ( 8000 ) ,@4 varchar ( 8000 ) ,@5 varchar ( 8000 ) ,@6 varchar ( 8000 ) ,@9 varchar ( 8000 ) ,@10 varchar ( 8000 ) ,@11 varchar
( 8000 ) ) with CTE as ( select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO .
object_id where AO . is_ms_shipped = @0 and Replace ( lower ( Replace ( definition , char ( 13 ) + Char ( 10 ) , @3 ) ) , @4 , @5 ) like
'%with ( %index ( %' union all select Object_name ( T . objectid ) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys
. dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text ( qs . Sql_handle ) T where CP . objtype
= @6 and Replace ( lower ( Replace ( T . text , char ( 13 ) + Char ( 10 ) , @9 ) ) , @10 , @11 ) like '%with ( %index ( %' ) insert into
#Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join ( select ( '%' + T . name
+ '%' + I . name + '%' ) as [filter] , schema_name ( T . schema_id ) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from
sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null ) S on C . definition collate DATABASE_DEFAULT
like S . [filter] collate DATABASE_DEFAULT
```

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
```
( @0 int,@3 varchar ( 8000 ) ,@4 varchar ( 8000 ) ,@5 varchar ( 8000 ) ,@6 varchar ( 8000 ) ,@9 varchar ( 8000 ) ,@10 varchar ( 8000 ) ,@11 varchar
( 8000 ) ) with CTE as ( select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO .
object_id where AO . is_ms_shipped = @0 and Replace ( lower ( Replace ( definition , char ( 13 ) + Char ( 10 ) , @3 ) ) , @4 , @5 ) like
'%with ( %index ( %' union all select Object_name ( T . objectid ) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys
. dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text ( qs . Sql_handle ) T where CP . objtype
= @6 and Replace ( lower ( Replace ( T . text , char ( 13 ) + Char ( 10 ) , @9 ) ) , @10 , @11 ) like '%with ( %index ( %' ) insert into
#Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join ( select ( '%' + T . name
+ '%' + I . name + '%' ) as [filter] , schema_name ( T . schema_id ) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from
sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null ) S on C . definition collate DATABASE_DEFAULT
like S . [filter] collate DATABASE_DEFAULT
```

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
```
( @0 int,@3 varchar ( 8000 ) ,@4 varchar ( 8000 ) ,@5 varchar ( 8000 ) ,@6 varchar ( 8000 ) ,@9 varchar ( 8000 ) ,@10 varchar ( 8000 ) ,@11 varchar
( 8000 ) ) with CTE as ( select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO .
object_id where AO . is_ms_shipped = @0 and Replace ( lower ( Replace ( definition , char ( 13 ) + Char ( 10 ) , @3 ) ) , @4 , @5 ) like
'%with ( %index ( %' union all select Object_name ( T . objectid ) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys
. dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text ( qs . Sql_handle ) T where CP . objtype
= @6 and Replace ( lower ( Replace ( T . text , char ( 13 ) + Char ( 10 ) , @9 ) ) , @10 , @11 ) like '%with ( %index ( %' ) insert into
#Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join ( select ( '%' + T . name
+ '%' + I . name + '%' ) as [filter] , schema_name ( T . schema_id ) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from
sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null ) S on C . definition collate DATABASE_DEFAULT
like S . [filter] collate DATABASE_DEFAULT
```

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: Test33

Exec.Count: 3

Avg.ElapseTime: 19.28 Seconds

QueryScript:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from

```
sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null ) S on C . definition collate DATABASE_DEFAULT
like S . [filter] collate DATABASE_DEFAULT
```

Database Backup Verification

The listed backup files are verified if any one of them are corrupted or protected by a password.

Database	Type	Position	Path	Date/Time	Size	Is Corrupted	Password Protected
AdventureWorks	D	1.0	\\Ai-DBA-DEMO\AiDBASharedDir20240424075026\AdventureWorks_FULL_20240424075023.bak	4/24/2024 7:50:26 PM	198.1 MB	No	No
AdventureWorks	D	1.0	\\Ai-DBA-DEMO\AiDBASharedDir20240423043903\AdventureWorks_FULL_20240423043903.bak	4/23/2024 4:39:03 AM	212.1 MB	No	No
AdventureWorks	D	1.0	https://aidbacachedata.blob.core.windows.net/demo/AdventureWorks_FULL_20240411023909.bak	4/11/2024 2:39:09 PM	206.3 MB	No	No
AdventureWorks	D	1.0	\\Ai-DBA-DEMO\AiDBASharedDir20240403061913\AdventureWorks_FULL_20240403061913.bak	4/3/2024 6:19:13 PM	212.1 MB	No	No
AdventureWorks	D	1.0	\\Ai-DBA-DEMO\AiDBASharedDir20240319120858\AdventureWorks_FULL_20240319120858.bak	3/19/2024 12:08:59 AM	198.1 MB	No	No
AdventureWorks	D	1.0	\\Ai-DBA-DEMO\AiDBASharedDir20240313031536\AdventureWorks_FULL_20240313031536.bak	3/13/2024 3:15:36 PM	212.1 MB	No	No
AdventureWorks	D	1.0	\\Ai-DBA-DEMO\AIDBABackup\AdventureWorks_FULL_20240212024822.bak	2/12/2024 2:48:22 AM	198.1 MB	No	No

Database	Type	Position	Path	Date/Time	Size	Is Corrupted	Password Protected
AdventureWorks	D	1.0	\\Ai-DBA-DEMO\AIDBA Backup\AdventureWorks_FULL_20240212024730.bak	2/12/2024 2:47:30 AM	198.1 MB	No	No
AdventureWorks	D	1.0	\\Ai-DBA-DEMO\AiDBA SharedDir20240115090459\AdventureWorks_FULL_20240115090459.bak	1/15/2024 9:04:59 AM	212.1 MB	No	No
AdventureWorks	D	1.0	\\Ai-DBA-DEMO\AiDBA SharedDir20231216083635\AdventureWorks_FULL_20231216083635.bak	12/16/2023 8:36:35 AM	206.1 MB	No	No
Advnew2019	D	1.0	\\Ai-DBA-DEMO\AiDBA SharedDir20241029043303\Advnew2019_FULL_20241029043303.bak	10/29/2024 4:33:10 PM	207.1 MB	No	No
Advnew2022	D	1.0	G:\BACKUPS\Advnew2022_FULL_20241206122841.bak	12/6/2024 12:28:42 AM	203.1 MB	No	No
Advnew2022	D	1.0	G:\BACKUPS\Advnew2022_FULL_20241002075712.bak	10/2/2024 7:57:12 PM	203.1 MB	No	No
Advnew2022	D	1.0	\\Ai-DBA-DEMO\AiDBA SharedDir20240319120936\Advnew2022_FULL_20240319120936.bak	3/19/2024 12:09:36 AM	207.1 MB	No	No
Advnew2022	D	1.0	\\Ai-DBA-DEMO\AiDBA SharedDir20240313024212\Advnew2022_FULL_20240313024212.bak	3/13/2024 2:42:12 AM	207.1 MB	No	No
ReportDB	D	1.0	\\Ai-DBA-DEMO\AiDBA SharedDir20240313055\Rep	3/13/2024 5:57:25 AM	2.6 MB	No	No

Database	Type	Position	Path	Date/Time	Size	Is Corrupted	Password Protected
			ortDB_FULL_2024031305723.bak				
ReportDB	D	1.0	\\Ai-DBA-DEMO\AiDBASharedDir20231218041052\ReportDB_FULL_20231218041052.bak	12/18/2023 4:10:52 AM	2.6 MB	No	No
ReportDB	D	1.0	\\Ai-DBA-DEMO\AiDBASharedDir20231218035323\ReportDB_FULL_20231218035323.bak	12/18/2023 3:53:23 AM	2.6 MB	No	No
SalesDB	D	1.0	https://aidbacachedata.blob.core.windows.net/demo/SalesDB_FULL_20231215070521.bak	12/15/2023 7:05:21 AM	2.8 MB	No	No

Database Warnings

The warning types stated below indicates severe performance degradation if the value is greater than 1,500 for each warning.

Warning	Occurrence
Sort Warnings	15.0

Database Missing Indexes

The listed tables below are involved in majority of the workloads, therefore some useful indexes are currently missing to support queries. The script of missing indexes are available in AI-DBA portal.

Object	Estimated Improvement	# of Indexes
[msdb].[dbo].[sysjobhistory]	73%	2.0

Database Unused Indexes

Unused indexes are able to slow down certain queries commands such as INSERT, DELETE and UPDATE. However, it can be used via SQL Server database engine internally. The following indexes are ready to be dropped.

Database	Table	Index	Type
Demo20240411	ContactType	AK_ContactType_Name	NONCLUSTERED
NewDB20241029	ContactType	AK_ContactType_Name	NONCLUSTERED

Login Credentials and Permissions

The listed login credentials are for future reference only. Each table shows up to 5,000 records.

Name	Status	Last Modification	Options	Last Password Reset	Bad Password Count
sa	Enabled	12/7/2024 5:35:10 PM	Policy Check	6/13/2023 1:17:03 AM	0.0
AIDBA	Enabled	6/15/2023 1:20:09 AM		6/15/2023 1:20:09 AM	0.0
System_Administrator_20240210	Enabled	2/11/2024 3:53:11 AM		12/29/2023 6:53:49 AM	0.0

Dropped Login	Dropped By	Dropped Through	Dropped At

Database	Schema	Object	Type	Grantee	Permission	Grantor
msdb	dbo	fn_sysutility_ucp_get_policy_violations	SQL_TABLE_VALUED_FUNCTION	UtilityCMRReader	SELECT	dbo

SQL Agent Objects

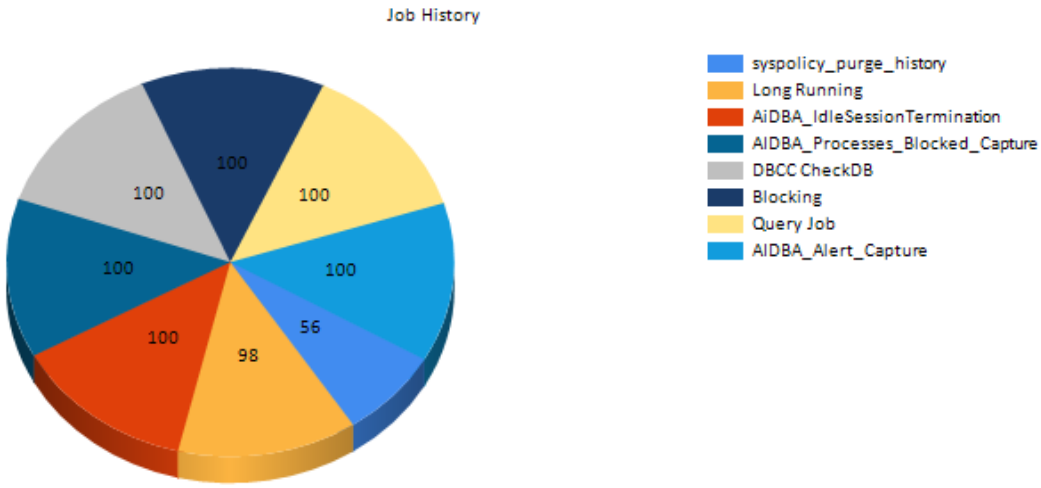
The listed agent objects are for the future reference only.

Alert	Status
Processes blocked	Enabled
Severity 17	Enabled
Severity 18	Enabled
Severity 19	Enabled
Severity 20	Enabled
Severity 21	Enabled
Severity 22	Enabled
Severity 23	Enabled
Severity 24	Enabled
Severity 25	Enabled

Job	Status	Description
AIDBA_Alert_Capture	Enabled	No description available.
AiDBA_IdleSessionTermination	Enabled	No description available.
AIDBA_Processes_Blocked_Capture	Enabled	No description available.
Blocking	Enabled	No description available.
DBCC CheckDB	Enabled	No description available.
Long Running	Enabled	No description available.
Query Job	Enabled	No description available.
syspolicy_purge_history	Enabled	No description available.

SQL Agent Jobs History

Often times it is needed to come with a list of durations per SQL Server Agent Job to trend the run times and order the results by date.



JobName	Status	RunDate	RUNTIME
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	9:45:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	9:30:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	9:15:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	9:00:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	8:45:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	8:30:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	8:15:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	8:00:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	7:45:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	7:30:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	7:15:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	7:00:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	6:45:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	6:30:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	6:15:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	6:00:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	5:45:00
syspolicy_purge_history	Succeeded	2025-02-14	2:00:00
Query Job	Failed	2025-02-14	18:02:40
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:02:39
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:02:37
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:02:35
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:02:34
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:02:32
Blocking	Failed	2025-02-14	18:02:32
Query Job	Failed	2025-02-14	18:02:32
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:02:30

JobName	Status	RunDate	RUNTIME
Blocking	Failed	2025-02-14	18:02:30
Query Job	Failed	2025-02-14	18:02:30
Query Job	Failed	2025-02-14	18:02:20
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:02:12
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:02:11
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:02:10
Query Job	Failed	2025-02-14	18:02:10
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:02:09
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:02:08
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:02:07
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:02:05
Blocking	Failed	2025-02-14	18:02:00
DBCC CheckDB	Succeeded	2025-02-14	18:02:00
Query Job	Failed	2025-02-14	18:02:00
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:53
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:52
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:51
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:50
Query Job	Failed	2025-02-14	18:01:50
Query Job	Failed	2025-02-14	18:01:40
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:36
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:34
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:32
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:31
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:30
Blocking	Failed	2025-02-14	18:01:30
DBCC CheckDB	Succeeded	2025-02-14	18:01:30
Query Job	Failed	2025-02-14	18:01:30
Query Job	Failed	2025-02-14	18:01:20
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:19
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:18
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:15
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:13
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:10
Query Job	Failed	2025-02-14	18:01:10
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:09
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:08
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:01:05
Blocking	Failed	2025-02-14	18:01:00
DBCC CheckDB	Succeeded	2025-02-14	18:01:00
Query Job	Failed	2025-02-14	18:01:00
Query Job	Failed	2025-02-14	18:00:50
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:49

JobName	Status	RunDate	RUNTIME
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:48
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:47
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:46
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:45
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:44
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:42
Query Job	Failed	2025-02-14	18:00:40
Blocking	Failed	2025-02-14	18:00:30
Query Job	Failed	2025-02-14	18:00:30
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:28
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:27
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:26
Query Job	Failed	2025-02-14	18:00:21
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:15
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:14
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:13
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:12
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:11
Query Job	Failed	2025-02-14	18:00:10
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:03
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:01
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	18:00:01
Blocking	Failed	2025-02-14	18:00:01
DBCC CheckDB	Succeeded	2025-02-14	18:00:01
Long Running	Succeeded	2025-02-14	18:00:01
Query Job	Failed	2025-02-14	18:00:01
AIDBA_Alert_Capture	Succeeded	2025-02-14	18:00:00
AiDBA_IdleSessionTermination	Succeeded	2025-02-14	18:00:00
Blocking	Failed	2025-02-14	18:00:00
DBCC CheckDB	Succeeded	2025-02-14	18:00:00
Long Running	Succeeded	2025-02-14	18:00:00
Query Job	Failed	2025-02-14	18:00:00
AIDBA_Alert_Capture	Succeeded	2025-02-14	17:59:59
AIDBA_Alert_Capture	Succeeded	2025-02-14	17:59:58
AIDBA_Alert_Capture	Succeeded	2025-02-14	17:59:57
Query Job	Failed	2025-02-14	17:59:50
Query Job	Failed	2025-02-14	17:59:40
Blocking	Failed	2025-02-14	17:59:30
DBCC CheckDB	Succeeded	2025-02-14	17:59:30
Query Job	Failed	2025-02-14	17:59:30
Query Job	Failed	2025-02-14	17:59:20
Query Job	Failed	2025-02-14	17:59:10
Blocking	Failed	2025-02-14	17:59:00

JobName	Status	RunDate	RUNTIME
DBCC CheckDB	Succeeded	2025-02-14	17:59:00
Query Job	Failed	2025-02-14	17:59:00
Query Job	Failed	2025-02-14	17:58:50
Query Job	Failed	2025-02-14	17:58:40
Blocking	Failed	2025-02-14	17:58:30
DBCC CheckDB	Succeeded	2025-02-14	17:58:30
Query Job	Failed	2025-02-14	17:58:30
Query Job	Failed	2025-02-14	17:58:20
Query Job	Failed	2025-02-14	17:58:10
Blocking	Failed	2025-02-14	17:58:00
DBCC CheckDB	Succeeded	2025-02-14	17:58:00
Query Job	Failed	2025-02-14	17:58:00
Query Job	Failed	2025-02-14	17:57:50
Query Job	Failed	2025-02-14	17:57:40
Blocking	Failed	2025-02-14	17:57:30
DBCC CheckDB	Succeeded	2025-02-14	17:57:30
Query Job	Failed	2025-02-14	17:57:30
Query Job	Failed	2025-02-14	17:57:20
Query Job	Failed	2025-02-14	17:57:10
Blocking	Failed	2025-02-14	17:57:00
DBCC CheckDB	Succeeded	2025-02-14	17:57:00
Query Job	Failed	2025-02-14	17:57:00
Query Job	Failed	2025-02-14	17:56:50
Query Job	Failed	2025-02-14	17:56:41
Query Job	Failed	2025-02-14	17:56:40
DBCC CheckDB	Succeeded	2025-02-14	17:56:31
Query Job	Failed	2025-02-14	17:56:31
Blocking	Failed	2025-02-14	17:56:30
DBCC CheckDB	Succeeded	2025-02-14	17:56:30
Query Job	Failed	2025-02-14	17:56:30
Query Job	Failed	2025-02-14	17:56:20
Query Job	Failed	2025-02-14	17:56:10
Blocking	Failed	2025-02-14	17:56:00
DBCC CheckDB	Succeeded	2025-02-14	17:56:00
Query Job	Failed	2025-02-14	17:56:00
Query Job	Failed	2025-02-14	17:55:50
Query Job	Failed	2025-02-14	17:55:41

Windows and SQL Server Severe Alerts and Errors

Critical Alerts: A critical level alert should be a worst-case scenario – there is an issue that requires attention. They are designed to be reactive alerts, meaning someone should react to these alerts as soon as possible.

Error Alerts: An error level alert is less severe and should convey that something is wrong or isn't behaving normally, but there isn't necessarily a specific action that has to be taken. You should know about these scenarios, but they shouldn't have the same sense of urgency as a critical alert. Error alerts are designed to be more proactive than critical alerts, but you may want to know about them sooner and they may be treated more as reactive alerts depending on your use case.

Warning Alerts: A warning alert indicates that there is something you should be aware of, but it may not be causing a problem yet. Warning alerts are designed to usually be proactive alerts, meaning we're notifying you that there may be a future problem so that you can avoid the problem all together.

EventID	Type	Source	Message	Raised At

Severe Alerts and Error Summary

Appendix A: Query Performance Comparison

The table below indicates the query's elapse time in miliseconds as per health checks. Blank value indicates the query did not meet long running queries criteria.

Database	Query Id	2025-02-12	2025-02-13	2025-02-14	2025-02-15	2025-02-16	2025-02-17
Adv2022ShalevSoft	0	569,415	596,530	135,575	189,320	431,940	355,920
AdventureWorks	0			12,920			
Advnew2022Moved	1			600	3,520	1,840	960
AdventureWorks	5			2,380			
AdvNew2022Restored	9					350	1,140
AdventureWorks	20			860			
AdventureWorks	21			540			

Database	Query Id	2025-02-12	2025-02-13	2025-02-14	2025-02-15	2025-02-16	2025-02-17
AdvDest20240317	22				28		60
AIDBAADV2	22			500	1,800		
AdventureWorks	23			390			
AdvNew2022Restored3	25	420	3,080	1,010	2,810	1,260	
NewDB20241029	25				1,190	1,260	
Adv2022ShalevSoft	28			100			350
AdventureWorks	31			590			
newdbemo3099	38		520	1,820	5,170	400	1,460
Advnew2022	39			827	3,880	1,890	1,780
AdventureWorks	42			2,700			
Test33	43						100
AdventureWorks	47			295,420,200			
Adv2022ShalevSoft	48				18		10
AdventureWorks	48			560			
Demo202404111	52			500	1,800		350
Adv2022ShalevSoft	53						60
Advnew2022	55	7,770	8,140	1,850			300
AdvNewDB2022Portal	59			1,200	6,020	2,200	1,360
AdventureWorks	63			45,530			
Test33	64				30		0
Adv2022ShalevSoft	65						51
AdvDest20240317	68						62
AdventureWorks	69			12,040			
AdvNew2022Restored3	69			500	3,330	1,620	
AdvNew2022Restored3	81	1,980	560	1,610	5,140	1,890	1,530
Advnew2022Moved	82						91

Database	Query Id	2025-02-12	2025-02-13	2025-02-14	2025-02-15	2025-02-16	2025-02-17
AdventureWorks	87			3,390			
Advnew2022	92						81
Test33	94	6,720	7,040	1,600			
AdvDest20240317	97						350
AdventureWorks	99			3,450	12,420		
Test33	101						94
AIDBAADV2	103	2,160		950	4,780	1,890	1,530
AdventureWorks	104			760			
NewDB20241029	106		520	1,220	4,050	2,020	1,360
Advnew2022Moved	110						90
newdbemo3099	110	2,700			490		
Test33	111	573,720	601,040	136,600	194,031	443,509	349,236
AdventureWorks	112			490			
AdvDest20240317	113						80
AIDBAADV2	114	2,340				350	840
Advnew2022Moved	124	674,940	707,080	160,700	187,850	430,790	351,080
AdvDest20240317	127			750	3,890	1,610	1,240
AdventureWorks	129			100,740			
AdventureWorks	133						
AdventureWorks	134			390			
Advnew2022Moved	139	2,940	3,080	700			350
AdvNewDB2022Portal	139	420	2,520	550	1,980		
Advnew2022Moved	145				48		20
Advnew2022	146	3,075	2,250	620	3,330	1,995	900
AdventureWorks	149	3,150	4,500	720			350
AdventureWorks	153			8,350			
Advnew2022	153				0		0
Adv2022Shale	158	3,135	3,238	1,535	4,570	2,480	1,550

Database	Query Id	2025-02-12	2025-02-13	2025-02-14	2025-02-15	2025-02-16	2025-02-17
vSoft							
AdventureWorks	161			380			
Advnew2022	165	654,570	685,740	155,850	188,750	433,320	349,600
Advnew2022Moved	168			110	1,190	1,635	900
AdventureWorks	171			470			
AdvDest20240317	173			550	1,980		
Advnew2022Moved	174	420	3,132	1,265	2,290	350	1,190
AdvNew2022Restored2	174	360	2,160				300
Demo20240411	178		540	1,190	2,830	350	1,290
Test33	179			663	3,510	2,045	1,420
AdventureWorks	182			400			
AdventureWorks	184			540			
AdvNew2022Restored2	184						
ReportDB_Copy	186						
Adv2022ShalevSoft	187			630	1,800		
ReportDB678	188						
Demo20240411	189						
AdvNew2022Restored	193	2,340		1,300	5,870	1,660	1,310
AdvNew2022Restored2	195	2,160	620	1,320	4,050	2,020	1,310
AdvNewDB2022Portal	196	2,520			2,210	2,340	
AdventureWorks	197			8,650	31,140		
Demo20240411	197				1,360	1,440	350
AdventureWorks	198			460			
newdbemo3099	199						
AdventureWorks	201			44,880,736			
AdvDest20240317	203	3,120	3,052	1,565	4,770	2,020	1,410

Database	Query Id	2025-02-12	2025-02-13	2025-02-14	2025-02-15	2025-02-16	2025-02-17
AdventureWorks	210			870			
AdventureWorks	215	420	3,080	1,410	4,590	2,020	1,410
AdventureWorks	216			1,290			
AdventureWorks	217			800	2,880		
AdvNew2022Restored	219				1,530	1,620	
Test33	223			760	3,870	2,020	960
AdventureWorks	228			420			
Advnew2022	228						80
ReportDB678	228						
AdvDest20240317	231	630,105	660,110	150,025	191,000	435,620	357,680
AdventureWorks	237			7,380			
AIDBAADV2	238	2,370	2,900	560			
AdventureWorks	240			4,070			
Test33	243	375	2,250	113	1,360	1,815	900
AdvDest20240317	244			110	1,190	1,635	900
NewDB20241029	245	360	2,160		1,190	1,260	
AdventureWorks	246			500	2,290	350	840
newdbemo3099	246			750	2,700	350	840
AdventureWorks	252			550			

Database: Adv2022ShalevSoft

QueryId: 0

Script:

```
( @0 int,@3 varchar ( 8000 ) ,@4 varchar ( 8000 ) ,@5 varchar ( 8000 ) ,@6 varchar ( 8000 ) ,@9 varchar ( 8000 ) ,@10 varchar ( 8000 ) ,@11 varchar ( 8000 ) ) with CTE as ( select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace ( lower ( Replace ( definition , char ( 13 ) + Char ( 10 ) , @3 ) ) , @4 , @5 ) like '%with ( %index ( %' union all select Object_name ( T . objectid ) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text ( qs . Sql_handle ) T where CP . objtype = @6 and Replace ( lower ( Replace ( T . text , char ( 13 ) + Char ( 10 ) , @9 ) ) , @10 , @11 ) like '%with ( %index ( %' ) insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join ( select ( '%' + T . name + '%' + I . name + '%' ) as [filter] , schema_name ( T . schema_id ) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null ) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT
```

Database: AdventureWorks

QueryId: 0

Script:

```
/*AI-DBA*/
SELECT
[CreditCardApprovalCode] AS [VALUE],
COUNT ( * )  [ROW_COUNT]
from [AdventureWorks].[Sales].[SalesOrderHeader]
WHERE [CreditCardApprovalCode] IS NOT null
GROUP BY [CreditCardApprovalCode]
ORDER BY [VALUE] desc
```

Database: Advnew2022Moved

QueryId: 1

Script:

```
( @0 varchar ( 8000 ) ) select db_id  (  )  as database_id , sm . [is_inlineable] as InlineableScalarCount , sm . [inline_type] as InlineType , COUNT_BIG  (  * )  as ScalarCount , COUNT_BIG  (  case when sm . [definition] like '%getdate%' or sm . [definition] like '%getutcdate%' or sm . [definition] like '%sysdatetime%' or sm . [definition] like '%sysutcdatetime%' or sm . [definition] like '%sysdatetimeoffset%' or sm . [definition] like '%CURRENT_TIMESTAMP%' then 1 end  )  as ScalarCountWithDate from [sys] . [objects] o inner join [sys] . [sql_modules] sm on o . [object_id]  =  sm . [object_id] where o . [type]  =  @0 group by sm . [is_inlineable] , sm . [inline_type]
```

Database: AdventureWorks

QueryId: 5

Script:

```
/*AI-DBA*/
SELECT
[CarrierTrackingNumber] AS [VALUE],
COUNT ( * )  [ROW_COUNT]
from [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE [CarrierTrackingNumber] IS NOT null
GROUP BY [CarrierTrackingNumber]
ORDER BY [VALUE] desc
```

Database: AdvNew2022Restored

QueryId: 9

Script:

```
SELECT  db_id ( )  AS database_id,  o.[type] AS object_type,  i.[type] AS index_type,  p.[data_compression],  COUNT_BIG ( DISTINCT p.[object_id] )  AS NumTables,  COUNT_BIG ( DISTINCT CAST ( p.[object_id] AS VARCHAR ( 30 ) )  +  '|' + CAST ( p.[index_id] AS VARCHAR ( 10 ) ) )  AS NumIndexes,  ISNULL ( px.[IsPartitioned], 0 )  AS IsPartitioned,  IIF ( px.[IsPartitioned]  =  1, COUNT_BIG ( 1 ) , 0 )  NumPartitions,  SUM ( p.[rows] )  NumRows  FROM sys.partitions p  INNER JOIN sys.objects o  ON o.[object_id]  =  p.[object_id]  INNER JOIN sys.indexes i  ON i.[object_id]  =  p.[object_id]  AND i.[index_id]  =  p.[index_id]  OUTER APPLY  ( SELECT  x.[object_id], 1 AS [IsPartitioned]  FROM sys.partitions x  WHERE x.[object_id]  =  p.[object_id]  GROUP by  x.[object_id]  HAVING MAX ( x.partition_number )  >  1 )  px  WHERE o.[type]  NOT IN  ( 'S', 'IT' )  GROUP BY  o.[type]  ,i.[type]  ,p.[data_compression]  ,px.[IsPartitioned]
```

Database: AdventureWorks

QueryId: 20

Script:

```
/*AI-DBA*/
SELECT CONVERT ( DECIMAL,MIN ( [UnitPrice] ) ) AS [MIN],
CONVERT ( DECIMAL,MAX ( [UnitPrice] ) ) AS [MAX],
CONVERT ( DECIMAL,AVG ( [UnitPrice] ) ) AS [AVG],
CONVERT ( DECIMAL,STDEV ( [UnitPrice] ) ) AS [STDDEV],
CONVERT ( DECIMAL,VAR ( [UnitPrice] ) ) AS [VAR]
FROM [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE [UnitPrice] IS NOT NULL
```

Database: AdventureWorks

QueryId: 21

Script:

```
/*AI-DBA*/
SELECT COUNT ( 1 ) AS [NONEMPTY]
FROM [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE LEN ( CONVERT ( NVARCHAR ( MAX ) ,[SalesOrderDetailID] ) ) > 0 AND LEN ( CONVERT ( NVARCHAR ( MAX )
,[SalesOrderDetailID] ) ) IS NOT NULL ;
```

Database: AdvDest20240317

QueryId: 22

Script:

```
( @0 int ) insert into #temp select db_name ( ) as [DB_Name] , object_name ( IU . object_id ) as [ObjName] , I . name as
[IndexName] , I . type_desc as [IndexType] , IU . user_seeks , IU . user_scans , IU . user_lookups , case when IU . user_seeks = 0
then 0 else ( IU . user_seeks * 100 ) / ( IU . user_seeks + IU . user_scans + IU . user_lookups ) end as [User_Seeks_Pct] , case
when IU . user_scans = 0 then 0 else ( IU . user_scans * 100 ) / ( IU . user_seeks + IU . user_scans + IU . user_lookups ) end
as [User_Scans_Pct] , case when IU . user_lookups = 0 then 0 else ( IU . user_lookups * 100 ) / ( IU . user_seeks + IU .
user_scans + IU . user_lookups ) end as [User_Lookups_Pct] from sys . dm_db_index_usage_stats IU inner join sys . indexes I on I .
index_id = IU . index_id and I . object_id = IU . object_id inner join sys . all_objects AO on AO . object_id = IU . object_id and AO .
is_ms_shipped = @0
```

Database: AIDBAADV2

QueryId: 22

Script:

```
SELECT db_id ( ) AS database_id, o.[type] as ModuleType, COUNT_BIG ( * ) as ModuleCount FROM sys.objects AS o WITH (
nolock ) WHERE o.type in ( 'AF', 'F', 'FN', 'FS', 'FT', 'IF', 'P', 'PC', 'TA', 'TF', 'TR', 'X', 'C', 'D', 'PG', 'SN', 'SO', 'SQ', 'TT', 'UQ', 'V' )
GROUP BY o.[type]
```

Database: AdventureWorks

QueryId: 23

Script:

```
/*AI-DBA*/
SELECT CONVERT ( DECIMAL,MIN ( [LineTotal] ) ) AS [MIN],
CONVERT ( DECIMAL,MAX ( [LineTotal] ) ) AS [MAX],
CONVERT ( DECIMAL,AVG ( [LineTotal] ) ) AS [AVG],
CONVERT ( DECIMAL,STDEV ( [LineTotal] ) ) AS [STDDEV],
CONVERT ( DECIMAL,VAR ( [LineTotal] ) ) AS [VAR]
FROM [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE [LineTotal] IS NOT NULL
```

Database: AdvNew2022Restored3

QueryId: 25

Script:

```
SELECT db_id ( ) AS database_id, o.[type] AS object_type, i.[type] AS index_type, p.[data_compression], COUNT_BIG ( DISTINCT p.[object_id] ) AS NumTables, COUNT_BIG ( DISTINCT CAST ( p.[object_id] AS VARCHAR ( 30 ) ) + '|' + CAST ( p.[index_id] AS VARCHAR ( 10 ) ) ) AS NumIndexes, ISNULL ( px.[IsPartitioned], 0 ) AS IsPartitioned, IIF ( px.[IsPartitioned] = 1, COUNT_BIG ( 1 ) , 0 ) NumPartitions, SUM ( p.[rows] ) NumRows FROM sys.partitions p INNER JOIN sys.objects o ON o.[object_id] = p.[object_id] INNER JOIN sys.indexes i ON i.[object_id] = p.[object_id] AND i.[index_id] = p.[index_id] OUTER APPLY ( SELECT x.[object_id], 1 AS [IsPartitioned] FROM sys.partitions x WHERE x.[object_id] = p.[object_id] GROUP by x.[object_id] HAVING MAX ( x.partition_number ) > 1 ) px WHERE o.[type] NOT IN ( 'S', 'IT' ) GROUP BY o.[type] ,i.[type] ,p.[data_compression] ,px.[IsPartitioned]
```

Database: NewDB20241029

QueryId: 25

Script:

```
SELECT db_id ( ) as database_id, sm.[is_inlineable] AS InlineableScalarCount, sm.[inline_type] AS InlineType, COUNT_BIG ( * ) AS ScalarCount, COUNT_BIG ( CASE WHEN sm.[definition] LIKE '%getdate%' OR sm.[definition] LIKE '%getutcdate%' OR sm.[definition] LIKE '%sysdatetime%' OR sm.[definition] LIKE '%sysutcdatetime%' OR sm.[definition] LIKE '%sysdatetimeoffset%' OR sm.[definition] LIKE '%CURRENT_TIMESTAMP%' THEN 1 END ) AS ScalarCountWithDate FROM [sys].[objects] o INNER JOIN [sys].[sql_modules] sm ON o.[object_id] = sm.[object_id] WHERE o.[type] = 'FN' GROUP BY sm.[is_inlineable], sm.[inline_type]
```

Database: Adv2022ShalevSoft

QueryId: 28

Script:

```
( @0 int ) insert into #sqls select DB_ID ( ) , [definition] from sys . all_sql_modules where object_id > @0 and ( [definition] like '%cast%' or [Definition] like '%convert%' )
```

Database: AdventureWorks

QueryId: 31

Script:

```
/*AI-DBA*/
SELECT COUNT ( 1 ) AS [NONEMPTY]
FROM [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE LEN ( CONVERT ( NVARCHAR ( MAX ) ,[ModifiedDate] ) ) > 0 AND LEN ( CONVERT ( NVARCHAR ( MAX ) ,[ModifiedDate] ) ) IS NOT NULL ;
```

Database: newdbemo3099

QueryId: 38

Script:

```
SELECT db_id ( ) AS database_id, c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.is_filestream, c.encryption_type, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END AS is_user, COUNT_BIG ( * ) AS [ColCount], CASE WHEN c.collation_name IS NULL THEN CONVERT ( VARCHAR ( 128 ) , SERVERPROPERTY ( 'Collation' ) ) ELSE c.collation_name END AS collation_name, AVG ( c.max_length ) AS avg_max_length FROM sys.columns c WITH ( NOLOCK ) LEFT OUTER JOIN sys.objects o WITH ( NOLOCK ) ON o.object_id = c.object_id AND o.type = 'U' GROUP BY c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.encryption_type, c.is_filestream, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END, CASE WHEN c.collation_name IS NULL THEN CONVERT ( VARCHAR ( 128 ) , SERVERPROPERTY ( 'Collation' ) ) ELSE c.collation_name END
```

Database: Advnew2022

QueryId: 39

Script:

```
( @0 varchar ( 8000 ) ) select db_id ( ) as database_id , c . system_type_id , c . user_type_id , c . is_sparse , c . is_column_set , c . is_filestream , c . encryption_type , case when o . object_id is not null then 1 else 0 end as is_user , COUNT_BIG ( * ) as [ColCount]
```

, case when c . collation_name is null then convert (VARCHAR (128) , SERVERPROPERTY ('Collation')) else c . collation_name end as collation_name , AVG (c . max_length) as avg_max_length from sys . columns c with (NOLOCK) left outer join sys . objects o with (NOLOCK) on o . object_id = c . object_id and o . type = @0 group by c . system_type_id , c . user_type_id , c . is_sparse , c . is_column_set , c . encryption_type , c . is_filestream , case when o . object_id is not null then 1 else 0 end , case when c . collation_name is null then convert (VARCHAR (128) , SERVERPROPERTY ('Collation')) else c . collation_name end

---

Database: Advnew2022

QueryId: 39

Script:
(@0 varchar (8000) ,@1 varchar (8000)) select db_id ( ) as database_id , o . [type] as object_type , i . [type] as index_type , p . [data_compression] , COUNT_BIG (distinct p . [object_id]) as NumTables , COUNT_BIG (distinct CAST (p . [object_id] as VARCHAR (30)) + '|' + CAST (p . [index_id] as VARCHAR (10))) as NumIndexes , ISNULL (px . [IsPartitioned] , 0) as IsPartitioned , IIF (px . [IsPartitioned] = 1 , COUNT_BIG (1) , 0) NumPartitions , SUM (p . [rows]) NumRows from sys . partitions p inner join sys . objects o on o . [object_id] = p . [object_id] inner join sys . indexes i on i . [object_id] = p . [object_id] and i . [index_id] = p . [index_id] outer APPLY (select x . [object_id] , 1 as [IsPartitioned] from sys . partitions x where x . [object_id] = p . [object_id] group by x . [object_id] having MAX (x . partition_number) > 1) px where o . [type] not in (@0 , @1) group by o . [type] , i . [type] , p . [data_compression] , px . [IsPartitioned]

Database: AdventureWorks

QueryId: 42

Script:

```
/*AI-DBA*/
SELECT
[PurchaseOrderNumber] AS [VALUE],
COUNT ( * ) [ROW_COUNT]
from [AdventureWorks].[Sales].[SalesOrderHeader]
WHERE [PurchaseOrderNumber] IS NOT null
GROUP BY [PurchaseOrderNumber]
ORDER BY [VALUE] desc
```

Database: Test33

QueryId: 43

Script:
(@0 int,@1 varchar (8000)) insert into #temp select DB_Name ( ) [DBName] , case when T . system_type_id in (35 , 99 , 167 , 175 , 239 , 231) then 'Wrong' else 'Right' end as [Design] , Count (*) as [Count] from sys . sysindexkeys IK inner join sys . all_objects AO on AO . object_id = IK . id and AO . is_ms_shipped = @0 and AO . type = @1 inner join sys . indexes I on I . index_id = IK . indid and I . object_id = IK . id cross Apply (select * from sys . all_columns AC where AC . column_id = IK . colid and AC . object_id = IK . id) C inner join sys . types T on T . system_type_id = C . system_type_id group by case when T . system_type_id in (35 , 99 , 167 , 175 , 239 , 231) then 'Wrong' else 'Right' end

---

Database: AdventureWorks

QueryId: 47

Script:

```
select top ( 1000000 ) * from sales.SalesOrderHeader h
cross join sales.SalesOrderDetail d
```

Database: Adv2022ShalevSoft

QueryId: 48

Script:
(@0 int) insert into #temp select db_name ( ) as [DB_Name] , object_name (IU . object_id) as [ObjName] , I . name as

[IndexName] , I . type_desc as [IndexType] , IU . user_seeks , IU . user_scans , IU . user_lookups , case when IU . user_seeks = 0 then 0 else (IU . user_seeks * 100) / (IU . user_seeks + IU . user_scans + IU . user_lookups) end as [User_Seeks_Pct] , case when IU . user_scans = 0 then 0 else (IU . user_scans * 100) / (IU . user_seeks + IU . user_scans + IU . user_lookups) end as [User_Scans_Pct] , case when IU . user_lookups = 0 then 0 else (IU . user_lookups * 100) / (IU . user_seeks + IU . user_scans + IU . user_lookups) end as [User_Lookups_Pct] from sys . dm_db_index_usage_stats IU inner join sys . indexes I on I . index_id = IU . index_id and I . object_id = IU . object_id inner join sys . all_objects AO on AO . object_id = IU . object_id and AO . is_ms_shipped = @0

Database: AdventureWorks

QueryId: 48

Script:

```
/*AI-DBA*/
SELECT COUNT ( 1 ) AS [EMPTY]
FROM [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE LEN ( CONVERT ( NVARCHAR ( MAX ) ,[SalesOrderDetailID] ) ) = 0 AND LEN ( CONVERT ( NVARCHAR ( MAX ) ,[SalesOrderDetailID] ) ) IS NOT NULL ;
```

Database: Demo20240411

QueryId: 52

Script:

```
SELECT db_id ( ) as database_id, sm.[is_inlineable] AS InlineableScalarCount, sm.[inline_type] AS InlineType, COUNT_BIG ( * ) AS ScalarCount, COUNT_BIG ( CASE WHEN sm.[definition] LIKE '%getdate%' OR sm.[definition] LIKE '%getutcdate%' OR sm.[definition] LIKE '%sysdatetime%' OR sm.[definition] LIKE '%sysutcdatetime%' OR sm.[definition] LIKE '%sysdatetimeoffset%' OR sm.[definition] LIKE '%CURRENT_TIMESTAMP%' THEN 1 END ) AS ScalarCountWithDate FROM [sys].[objects] o INNER JOIN [sys].[sql_modules] sm ON o.[object_id] = sm.[object_id] WHERE o.[type] = 'FN' GROUP BY sm.[is_inlineable], sm.[inline_type]
```

Database: Adv2022ShalevSoft

QueryId: 53

Script:

```
( @0 int,@1 varchar ( 8000 ) ) insert into #temp select DB_Name ( ) [DBName] , case when T . system_type_id in ( 35 , 99 , 167 , 175 , 239 , 231 ) then 'Wrong' else 'Right' end as [Design] , Count ( * ) as [Count] from sys . sysindexkeys IK inner join sys . all_objects AO on AO . object_id = IK . id and AO . is_ms_shipped = @0 and AO . type = @1 inner join sys . indexes I on I . index_id = IK . indid and I . object_id = IK . id cross Apply ( select * from sys . all_columns AC where AC . column_id = IK . colid and AC . object_id = IK . id ) C inner join sys . types T on T . system_type_id = C . system_type_id group by case when T . system_type_id in ( 35 , 99 , 167 , 175 , 239 , 231 ) then 'Wrong' else 'Right' end
```

Database: Advnew2022

QueryId: 55

Script:

```
( @0 int ) insert into #sqls select DB_ID ( ) , [definition] from sys . all_sql_modules where object_id > @0 and ( [definition] like '%cast%' or [Definition] like '%convert%' )
```

Database: AdvNewDB2022Portal

QueryId: 59

Script:

```
SELECT db_id ( ) AS database_id, c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.is_filestream, c.encryption_type, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END AS is_user, COUNT_BIG ( * ) AS [ColCount], CASE WHEN c.collation_name IS NULL THEN CONVERT ( VARCHAR ( 128 ) , SERVERPROPERTY ( 'Collation' ) ) ELSE c.collation_name END AS collation_name, AVG ( c.max_length ) AS avg_max_length FROM sys.columns c WITH ( NOLOCK ) LEFT OUTER JOIN sys.objects o WITH ( NOLOCK ) ON o.object_id = c.object_id AND o.type = 'U' GROUP BY c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set,
```

c.encryption_type, c.is_filestream, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END, CASE WHEN c.collation_name IS NULL THEN CONVERT (VARCHAR (128) , SERVERPROPERTY ('Collation')) ELSE c.collation_name END

Database: AdventureWorks

QueryId: 63

Script:

```
/*AI-DBA*/
SELECT
[SalesOrderDetailID] AS [VALUE],
COUNT ( * ) [ROW_COUNT]
from [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE [SalesOrderDetailID] IS NOT null
GROUP BY [SalesOrderDetailID]
ORDER BY [VALUE] desc
```

Database: Test33

QueryId: 64

Script:
(@0 int) insert into #temp select db_name ( ) as [DB_Name] , object_name (IU . object_id) as [ObjName] , I . name as [IndexName] , I . type_desc as [IndexType] , IU . user_seeks , IU . user_scans , IU . user_lookups , case when IU . user_seeks = 0 then 0 else (IU . user_seeks * 100) / (IU . user_seeks + IU . user_scans + IU . user_lookups) end as [User_Seeks_Pct] , case when IU . user_scans = 0 then 0 else (IU . user_scans * 100) / (IU . user_seeks + IU . user_scans + IU . user_lookups) end as [User_Scans_Pct] , case when IU . user_lookups = 0 then 0 else (IU . user_lookups * 100) / (IU . user_seeks + IU . user_scans + IU . user_lookups) end as [User_Lookups_Pct] from sys . dm_db_index_usage_stats IU inner join sys . indexes I on I . index_id = IU . index_id and I . object_id = IU . object_id inner join sys . all_objects AO on AO . object_id = IU . object_id and AO . is_ms_shipped = @0

Database: Adv2022ShalevSoft

QueryId: 65

Script:
(@0 int,@1 varchar (8000)) insert into #temp select db_name ( ) [DBName] , Object_name (IK . id) [TableName] , schema_name (AO . schema_id) as [SchemaName] , I . name [IndexName] , C . name [KeyName] , T . name [DataType] , case when T . system_type_id in (35 , 99 , 167 , 175 , 239 , 231) then 'Wrong' else 'Right' end as [Design] from sys . sysindexkeys IK inner join sys . all_objects AO on AO . object_id = IK . id and AO . is_ms_shipped = @0 and AO . type = @1 inner join sys . indexes I on I . index_id = IK . indid and I . object_id = IK . id cross Apply (select * from sys . all_columns AC where AC . column_id = IK . colid and AC . object_id = IK . id) C inner join sys . types T on T . system_type_id = C . system_type_id order by Object_name (IK . id) asc , I . name asc , C . name asc

---

Database: AdvDest20240317

QueryId: 68

Script:
(@0 int,@1 varchar (8000)) insert into #temp select db_name ( ) [DBName] , Object_name (IK . id) [TableName] , schema_name (AO . schema_id) as [SchemaName] , I . name [IndexName] , C . name [KeyName] , T . name [DataType] , case when T . system_type_id in (35 , 99 , 167 , 175 , 239 , 231) then 'Wrong' else 'Right' end as [Design] from sys . sysindexkeys IK inner join sys . all_objects AO on AO . object_id = IK . id and AO . is_ms_shipped = @0 and AO . type = @1 inner join sys . indexes I on I . index_id = IK . indid and I . object_id = IK . id cross Apply (select * from sys . all_columns AC where AC . column_id = IK . colid and AC . object_id = IK . id) C inner join sys . types T on T . system_type_id = C . system_type_id order by Object_name (IK . id) asc , I . name asc , C . name asc

---

Database: AdventureWorks

QueryId: 69

Script:

```
/*AI-DBA*/
SELECT
[AccountNumber] AS [VALUE],
COUNT ( * )  [ROW_COUNT]
from [AdventureWorks].[Sales].[SalesOrderHeader]
WHERE [AccountNumber] IS NOT null
GROUP BY [AccountNumber]
ORDER BY [VALUE] desc
```

Database: AdvNew2022Restored3

QueryId: 69

Script:

```
SELECT db_id ( ) as database_id, sm.[is_inlineable] AS InlineableScalarCount, sm.[inline_type] AS InlineType, COUNT_BIG ( * ) AS ScalarCount, COUNT_BIG ( CASE WHEN sm.[definition] LIKE '%getdate%' OR sm.[definition] LIKE '%getutcdate%' OR sm.[definition] LIKE '%sysdatetime%' OR sm.[definition] LIKE '%sysutcdatetime%' OR sm.[definition] LIKE '%sysdatetimeoffset%' OR sm.[definition] LIKE '%CURRENT_TIMESTAMP%' THEN 1 END ) AS ScalarCountWithDate FROM [sys].[objects] o INNER JOIN [sys].[sql_modules] sm ON o.[object_id] = sm.[object_id] WHERE o.[type] = 'FN' GROUP BY sm.[is_inlineable], sm.[inline_type]
```

Database: AdvNew2022Restored3

QueryId: 81

Script:

```
SELECT db_id ( ) AS database_id, c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.is_filestream, c.encryption_type, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END AS is_user, COUNT_BIG ( * ) AS [ColCount], CASE WHEN c.collation_name IS NULL THEN CONVERT ( VARCHAR ( 128 ) , SERVERPROPERTY ( 'Collation' ) ) ELSE c.collation_name END AS collation_name, AVG ( c.max_length ) AS avg_max_length FROM sys.columns c WITH ( NOLOCK ) LEFT OUTER JOIN sys.objects o WITH ( NOLOCK ) ON o.object_id = c.object_id AND o.type = 'U' GROUP BY c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.encryption_type, c.is_filestream, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END, CASE WHEN c.collation_name IS NULL THEN CONVERT ( VARCHAR ( 128 ) , SERVERPROPERTY ( 'Collation' ) ) ELSE c.collation_name END
```

Database: Advnew2022Moved

QueryId: 82

Script:

```
( @0 int,@1 varchar ( 8000 ) ) insert into #temp select db_name ( ) [DBName] , Object_name ( IK . id ) [TableName] , schema_name ( AO . schema_id ) as [SchemaName] , I . name [IndexName] , C . name [KeyName] , T . name [DataType] , case when T . system_type_id in ( 35 , 99 , 167 , 175 , 239 , 231 ) then 'Wrong' else 'Right' end as [Design] from sys . sysindexkeys IK inner join sys . all_objects AO on AO . object_id = IK . id and AO . is_ms_shipped = @0 and AO . type = @1 inner join sys . indexes I on I . index_id = IK . indid and I . object_id = IK . id cross Apply ( select * from sys . all_columns AC where AC . column_id = IK . colid and AC . object_id = IK . id ) C inner join sys . types T on T . system_type_id = C . system_type_id order by Object_name ( IK . id ) asc , I . name asc , C . name asc
```

Database: AdventureWorks

QueryId: 87

Script:

```
/*AI-DBA*/
SELECT
[CustomerID] AS [VALUE],
COUNT ( * )  [ROW_COUNT]
from [AdventureWorks].[Sales].[SalesOrderHeader]
```

```
WHERE [CustomerID] IS NOT null
GROUP BY [CustomerID]
ORDER BY [VALUE] desc
```

Database: Advnew2022

QueryId: 92

Script:

```
( @0 int,@1 varchar ( 8000 ) ) insert into #temp select db_name ( ) [DBName] , Object_name ( IK . id ) [TableName] , schema_name ( AO . schema_id ) as [SchemaName] , I . name [IndexName] , C . name [KeyName] , T . name [DataType] , case when T . system_type_id in ( 35 , 99 , 167 , 175 , 239 , 231 ) then 'Wrong' else 'Right' end as [Design] from sys . sysindexkeys IK inner join sys . all_objects AO on AO . object_id = IK . id and AO . is_ms_shipped = @0 and AO . type = @1 inner join sys . indexes I on I . index_id = IK . indid and I . object_id = IK . id cross Apply ( select * from sys . all_columns AC where AC . column_id = IK . colid and AC . object_id = IK . id ) C inner join sys . types T on T . system_type_id = C . system_type_id order by Object_name ( IK . id ) asc , I . name asc , C . name asc
```

Database: Test33

QueryId: 94

Script:

```
( @0 int ) insert into #sqls select DB_ID ( ) , [definition] from sys . all_sql_modules where object_id > @0 and ( [definition] like '%cast%' or [Definition] like '%convert%' )
```

Database: AdvDest20240317

QueryId: 97

Script:

```
( @0 int ) insert into #sqls select DB_ID ( ) , [definition] from sys . all_sql_modules where object_id > @0 and ( [definition] like '%cast%' or [Definition] like '%convert%' )
```

Database: AdventureWorks

QueryId: 99

Script:

```
/*AI-DBA*/
SELECT DATA_TYPE + ' ( '+ISNULL ( IIF ( CHARACTER_MAXIMUM_LENGTH = -1,'max',FORMAT ( CHARACTER_MAXIMUM_LENGTH,'0' ) ) ,'' ) +' ) ' AS [DATA_TYPE]
FROM [AdventureWorks].INFORMATION_SCHEMA.COLUMNS
WHERE '['+TABLE_CATALOG+'].['+TABLE_SCHEMA+'].['+TABLE_NAME+']' = '[AdventureWorks].[Sales].[SalesTaxRate]' AND COLUMN_NAME = 'StateProvinceID' ;
```

Database: Test33

QueryId: 101

Script:

```
( @0 int,@1 varchar ( 8000 ) ) insert into #temp select db_name ( ) [DBName] , Object_name ( IK . id ) [TableName] , schema_name ( AO . schema_id ) as [SchemaName] , I . name [IndexName] , C . name [KeyName] , T . name [DataType] , case when T . system_type_id in ( 35 , 99 , 167 , 175 , 239 , 231 ) then 'Wrong' else 'Right' end as [Design] from sys . sysindexkeys IK inner join sys . all_objects AO on AO . object_id = IK . id and AO . is_ms_shipped = @0 and AO . type = @1 inner join sys . indexes I on I . index_id = IK . indid and I . object_id = IK . id cross Apply ( select * from sys . all_columns AC where AC . column_id = IK . colid and AC . object_id = IK . id ) C inner join sys . types T on T . system_type_id = C . system_type_id order by Object_name ( IK . id ) asc , I . name asc , C . name asc
```

Database: AIDBAADV2

QueryId: 103

Script:

```
SELECT db_id ( ) AS database_id, c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.is_filestream, c.encryption_type, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END AS is_user, COUNT_BIG ( * ) AS [ColCount], CASE WHEN c.collation_name IS NULL THEN CONVERT ( VARCHAR ( 128 ) , SERVERPROPERTY ( 'Collation' ) ) ELSE c.collation_name END AS collation_name, AVG ( c.max_length ) AS avg_max_length FROM sys.columns c WITH ( NOLOCK ) LEFT OUTER JOIN sys.objects o WITH ( NOLOCK ) ON o.object_id = c.object_id AND o.type = 'U' GROUP BY c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.encryption_type, c.is_filestream, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END, CASE WHEN c.collation_name IS NULL THEN CONVERT ( VARCHAR ( 128 ) , SERVERPROPERTY ( 'Collation' ) ) ELSE c.collation_name END
```

Database: AdventureWorks

QueryId: 104

Script:

```
/*AI-DBA*/
SELECT MIN ( LEN ( [rowguid] ) ) AS [MIN],
MAX ( LEN ( [rowguid] ) ) AS [MAX],
AVG ( LEN ( [rowguid] ) ) AS [AVG],
STDEV ( LEN ( [rowguid] ) ) AS [STDDEV],
VAR ( LEN ( [rowguid] ) ) AS [VAR]
FROM [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE [rowguid] IS NOT NULL
AND LEN ( [rowguid] ) > 0
```

Database: NewDB20241029

QueryId: 106

Script:

```
SELECT db_id ( ) AS database_id, c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.is_filestream, c.encryption_type, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END AS is_user, COUNT_BIG ( * ) AS [ColCount], CASE WHEN c.collation_name IS NULL THEN CONVERT ( VARCHAR ( 128 ) , SERVERPROPERTY ( 'Collation' ) ) ELSE c.collation_name END AS collation_name, AVG ( c.max_length ) AS avg_max_length FROM sys.columns c WITH ( NOLOCK ) LEFT OUTER JOIN sys.objects o WITH ( NOLOCK ) ON o.object_id = c.object_id AND o.type = 'U' GROUP BY c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.encryption_type, c.is_filestream, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END, CASE WHEN c.collation_name IS NULL THEN CONVERT ( VARCHAR ( 128 ) , SERVERPROPERTY ( 'Collation' ) ) ELSE c.collation_name END
```

Database: Advnew2022Moved

QueryId: 110

Script:

```
( @0 int,@1 varchar ( 8000 ) ) insert into #temp select DB_Name ( ) [DBName] , case when T . system_type_id in ( 35 , 99 , 167 , 175 , 239 , 231 ) then 'Wrong' else 'Right' end as [Design] , Count ( * ) as [Count] from sys . sysindexkeys IK inner join sys . all_objects AO on AO . object_id = IK . id and AO . is_ms_shipped = @0 and AO . type = @1 inner join sys . indexes I on I . index_id = IK . indid and I . object_id = IK . id cross Apply ( select * from sys . all_columns AC where AC . column_id = IK . colid and AC . object_id = IK . id ) C inner join sys . types T on T . system_type_id = C . system_type_id group by case when T . system_type_id in ( 35 , 99 , 167 , 175 , 239 , 231 ) then 'Wrong' else 'Right' end
```

Database: newdbemo3099

QueryId: 110

Script:

```
SELECT db_id ( ) as database_id, sm.[is_inlineable] AS InlineableScalarCount, sm.[inline_type] AS InlineType, COUNT_BIG ( * ) AS ScalarCount, COUNT_BIG ( CASE WHEN sm.[definition] LIKE '%getdate%' OR sm.[definition] LIKE '%getutcdate%' OR sm.[definition] LIKE '%sysdatetime%' OR sm.[definition] LIKE '%sysutcdatetime%' OR sm.[definition] LIKE '%sysdatetimeoffset%' OR sm.[definition] LIKE '%CURRENT_TIMESTAMP%' THEN 1 END ) AS
```

ScalarCountWithDate FROM [sys].[objects] o INNER JOIN [sys].[sql_modules] sm ON o.[object_id] = sm.[object_id] WHERE o.[type] = 'FN' GROUP BY sm.[is_inlineable], sm.[inline_type]

Database: Test33

QueryId: 111

Script:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: AdventureWorks

QueryId: 112

Script:

/*AI-DBA*/
SELECT COUNT (1) AS [NONEMPTY]
FROM [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE LEN (CONVERT (NVARCHAR (MAX) ,[SpecialOfferID])) > 0 AND LEN (CONVERT (NVARCHAR (MAX) ,[SpecialOfferID])) IS NOT NULL ;

Database: AdvDest20240317

QueryId: 113

Script:
(@0 int,@1 varchar (8000)) insert into #temp select DB_Name ( ) [DBName] , case when T . system_type_id in (35 , 99 , 167 , 175 , 239 , 231) then 'Wrong' else 'Right' end as [Design] , Count (*) as [Count] from sys . sysindexkeys IK inner join sys . all_objects AO on AO . object_id = IK . id and AO . is_ms_shipped = @0 and AO . type = @1 inner join sys . indexes I on I . index_id = IK . indid and I . object_id = IK . id cross Apply (select * from sys . all_columns AC where AC . column_id = IK . colid and AC . object_id = IK . id) C inner join sys . types T on T . system_type_id = C . system_type_id group by case when T . system_type_id in (35 , 99 , 167 , 175 , 239 , 231) then 'Wrong' else 'Right' end

---

Database: AIDBAADV2

QueryId: 114

Script:
SELECT db_id ( ) as database_id, sm.[is_inlineable] AS InlineableScalarCount, sm.[inline_type] AS InlineType, COUNT_BIG (*) AS ScalarCount, COUNT_BIG (CASE WHEN sm.[definition] LIKE '%getdate%' OR sm.[definition] LIKE '%getutcdate%' OR sm.[definition] LIKE '%sysdatetime%' OR sm.[definition] LIKE '%sysutcdatetime%' OR sm.[definition] LIKE '%sysdatetimeoffset%' OR sm.[definition] LIKE '%CURRENT_TIMESTAMP%' THEN 1 END) AS ScalarCountWithDate FROM [sys].[objects] o INNER JOIN [sys].[sql_modules] sm ON o.[object_id] = sm.[object_id] WHERE o.[type] = 'FN' GROUP BY sm.[is_inlineable], sm.[inline_type]

Database: Advnew2022Moved

QueryId: 124

Script:
(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys .

all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: AdvDest20240317

QueryId: 127

Script:
(@0 varchar (8000) ,@1 varchar (8000)) select db_id ( ) as database_id , o . [type] as object_type , i . [type] as index_type , p . [data_compression] , COUNT_BIG (distinct p . [object_id]) as NumTables , COUNT_BIG (distinct CAST (p . [object_id] as VARCHAR (30)) + '|' + CAST (p . [index_id] as VARCHAR (10))) as NumIndexes , ISNULL (px . [IsPartitioned] , 0) as IsPartitioned , IIF (px . [IsPartitioned] = 1 , COUNT_BIG (1) , 0) NumPartitions , SUM (p . [rows]) NumRows from sys . partitions p inner join sys . objects o on o . [object_id] = p . [object_id] inner join sys . indexes i on i . [object_id] = p . [object_id] and i . [index_id] = p . [index_id] outer APPLY (select x . [object_id] , 1 as [IsPartitioned] from sys . partitions x where x . [object_id] = p . [object_id] group by x . [object_id] having MAX (x . partition_number) > 1) px where o . [type] not in (@0 , @1) group by o . [type] , i . [type] , p . [data_compression] , px . [IsPartitioned]

Database: AdventureWorks

QueryId: 129

Script:

```
/*AI-DBA*/
SELECT
[rowguid] AS [VALUE],
COUNT ( * )  [ROW_COUNT]
from [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE [rowguid] IS NOT null
GROUP BY [rowguid]
ORDER BY [VALUE] desc
```

Database: AdventureWorks

QueryId: 133

Script:
(@_msparam_0 nvarchar (4000) ,@_msparam_1 nvarchar (4000) ,@_msparam_2 nvarchar (4000) ,@_msparam_3 nvarchar (4000) ,@_msparam_4 nvarchar (4000)) SELECT sp.name AS [Name], sp.object_id AS [ID], sp.create_date AS [CreateDate], sp.modify_date AS [DateLastModified], ISNULL (ssp.name, N'') AS [Owner], CAST (case when sp.principal_id is null then 1 else 0 end AS bit) AS [IsSchemaOwned], SCHEMA_NAME (sp.schema_id) AS [Schema], CAST (
case
when sp.is_ms_shipped = 1 then 1
when (
select
major_id
from
sys.extended_properties
where
major_id = sp.object_id and
minor_id = 0 and
class = 1 and
name = N'microsoft_database_tools_support')

```
is not null then 1
else 0
end
AS bit ) AS [IsSystemObject], CAST ( ISNULL ( OBJECTPROPERTYEX ( sp.object_id,N'ExecIsAnsiNullsOn' ) ,0 ) AS bit ) AS
[AnsiNullsStatus], CAST ( ISNULL ( OBJECTPROPERTYEX ( sp.object_id,N'ExecIsQuotedIdentOn' ) ,0 ) AS bit ) AS
[QuotedIdentifierStatus], CAST ( ISNULL ( OBJECTPROPERTYEX ( sp.object_id, N'IsSchemaBound' ) ,0 ) AS bit ) AS
[IsSchemaBound], CAST ( CASE WHEN ISNULL ( smsp.definition, ssmsp.definition ) IS NULL THEN 1 ELSE 0 END AS bit ) AS
[IsEncrypted], CAST ( ISNULL ( smsp.is_recompiled, ssmsp.is_recompiled ) AS bit ) AS [Recompile], case when amsp.object_id is
null then N'' else asmblsp.name end AS [AssemblyName], case when amsp.object_id is null then N'' else amsp.assembly_class end
AS [ClassName], case when amsp.object_id is null then N'' else amsp.assembly_method end AS [MethodName], case when
amsp.object_id is null then case isnull ( smsp.execute_as_principal_id, -1 ) when -1 then 1 when -2 then 2 else 3 end else case isnull
( amsp.execute_as_principal_id, -1 ) when -1 then 1 when -2 then 2 else 3 end end AS [ExecutionContext], case when
amsp.object_id is null then ISNULL ( user_name ( smsp.execute_as_principal_id ) ,N'' ) else ISNULL ( user_name (
amsp.execute_as_principal_id ) , N'' ) end AS [ExecutionContextPrincipal], CAST ( ISNULL ( spp.is_auto_executed,0 ) AS bit ) AS
[Startup], CASE WHEN sp.type = N'P' THEN 1 WHEN sp.type = N'PC' THEN 2 ELSE 1 END AS [ImplementationType], CAST (
CASE sp.type WHEN N'RF' THEN 1 ELSE 0 END AS bit ) AS [ForReplication], ISNULL ( sm.uses_native_compilation,0 ) AS
[IsNativelyCompiled] FROM sys.all_objects AS sp LEFT OUTER JOIN sys.database_principals AS ssp ON ssp.principal_id =
ISNULL ( sp.principal_id, ( OBJECTPROPERTY ( sp.object_id, 'OwnerId' ) ) ) LEFT OUTER JOIN sys.sql_modules AS smsp ON
smsp.object_id = sp.object_id LEFT OUTER JOIN sys.system_sql_modules AS ssmsp ON ssmsp.object_id = sp.object_id LEFT
OUTER JOIN sys.assembly_modules AS amsp ON amsp.object_id = sp.object_id LEFT OUTER JOIN sys.assemblies AS asmblsp
ON asmblsp.assembly_id = amsp.assembly_id LEFT OUTER JOIN sys.procedures AS spp ON spp.object_id = sp.object_id LEFT
OUTER JOIN sys.all_sql_modules AS sm ON sm.object_id = sp.object_id WHERE ( sp.type = @_msparam_0 OR sp.type =
@_msparam_1 OR sp.type = @_msparam_2 ) and ( sp.name = @_msparam_3 and SCHEMA_NAME ( sp.schema_id ) =
@_msparam_4 )
```

Database: AdventureWorks

QueryId: 134

Script:

```
/*AI-DBA*/
SELECT COUNT ( 1 ) AS [NONEMPTY]
FROM [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE LEN ( CONVERT ( NVARCHAR ( MAX ) ,[ProductID] ) ) > 0 AND LEN ( CONVERT ( NVARCHAR ( MAX ) ,[ProductID] ) )
IS NOT NULL ;
```

Database: Advnew2022Moved

QueryId: 139

Script:

```
( @0 int ) insert into #sqls select DB_ID ( ) , [definition] from sys . all_sql_modules where object_id > @0 and ( [definition] like
'%cast%' or [Definition] like '%convert%' )
```

Database: AdvNewDB2022Portal

QueryId: 139

Script:

```
SELECT db_id ( ) AS database_id, o.[type] AS object_type, i.[type] AS index_type, p.[data_compression],
COUNT_BIG ( DISTINCT p.[object_id] ) AS NumTables, COUNT_BIG ( DISTINCT CAST ( p.[object_id] AS VARCHAR ( 30 ) ) +
'|' + CAST ( p.[index_id] AS VARCHAR ( 10 ) ) ) AS NumIndexes, ISNULL ( px.[IsPartitioned], 0 ) AS IsPartitioned, IIF (
px.[IsPartitioned] = 1, COUNT_BIG ( 1 ) , 0 ) NumPartitions, SUM ( p.[rows] ) NumRows FROM sys.partitions p INNER
JOIN sys.objects o ON o.[object_id] = p.[object_id] INNER JOIN sys.indexes i ON i.[object_id] = p.[object_id] AND
i.[index_id] = p.[index_id] OUTER APPLY ( SELECT x.[object_id], 1 AS [IsPartitioned] FROM sys.partitions x WHERE
x.[object_id] = p.[object_id] GROUP by x.[object_id] HAVING MAX ( x.partition_number ) > 1 ) px WHERE o.[type]
NOT IN ( 'S', 'IT' ) GROUP BY o.[type] ,i.[type] ,p.[data_compression] ,px.[IsPartitioned]
```

Database: Advnew2022Moved

QueryId: 145

Script:
(@0 int) insert into #temp select db_name ( ) as [DB_Name] , object_name (IU . object_id) as [ObjName] , I . name as [IndexName] , I . type_desc as [IndexType] , IU . user_seeks , IU . user_scans , IU . user_lookups , case when IU . user_seeks = 0 then 0 else (IU . user_seeks * 100) / (IU . user_seeks + IU . user_scans + IU . user_lookups) end as [User_Seeks_Pct] , case when IU . user_scans = 0 then 0 else (IU . user_scans * 100) / (IU . user_seeks + IU . user_scans + IU . user_lookups) end as [User_Scans_Pct] , case when IU . user_lookups = 0 then 0 else (IU . user_lookups * 100) / (IU . user_seeks + IU . user_scans + IU . user_lookups) end as [User_Lookups_Pct] from sys . dm_db_index_usage_stats IU inner join sys . indexes I on I . index_id = IU . index_id and I . object_id = IU . object_id inner join sys . all_objects AO on AO . object_id = IU . object_id and AO . is_ms_shipped = @0

Database: Advnew2022

QueryId: 146

Script:
(@0 varchar (8000)) select db_id ( ) as database_id , sm . [is_inlineable] as InlineableScalarCount , sm . [inline_type] as InlineType , COUNT_BIG (*) as ScalarCount , COUNT_BIG (case when sm . [definition] like '%getdate%' or sm . [definition] like '%getutcdate%' or sm . [definition] like '%sysdatetime%' or sm . [definition] like '%sysutcdatetime%' or sm . [definition] like '%sysdatetimeoffset%' or sm . [definition] like '%CURRENT_TIMESTAMP%' then 1 end) as ScalarCountWithDate from [sys] . [objects] o inner join [sys] . [sql_modules] sm on o . [object_id] = sm . [object_id] where o . [type] = @0 group by sm . [is_inlineable] , sm . [inline_type]

Database: AdventureWorks

QueryId: 149

Script:
SELECT db_id ( ) AS database_id, o.[type] AS object_type, i.[type] AS index_type, p.[data_compression], COUNT_BIG (DISTINCT p.[object_id]) AS NumTables, COUNT_BIG (DISTINCT CAST (p.[object_id] AS VARCHAR (30)) + '|' + CAST (p.[index_id] AS VARCHAR (10))) AS NumIndexes, ISNULL (px.[IsPartitioned], 0) AS IsPartitioned, IIF (px.[IsPartitioned] = 1, COUNT_BIG (1) , 0) NumPartitions, SUM (p.[rows]) NumRows FROM sys.partitions p INNER JOIN sys.objects o ON o.[object_id] = p.[object_id] INNER JOIN sys.indexes i ON i.[object_id] = p.[object_id] AND i.[index_id] = p.[index_id] OUTER APPLY (SELECT x.[object_id], 1 AS [IsPartitioned] FROM sys.partitions x WHERE x.[object_id] = p.[object_id] GROUP by x.[object_id] HAVING MAX (x.partition_number) > 1) px WHERE o.[type] NOT IN ('S', 'IT') GROUP BY o.[type] ,i.[type] ,p.[data_compression] ,px.[IsPartitioned]

Database: AdventureWorks

QueryId: 153

Script:

```
/*AI-DBA*/
SELECT
[BillToAddressID] AS [VALUE],
COUNT ( * ) [ROW_COUNT]
from [AdventureWorks].[Sales].[SalesOrderHeader]
WHERE [BillToAddressID] IS NOT null
GROUP BY [BillToAddressID]
ORDER BY [VALUE] desc
```

Database: Advnew2022

QueryId: 153

Script:
(@0 int) insert into #temp select db_name ( ) as [DB_Name] , object_name (IU . object_id) as [ObjName] , I . name as [IndexName] , I . type_desc as [IndexType] , IU . user_seeks , IU . user_scans , IU . user_lookups , case when IU . user_seeks = 0 then 0 else (IU . user_seeks * 100) / (IU . user_seeks + IU . user_scans + IU . user_lookups) end as [User_Seeks_Pct] , case

```
when IU . user_scans = 0 then 0 else ( IU . user_scans * 100 ) / ( IU . user_seeks + IU . user_scans + IU . user_lookups ) end as [User_Scans_Pct] , case when IU . user_lookups = 0 then 0 else ( IU . user_lookups * 100 ) / ( IU . user_seeks + IU . user_scans + IU . user_lookups ) end as [User_Lookups_Pct] from sys . dm_db_index_usage_stats IU inner join sys . indexes I on I . index_id = IU . index_id and I . object_id = IU . object_id inner join sys . all_objects AO on AO . object_id = IU . object_id and AO . is_ms_shipped = @0
```

Database: Adv2022ShalevSoft

QueryId: 158

Script:

```
( @0 varchar ( 8000 ) ) select db_id ( ) as database_id , c . system_type_id , c . user_type_id , c . is_sparse , c . is_column_set , c . is_filestream , c . encryption_type , case when o . object_id is not null then 1 else 0 end as is_user , COUNT_BIG ( * ) as [ColCount] , case when c . collation_name is null then convert ( VARCHAR ( 128 ) , SERVERPROPERTY ( 'Collation' ) ) else c . collation_name end as collation_name , AVG ( c . max_length ) as avg_max_length from sys . columns c with ( NOLOCK ) left outer join sys . objects o with ( NOLOCK ) on o . object_id = c . object_id and o . type = @0 group by c . system_type_id , c . user_type_id , c . is_sparse , c . is_column_set , c . encryption_type , c . is_filestream , case when o . object_id is not null then 1 else 0 end , case when c . collation_name is null then convert ( VARCHAR ( 128 ) , SERVERPROPERTY ( 'Collation' ) ) else c . collation_name end
```

Database: AdventureWorks

QueryId: 161

Script:

```
/*AI-DBA*/
SELECT COUNT ( 1 ) AS [EMPTY]
FROM [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE LEN ( CONVERT ( NVARCHAR ( MAX ) ,[ModifiedDate] ) ) = 0 AND LEN ( CONVERT ( NVARCHAR ( MAX ) ,[ModifiedDate] ) ) IS NOT NULL ;
```

Database: Advnew2022

QueryId: 165

Script:

```
( @0 int,@3 varchar ( 8000 ) ,@4 varchar ( 8000 ) ,@5 varchar ( 8000 ) ,@6 varchar ( 8000 ) ,@9 varchar ( 8000 ) ,@10 varchar ( 8000 ) ,@11 varchar ( 8000 ) ) with CTE as ( select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace ( lower ( Replace ( definition , char ( 13 ) + Char ( 10 ) , @3 ) ) , @4 , @5 ) like '%with ( %index ( %' union all select Object_name ( T . objectid ) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text ( qs . Sql_handle ) T where CP . objtype = @6 and Replace ( lower ( Replace ( T . text , char ( 13 ) + Char ( 10 ) , @9 ) ) , @10 , @11 ) like '%with ( %index ( %' ) insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join ( select ( '%' + T . name + '%' + I . name + '%' ) as [filter] , schema_name ( T . schema_id ) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null ) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT
```

Database: Advnew2022Moved

QueryId: 168

Script:

```
( @0 varchar ( 8000 ) ,@1 varchar ( 8000 ) ) select db_id ( ) as database_id , o . [type] as object_type , i . [type] as index_type , p . [data_compression] , COUNT_BIG ( distinct p . [object_id] ) as NumTables , COUNT_BIG ( distinct CAST ( p . [object_id] as VARCHAR ( 30 ) ) + '|' + CAST ( p . [index_id] as VARCHAR ( 10 ) ) ) as NumIndexes , ISNULL ( px . [IsPartitioned] , 0 ) as IsPartitioned , IIF ( px . [IsPartitioned] = 1 , COUNT_BIG ( 1 ) , 0 ) NumPartitions , SUM ( p . [rows] ) NumRows from sys . partitions p inner join sys . objects o on o . [object_id] = p . [object_id] inner join sys . indexes i on i . [object_id] = p . [object_id] and i . [index_id] = p . [index_id] outer APPLY ( select x . [object_id] , 1 as [IsPartitioned] from sys . partitions x where x . [object_id] = p . [object_id] group by x . [object_id] having MAX ( x . partition_number ) > 1 ) px where o . [type] not in ( @0 , @1 ) group by o . [type] , i . [type] , p . [data_compression] , px . [IsPartitioned]
```

Database: AdventureWorks

QueryId: 171

Script:

```
/*AI-DBA*/
SELECT COUNT ( 1 ) AS [NONEMPTY]
FROM [AdventureWorks].[Sales].[SalesOrderHeader]
WHERE LEN ( CONVERT ( NVARCHAR ( MAX ) ,[RevisionNumber] ) ) > 0 AND LEN ( CONVERT ( NVARCHAR ( MAX )
,[RevisionNumber] ) ) IS NOT NULL ;
```

Database: AdvDest20240317

QueryId: 173

Script:

```
( @0 varchar ( 8000 ) ,@1 varchar ( 8000 ) ,@2 varchar ( 8000 ) ,@3 varchar ( 8000 ) ,@4 varchar ( 8000 ) ,@5 varchar ( 8000 ) ,@6
varchar ( 8000 ) ,@7 varchar ( 8000 ) ,@8 varchar ( 8000 ) ,@9 varchar ( 8000 ) ,@10 varchar ( 8000 ) ,@11 varchar ( 8000 ) ,@12
varchar ( 8000 ) ,@13 varchar ( 8000 ) ,@14 varchar ( 8000 ) ,@15 varchar ( 8000 ) ,@16 varchar ( 8000 ) ,@17 varchar ( 8000 )
,@18 varchar ( 8000 ) ,@19 varchar ( 8000 ) ,@20 varchar ( 8000 ) ) select db_id ( ) as database_id , o . [type] as ModuleType ,
COUNT_BIG ( * ) as ModuleCount from sys . objects as o with ( nolock ) where o . type in ( @0 , @1 , @2 , @3 , @4 , @5 , @6
, @7 , @8 , @9 , @10 , @11 , @12 , @13 , @14 , @15 , @16 , @17 , @18 , @19 , @20 ) group by o . [type]
```

Database: Advnew2022Moved

QueryId: 174

Script:

```
( @0 varchar ( 8000 ) ) select db_id ( ) as database_id , c . system_type_id , c . user_type_id , c . is_sparse , c . is_column_set , c .
is_filestream , c . encryption_type , case when o . object_id is not null then 1 else 0 end as is_user , COUNT_BIG ( * ) as [ColCount]
, case when c . collation_name is null then convert ( VARCHAR ( 128 ) , SERVERPROPERTY ( 'Collation' ) ) else c .
collation_name end as collation_name , AVG ( c . max_length ) as avg_max_length from sys . columns c with ( NOLOCK ) left
outer join sys . objects o with ( NOLOCK ) on o . object_id = c . object_id and o . type = @0 group by c . system_type_id , c .
user_type_id , c . is_sparse , c . is_column_set , c . encryption_type , c . is_filestream , case when o . object_id is not null then 1 else
0 end , case when c . collation_name is null then convert ( VARCHAR ( 128 ) , SERVERPROPERTY ( 'Collation' ) ) else c .
collation_name end
```

Database: AdvNew2022Restored2

QueryId: 174

Script:

```
SELECT db_id ( ) AS database_id, o.[type] AS object_type, i.[type] AS index_type, p.[data_compression],
COUNT_BIG ( DISTINCT p.[object_id] ) AS NumTables, COUNT_BIG ( DISTINCT CAST ( p.[object_id] AS VARCHAR ( 30 ) ) +
'|' + CAST ( p.[index_id] AS VARCHAR ( 10 ) ) ) AS NumIndexes, ISNULL ( px.[IsPartitioned], 0 ) AS IsPartitioned, IIF (
px.[IsPartitioned] = 1, COUNT_BIG ( 1 ) , 0 ) NumPartitions, SUM ( p.[rows] ) NumRows FROM sys.partitions p INNER
JOIN sys.objects o ON o.[object_id] = p.[object_id] INNER JOIN sys.indexes i ON i.[object_id] = p.[object_id] AND
i.[index_id] = p.[index_id] OUTER APPLY ( SELECT x.[object_id], 1 AS [IsPartitioned] FROM sys.partitions x WHERE
x.[object_id] = p.[object_id] GROUP by x.[object_id] HAVING MAX ( x.partition_number ) > 1 ) px WHERE o.[type]
NOT IN ( 'S', 'IT' ) GROUP BY o.[type] ,i.[type] ,p.[data_compression] ,px.[IsPartitioned]
```

Database: Demo20240411

QueryId: 178

Script:

```
SELECT db_id ( ) AS database_id, c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set,
c.is_filestream, c.encryption_type, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END AS is_user, COUNT_BIG
( * ) AS [ColCount], CASE WHEN c.collation_name IS NULL THEN CONVERT ( VARCHAR ( 128 ) , SERVERPROPERTY (
'Collation' ) ) ELSE c.collation_name END AS collation_name, AVG ( c.max_length ) AS avg_max_length FROM
sys.columns c WITH ( NOLOCK ) LEFT OUTER JOIN sys.objects o WITH ( NOLOCK ) ON o.object_id = c.object_id
AND o.type = 'U' GROUP BY c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set,
```

c.encryption_type, c.is_filestream, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END, CASE WHEN c.collation_name IS NULL THEN CONVERT (VARCHAR (128) , SERVERPROPERTY ('Collation')) ELSE c.collation_name END

Database: Test33

QueryId: 179

Script:
(@0 varchar (8000)) select db_id ( ) as database_id , c . system_type_id , c . user_type_id , c . is_sparse , c . is_column_set , c . is_filestream , c . encryption_type , case when o . object_id is not null then 1 else 0 end as is_user , COUNT_BIG (*) as [ColCount] , case when c . collation_name is null then convert (VARCHAR (128) , SERVERPROPERTY ('Collation')) else c . collation_name end as collation_name , AVG (c . max_length) as avg_max_length from sys . columns c with (NOLOCK) left outer join sys . objects o with (NOLOCK) on o . object_id = c . object_id and o . type = @0 group by c . system_type_id , c . user_type_id , c . is_sparse , c . is_column_set , c . encryption_type , c . is_filestream , case when o . object_id is not null then 1 else 0 end , case when c . collation_name is null then convert (VARCHAR (128) , SERVERPROPERTY ('Collation')) else c . collation_name end

---

Database: AdventureWorks

QueryId: 182

Script:

/*AI-DBA*/
SELECT COUNT (1) AS [NONEMPTY]
FROM [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE LEN (CONVERT (NVARCHAR (MAX) ,[rowguid])) > 0 AND LEN (CONVERT (NVARCHAR (MAX) ,[rowguid])) IS NOT NULL ;

Database: AdventureWorks

QueryId: 184

Script:

/*AI-DBA*/

SELECT LEN ([SalesOrderDetailID]) AS [LENGTH],
count (*) AS [COUNT]
FROM [AdventureWorks].[Sales].[SalesOrderDetail]
GROUP BY LEN ([SalesOrderDetailID])
ORDER BY 1

Database: AdvNew2022Restored2

QueryId: 184

Script:
SELECT db_id ( ) as database_id, sm.[is_inlineable] AS InlineableScalarCount, sm.[inline_type] AS InlineType, COUNT_BIG (*) AS ScalarCount, COUNT_BIG (CASE WHEN sm.[definition] LIKE '%getdate%' OR sm.[definition] LIKE '%getutcdate%' OR sm.[definition] LIKE '%sysdatetime%' OR sm.[definition] LIKE '%sysutcdatetime%' OR sm.[definition] LIKE '%sysdatetimeoffset%' OR sm.[definition] LIKE '%CURRENT_TIMESTAMP%' THEN 1 END) AS ScalarCountWithDate FROM [sys].[objects] o INNER JOIN [sys].[sql_modules] sm ON o.[object_id] = sm.[object_id] WHERE o.[type] = 'FN' GROUP BY sm.[is_inlineable], sm.[inline_type]

Database: ReportDB_Copy

QueryId: 186

Script:
SELECT db_id ( ) AS database_id, c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.is_filestream, c.encryption_type, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END AS is_user, COUNT_BIG

(*) AS [ColCount], CASE WHEN c.collation_name IS NULL THEN CONVERT (VARCHAR (128) , SERVERPROPERTY ('Collation')) ELSE c.collation_name END AS collation_name, AVG (c.max_length) AS avg_max_length FROM sys.columns c WITH (NOLOCK) LEFT OUTER JOIN sys.objects o WITH (NOLOCK) ON o.object_id = c.object_id AND o.type = 'U' GROUP BY c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.encryption_type, c.is_filestream, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END, CASE WHEN c.collation_name IS NULL THEN CONVERT (VARCHAR (128) , SERVERPROPERTY ('Collation')) ELSE c.collation_name END

Database: Adv2022ShalevSoft

QueryId: 187

Script:
(@0 varchar (8000) ,@1 varchar (8000)) select db_id ( ) as database_id , o . [type] as object_type , i . [type] as index_type , p . [data_compression] , COUNT_BIG (distinct p . [object_id]) as NumTables , COUNT_BIG (distinct CAST (p . [object_id] as VARCHAR (30)) + '|' + CAST (p . [index_id] as VARCHAR (10))) as NumIndexes , ISNULL (px . [IsPartitioned] , 0) as IsPartitioned , IIF (px . [IsPartitioned] = 1 , COUNT_BIG (1) , 0) NumPartitions , SUM (p . [rows]) NumRows from sys . partitions p inner join sys . objects o on o . [object_id] = p . [object_id] inner join sys . indexes i on i . [object_id] = p . [object_id] and i . [index_id] = p . [index_id] outer APPLY (select x . [object_id] , 1 as [IsPartitioned] from sys . partitions x where x . [object_id] = p . [object_id] group by x . [object_id] having MAX (x . partition_number) > 1) px where o . [type] not in (@0 , @1) group by o . [type] , i . [type] , p . [data_compression] , px . [IsPartitioned]

Database: ReportDB678

QueryId: 188

Script:
SELECT db_id ( ) as database_id, sm.[is_inlineable] AS InlineableScalarCount, sm.[inline_type] AS InlineType, COUNT_BIG (*) AS ScalarCount, COUNT_BIG (CASE WHEN sm.[definition] LIKE '%getdate%' OR sm.[definition] LIKE '%getutcdate%' OR sm.[definition] LIKE '%sysdatetime%' OR sm.[definition] LIKE '%sysutcdatetime%' OR sm.[definition] LIKE '%sysdatetimeoffset%' OR sm.[definition] LIKE '%CURRENT_TIMESTAMP%' THEN 1 END) AS ScalarCountWithDate FROM [sys].[objects] o INNER JOIN [sys].[sql_modules] sm ON o.[object_id] = sm.[object_id] WHERE o.[type] = 'FN' GROUP BY sm.[is_inlineable], sm.[inline_type]

Database: Demo20240411

QueryId: 189

Script:
SELECT db_id ( ) AS database_id, COUNT_BIG (*) AS [NumExternalStats] FROM sys.tables t WITH (nolock) INNER JOIN sys.stats s WITH (nolock) ON t.object_id = s.object_id WHERE t.is_external = 1

Database: AdvNew2022Restored

QueryId: 193

Script:
SELECT db_id ( ) AS database_id, c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.is_filestream, c.encryption_type, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END AS is_user, COUNT_BIG (*) AS [ColCount], CASE WHEN c.collation_name IS NULL THEN CONVERT (VARCHAR (128) , SERVERPROPERTY ('Collation')) ELSE c.collation_name END AS collation_name, AVG (c.max_length) AS avg_max_length FROM sys.columns c WITH (NOLOCK) LEFT OUTER JOIN sys.objects o WITH (NOLOCK) ON o.object_id = c.object_id AND o.type = 'U' GROUP BY c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.encryption_type, c.is_filestream, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END, CASE WHEN c.collation_name IS NULL THEN CONVERT (VARCHAR (128) , SERVERPROPERTY ('Collation')) ELSE c.collation_name END

Database: AdvNew2022Restored2

QueryId: 195

Script:
SELECT db_id ( ) AS database_id, c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.is_filestream, c.encryption_type, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END AS is_user, COUNT_BIG (*) AS [ColCount], CASE WHEN c.collation_name IS NULL THEN CONVERT (VARCHAR (128) , SERVERPROPERTY (

'Collation')) ELSE c.collation_name END AS collation_name, AVG (c.max_length) AS avg_max_length FROM sys.columns c WITH (NOLOCK) LEFT OUTER JOIN sys.objects o WITH (NOLOCK) ON o.object_id = c.object_id AND o.type = 'U' GROUP BY c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.encryption_type, c.is_filestream, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END, CASE WHEN c.collation_name IS NULL THEN CONVERT (VARCHAR (128) , SERVERPROPERTY ('Collation')) ELSE c.collation_name END

Database: AdvNewDB2022Portal

QueryId: 196

Script:
SELECT db_id ( ) as database_id, sm.[is_inlineable] AS InlineableScalarCount, sm.[inline_type] AS InlineType, COUNT_BIG (*) AS ScalarCount, COUNT_BIG (CASE WHEN sm.[definition] LIKE '%getdate%' OR sm.[definition] LIKE '%getutcdate%' OR sm.[definition] LIKE '%sysdatetime%' OR sm.[definition] LIKE '%sysutcdatetime%' OR sm.[definition] LIKE '%sysdatetimeoffset%' OR sm.[definition] LIKE '%CURRENT_TIMESTAMP%' THEN 1 END) AS ScalarCountWithDate FROM [sys].[objects] o INNER JOIN [sys].[sql_modules] sm ON o.[object_id] = sm.[object_id] WHERE o.[type] = 'FN' GROUP BY sm.[is_inlineable], sm.[inline_type]

Database: AdventureWorks

QueryId: 197

Script:

```
/*AI-DBA*/
SELECT row_type,
SUM ( row_count ) AS row_count
FROM
( SELECT
CASE WHEN [value] IS NULL then 'NULL'
WHEN row_count = 1 then 'Unique'
ELSE 'Non Unique'
END AS row_type,
row_count
FROM (
SELECT [SalesTaxRateID] [value],
COUNT ( 1 ) row_count
FROM [AdventureWorks].[Sales].[SalesTaxRate]
GROUP BY [SalesTaxRateID] ) X ) Y
GROUP BY row_type
```

Database: Demo20240411

QueryId: 197

Script:
SELECT db_id ( ) AS database_id, o.[type] AS object_type, i.[type] AS index_type, p.[data_compression], COUNT_BIG (DISTINCT p.[object_id]) AS NumTables, COUNT_BIG (DISTINCT CAST (p.[object_id] AS VARCHAR (30)) + '|' + CAST (p.[index_id] AS VARCHAR (10))) AS NumIndexes, ISNULL (px.[IsPartitioned], 0) AS IsPartitioned, IIF (px.[IsPartitioned] = 1, COUNT_BIG (1) , 0) NumPartitions, SUM (p.[rows]) NumRows FROM sys.partitions p INNER JOIN sys.objects o ON o.[object_id] = p.[object_id] INNER JOIN sys.indexes i ON i.[object_id] = p.[object_id] AND i.[index_id] = p.[index_id] OUTER APPLY (SELECT x.[object_id], 1 AS [IsPartitioned] FROM sys.partitions x WHERE x.[object_id] = p.[object_id] GROUP by x.[object_id] HAVING MAX (x.partition_number) > 1) px WHERE o.[type] NOT IN ('S', 'IT') GROUP BY o.[type] ,i.[type] ,p.[data_compression] ,px.[IsPartitioned]

Database: AdventureWorks

QueryId: 198

Script:

```
/*AI-DBA*/
SELECT COUNT ( 1 ) AS [NONEMPTY]
FROM [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE LEN ( CONVERT ( NVARCHAR ( MAX ) ,[UnitPriceDiscount] ) ) > 0 AND LEN ( CONVERT ( NVARCHAR ( MAX )
,[UnitPriceDiscount] ) ) IS NOT NULL ;
```

Database: newdbemo3099

QueryId: 199

Script:

```
SELECT ISNULL ( AVG ( c ) ,0 ) AS [XTPAvgNumOfIndexes], ISNULL ( MAX ( c ) ,0 ) AS [XTPMaxNumOfIndexes], db_id ( )
AS database_id FROM ( SELECT COUNT_BIG ( * ) AS c FROM sys.indexes i WITH ( nolock ) JOIN sys.tables t WITH (
nolock ) ON i.object_id = t.object_id WHERE t.is_memory_optimized = 1 GROUP BY i.object_id ) a
```

Database: AdventureWorks

QueryId: 201

Script:

```
/*AI-DBA*/
SELECT COUNT ( 1 ) AS [rows]
FROM [AdventureWorks].[Sales].[SalesOrderDetail] ;
```

Database: AdvDest20240317

QueryId: 203

Script:

```
( @0 varchar ( 8000 ) ) select db_id ( ) as database_id , c . system_type_id , c . user_type_id , c . is_sparse , c . is_column_set , c .
is_filestream , c . encryption_type , case when o . object_id is not null then 1 else 0 end as is_user , COUNT_BIG ( * ) as [ColCount]
, case when c . collation_name is null then convert ( VARCHAR ( 128 ) , SERVERPROPERTY ( 'Collation' ) ) else c .
collation_name end as collation_name , AVG ( c . max_length ) as avg_max_length from sys . columns c with ( NOLOCK ) left
outer join sys . objects o with ( NOLOCK ) on o . object_id = c . object_id and o . type = @0 group by c . system_type_id , c .
user_type_id , c . is_sparse , c . is_column_set , c . encryption_type , c . is_filestream , case when o . object_id is not null then 1 else
0 end , case when c . collation_name is null then convert ( VARCHAR ( 128 ) , SERVERPROPERTY ( 'Collation' ) ) else c .
collation_name end
```

Database: AdventureWorks

QueryId: 210

Script:

```
/*AI-DBA*/
SELECT
[LineTotal] AS [VALUE],
COUNT ( * ) [ROW_COUNT]
from [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE [LineTotal] IS NOT null
GROUP BY [LineTotal]
ORDER BY [VALUE] desc
```

Database: AdventureWorks

QueryId: 215

Script:

```
SELECT db_id ( ) AS database_id, c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set,
c.is_filestream, c.encryption_type, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END AS is_user, COUNT_BIG
( * ) AS [ColCount], CASE WHEN c.collation_name IS NULL THEN CONVERT ( VARCHAR ( 128 ) , SERVERPROPERTY (
```

'Collation')) ELSE c.collation_name END AS collation_name, AVG (c.max_length) AS avg_max_length FROM sys.columns c WITH (NOLOCK) LEFT OUTER JOIN sys.objects o WITH (NOLOCK) ON o.object_id = c.object_id AND o.type = 'U' GROUP BY c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.encryption_type, c.is_filestream, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END, CASE WHEN c.collation_name IS NULL THEN CONVERT (VARCHAR (128) , SERVERPROPERTY ('Collation')) ELSE c.collation_name END

Database: AdventureWorks

QueryId: 216

Script:

```
/*AI-DBA*/
SELECT
[Freight] AS [VALUE],
COUNT ( * ) [ROW_COUNT]
from [AdventureWorks].[Sales].[SalesOrderHeader]
WHERE [Freight] IS NOT null
GROUP BY [Freight]
ORDER BY [VALUE] desc
```

Database: AdventureWorks

QueryId: 217

Script:

```
/*AI-DBA*/
SELECT K.TABLE_NAME ,
K.COLUMN_NAME ,
K.CONSTRAINT_NAME
FROM [AdventureWorks].INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS C
JOIN [AdventureWorks].INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS K ON C.TABLE_NAME = K.TABLE_NAME
AND C.CONSTRAINT_CATALOG = K.CONSTRAINT_CATALOG
AND C.CONSTRAINT_SCHEMA = K.CONSTRAINT_SCHEMA
AND C.CONSTRAINT_NAME = K.CONSTRAINT_NAME
WHERE C.CONSTRAINT_TYPE = 'PRIMARY KEY'
AND K.COLUMN_NAME = 'StateProvinceID'
AND '['+K.TABLE_CATALOG+'].['+K.TABLE_SCHEMA+'].['+ K.TABLE_NAME+']' = '[AdventureWorks].[Sales].[SalesTaxRate]' ;
```

Database: AdvNew2022Restored

QueryId: 219

Script:

SELECT db_id ( ) as database_id, sm.[is_inlineable] AS InlineableScalarCount, sm.[inline_type] AS InlineType, COUNT_BIG (*) AS ScalarCount, COUNT_BIG (CASE WHEN sm.[definition] LIKE '%getdate%' OR sm.[definition] LIKE '%getutcdate%' OR sm.[definition] LIKE '%sysdatetime%' OR sm.[definition] LIKE '%sysutcdatetime%' OR sm.[definition] LIKE '%sysdatetimeoffset%' OR sm.[definition] LIKE '%CURRENT_TIMESTAMP%' THEN 1 END) AS ScalarCountWithDate FROM [sys].[objects] o INNER JOIN [sys].[sql_modules] sm ON o.[object_id] = sm.[object_id] WHERE o.[type] = 'FN' GROUP BY sm.[is_inlineable], sm.[inline_type]

Database: Test33

QueryId: 223

Script:

(@0 varchar (8000)) select db_id ( ) as database_id , sm . [is_inlineable] as InlineableScalarCount , sm . [inline_type] as InlineType , COUNT_BIG (*) as ScalarCount , COUNT_BIG (case when sm . [definition] like '%getdate%' or sm . [definition] like '%getutcdate%' or sm . [definition] like '%sysdatetime%' or sm . [definition] like '%sysutcdatetime%' or sm . [definition] like '%sysdatetimeoffset%' or sm . [definition] like '%CURRENT_TIMESTAMP%' then 1 end) as ScalarCountWithDate from [sys] .

[objects] o inner join [sys] . [sql_modules] sm on o . [object_id] = sm . [object_id] where o . [type] = @0 group by sm . [is_inlineable] , sm . [inline_type]

Database: AdventureWorks

QueryId: 228

Script:

```
/*AI-DBA*/
SELECT COUNT ( 1 ) AS [NONEMPTY]
FROM [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE LEN ( CONVERT ( NVARCHAR ( MAX ) ,[SalesOrderID] ) ) > 0 AND LEN ( CONVERT ( NVARCHAR ( MAX ) ,[SalesOrderID] ) ) IS NOT NULL ;
```

Database: Advnew2022

QueryId: 228

Script:

(@0 int,@1 varchar (8000)) insert into #temp select DB_Name ( ) [DBName] , case when T . system_type_id in (35 , 99 , 167 , 175 , 239 , 231) then 'Wrong' else 'Right' end as [Design] , Count (*) as [Count] from sys . sysindexkeys IK inner join sys . all_objects AO on AO . object_id = IK . id and AO . is_ms_shipped = @0 and AO . type = @1 inner join sys . indexes I on I . index_id = IK . indid and I . object_id = IK . id cross Apply (select * from sys . all_columns AC where AC . column_id = IK . colid and AC . object_id = IK . id) C inner join sys . types T on T . system_type_id = C . system_type_id group by case when T . system_type_id in (35 , 99 , 167 , 175 , 239 , 231) then 'Wrong' else 'Right' end

---

Database: ReportDB678

QueryId: 228

Script:

SELECT db_id ( ) AS database_id, c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.is_filestream, c.encryption_type, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END AS is_user, COUNT_BIG (*) AS [ColCount], CASE WHEN c.collation_name IS NULL THEN CONVERT (VARCHAR (128) , SERVERPROPERTY ('Collation')) ELSE c.collation_name END AS collation_name, AVG (c.max_length) AS avg_max_length FROM sys.columns c WITH (NOLOCK) LEFT OUTER JOIN sys.objects o WITH (NOLOCK) ON o.object_id = c.object_id AND o.type = 'U' GROUP BY c.system_type_id, c.user_type_id, c.is_sparse, c.is_column_set, c.encryption_type, c.is_filestream, CASE WHEN o.object_id IS NOT NULL THEN 1 ELSE 0 END, CASE WHEN c.collation_name IS NULL THEN CONVERT (VARCHAR (128) , SERVERPROPERTY ('Collation')) ELSE c.collation_name END

Database: AdvDest20240317

QueryId: 231

Script:

(@0 int,@3 varchar (8000) ,@4 varchar (8000) ,@5 varchar (8000) ,@6 varchar (8000) ,@9 varchar (8000) ,@10 varchar (8000) ,@11 varchar (8000)) with CTE as (select AO . name , sm . definition from sys . all_objects AO inner join sys . all_sql_modules SM on SM . object_id = AO . object_id where AO . is_ms_shipped = @0 and Replace (lower (Replace (definition , char (13) + Char (10) , @3)) , @4 , @5) like '%with (%index (%' union all select Object_name (T . objectid) as [name] , T . text as [definition] from sys . dm_exec_query_stats qs inner join sys . dm_exec_cached_plans CP on CP . plan_handle = qs . plan_handle cross apply sys . dm_exec_sql_text (qs . Sql_handle) T where CP . objtype = @6 and Replace (lower (Replace (T . text , char (13) + Char (10) , @9)) , @10 , @11) like '%with (%index (%') insert into #Temp select DB_NAME ( ) as [DB_Name] , S . [SchemaName] , S . TableName , S . IndexName from CTE C inner join (select ('%' + T . name + '%' + I . name + '%') as [filter] , schema_name (T . schema_id) as [SchemaName] , T . name as [TableName] , I . name as [IndexName] from sys . indexes I inner join sys . tables T on T . object_id = I . object_id where I . name is not null) S on C . definition collate DATABASE_DEFAULT like S . [filter] collate DATABASE_DEFAULT

Database: AdventureWorks

QueryId: 237

Script:

```
/*AI-DBA*/
SELECT
[SalesOrderID] AS [VALUE],
COUNT ( * ) [ROW_COUNT]
from [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE [SalesOrderID] IS NOT null
GROUP BY [SalesOrderID]
ORDER BY [VALUE] desc
```

Database: AIDBAADV2

QueryId: 238

Script:

```
SELECT db_id ( ) AS database_id, o.[type] AS object_type, i.[type] AS index_type, p.[data_compression], COUNT_BIG ( DISTINCT p.[object_id] ) AS NumTables, COUNT_BIG ( DISTINCT CAST ( p.[object_id] AS VARCHAR ( 30 ) ) + '|' + CAST ( p.[index_id] AS VARCHAR ( 10 ) ) ) AS NumIndexes, ISNULL ( px.[IsPartitioned], 0 ) AS IsPartitioned, IIF ( px.[IsPartitioned] = 1, COUNT_BIG ( 1 ) , 0 ) NumPartitions, SUM ( p.[rows] ) NumRows FROM sys.partitions p INNER JOIN sys.objects o ON o.[object_id] = p.[object_id] INNER JOIN sys.indexes i ON i.[object_id] = p.[object_id] AND i.[index_id] = p.[index_id] OUTER APPLY ( SELECT x.[object_id], 1 AS [IsPartitioned] FROM sys.partitions x WHERE x.[object_id] = p.[object_id] GROUP by x.[object_id] HAVING MAX ( x.partition_number ) > 1 ) px WHERE o.[type] NOT IN ( 'S', 'IT' ) GROUP BY o.[type] ,i.[type] ,p.[data_compression] ,px.[IsPartitioned]
```

Database: AdventureWorks

QueryId: 240

Script:

```
/*AI-DBA*/
SELECT COUNT ( 1 ) AS [NONEMPTY]
FROM [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE LEN ( CONVERT ( NVARCHAR ( MAX ) ,[LineTotal] ) ) > 0 AND LEN ( CONVERT ( NVARCHAR ( MAX ) ,[LineTotal] ) ) IS NOT NULL ;
```

Database: AdventureWorks

QueryId: 240

Script:

```
/*AI-DBA*/
SELECT
[CreditCardID] AS [VALUE],
COUNT ( * ) [ROW_COUNT]
from [AdventureWorks].[Sales].[SalesOrderHeader]
WHERE [CreditCardID] IS NOT null
GROUP BY [CreditCardID]
ORDER BY [VALUE] desc
```

Database: Test33

QueryId: 243

Script:

```
( @0 varchar ( 8000 ) ,@1 varchar ( 8000 ) ) select db_id ( ) as database_id , o . [type] as object_type , i . [type] as index_type , p . [data_compression] , COUNT_BIG ( distinct p . [object_id] ) as NumTables , COUNT_BIG ( distinct CAST ( p . [object_id] as VARCHAR ( 30 ) ) + '|' + CAST ( p . [index_id] as VARCHAR ( 10 ) ) ) as NumIndexes , ISNULL ( px . [IsPartitioned] , 0 ) as IsPartitioned , IIF ( px . [IsPartitioned] = 1 , COUNT_BIG ( 1 ) , 0 ) NumPartitions , SUM ( p . [rows] ) NumRows from sys .
```

partitions p inner join sys . objects o on o . [object_id] = p . [object_id] inner join sys . indexes i on i . [object_id] = p . [object_id] and i . [index_id] = p . [index_id] outer APPLY (select x . [object_id] , 1 as [IsPartitioned] from sys . partitions x where x . [object_id] = p . [object_id] group by x . [object_id] having MAX (x . partition_number) > 1) px where o . [type] not in (@0 , @1) group by o . [type] , i . [type] , p . [data_compression] , px . [IsPartitioned]

Database: AdvDest20240317

QueryId: 244

Script:
(@0 varchar (8000)) select db_id ( ) as database_id , sm . [is_inlineable] as InlineableScalarCount , sm . [inline_type] as InlineType , COUNT_BIG (*) as ScalarCount , COUNT_BIG (case when sm . [definition] like '%getdate%' or sm . [definition] like '%getutcdate%' or sm . [definition] like '%sysdatetime%' or sm . [definition] like '%sysutcdatetime%' or sm . [definition] like '%sysdatetimeoffset%' or sm . [definition] like '%CURRENT_TIMESTAMP%' then 1 end) as ScalarCountWithDate from [sys] . [objects] o inner join [sys] . [sql_modules] sm on o . [object_id] = sm . [object_id] where o . [type] = @0 group by sm . [is_inlineable] , sm . [inline_type]

Database: NewDB20241029

QueryId: 245

Script:
SELECT db_id ( ) AS database_id, o.[type] AS object_type, i.[type] AS index_type, p.[data_compression], COUNT_BIG (DISTINCT p.[object_id]) AS NumTables, COUNT_BIG (DISTINCT CAST (p.[object_id] AS VARCHAR (30)) + '|' + CAST (p.[index_id] AS VARCHAR (10))) AS NumIndexes, ISNULL (px.[IsPartitioned], 0) AS IsPartitioned, IIF (px.[IsPartitioned] = 1, COUNT_BIG (1) , 0) NumPartitions, SUM (p.[rows]) NumRows FROM sys.partitions p INNER JOIN sys.objects o ON o.[object_id] = p.[object_id] INNER JOIN sys.indexes i ON i.[object_id] = p.[object_id] AND i.[index_id] = p.[index_id] OUTER APPLY (SELECT x.[object_id], 1 AS [IsPartitioned] FROM sys.partitions x WHERE x.[object_id] = p.[object_id] GROUP by x.[object_id] HAVING MAX (x.partition_number) > 1) px WHERE o.[type] NOT IN ('S', 'IT') GROUP BY o.[type] ,i.[type] ,p.[data_compression] ,px.[IsPartitioned]

Database: AdventureWorks

QueryId: 246

Script:
SELECT db_id ( ) as database_id, sm.[is_inlineable] AS InlineableScalarCount, sm.[inline_type] AS InlineType, COUNT_BIG (*) AS ScalarCount, COUNT_BIG (CASE WHEN sm.[definition] LIKE '%getdate%' OR sm.[definition] LIKE '%getutcdate%' OR sm.[definition] LIKE '%sysdatetime%' OR sm.[definition] LIKE '%sysutcdatetime%' OR sm.[definition] LIKE '%sysdatetimeoffset%' OR sm.[definition] LIKE '%CURRENT_TIMESTAMP%' THEN 1 END) AS ScalarCountWithDate FROM [sys].[objects] o INNER JOIN [sys].[sql_modules] sm ON o.[object_id] = sm.[object_id] WHERE o.[type] = 'FN' GROUP BY sm.[is_inlineable], sm.[inline_type]

Database: newdbemo3099

QueryId: 246

Script:
SELECT db_id ( ) AS database_id, o.[type] AS object_type, i.[type] AS index_type, p.[data_compression], COUNT_BIG (DISTINCT p.[object_id]) AS NumTables, COUNT_BIG (DISTINCT CAST (p.[object_id] AS VARCHAR (30)) + '|' + CAST (p.[index_id] AS VARCHAR (10))) AS NumIndexes, ISNULL (px.[IsPartitioned], 0) AS IsPartitioned, IIF (px.[IsPartitioned] = 1, COUNT_BIG (1) , 0) NumPartitions, SUM (p.[rows]) NumRows FROM sys.partitions p INNER JOIN sys.objects o ON o.[object_id] = p.[object_id] INNER JOIN sys.indexes i ON i.[object_id] = p.[object_id] AND i.[index_id] = p.[index_id] OUTER APPLY (SELECT x.[object_id], 1 AS [IsPartitioned] FROM sys.partitions x WHERE x.[object_id] = p.[object_id] GROUP by x.[object_id] HAVING MAX (x.partition_number) > 1) px WHERE o.[type] NOT IN ('S', 'IT') GROUP BY o.[type] ,i.[type] ,p.[data_compression] ,px.[IsPartitioned]

Database: AdventureWorks

QueryId: 252

Script:

```
/*AI-DBA*/
SELECT COUNT ( 1 )  AS [NONEMPTY]
FROM [AdventureWorks].[Sales].[SalesOrderDetail]
WHERE LEN ( CONVERT ( NVARCHAR ( MAX ) ,[UnitPrice] ) ) > 0 AND LEN ( CONVERT ( NVARCHAR ( MAX ) ,[UnitPrice] ) )
IS NOT NULL ;
```
