Documentation	Performance Analysis for **prod-lg-ci-mssql-mi-1.8b554cd229a5.database.windows.net** Server
Author	Mohamed Tharif
Version	V 1.0
Date of documentation	03-07-2025

Introduction

GeoPITS is currently managing 1000+ SQL databases remotely for our global customers.

We are trusted by more than 100 clients across the globe. With 75+ database experts, our team is working on creating a minimal studio dedicated to building meaningful business processes and extensive support models with which we help companies build, manage, optimize and leverage data to empower their business. We are also a certified Microsoft Partner. Combined with an understanding of business goals and effective project management, we deliver efficient SQL server solutions.

Analysis Report

Azure SQL Database analysis was performed by DB team to identify whether SQL instances are configured as per the industry standards and follow the SQL best practices. Analysis took place at Bangalore office. The output of this analysis report would give you complete insight into current SQL Server instance configuration. The report also provides recommendations to achieve better performance as per Microsoft standard and current workload.

Technical Environment

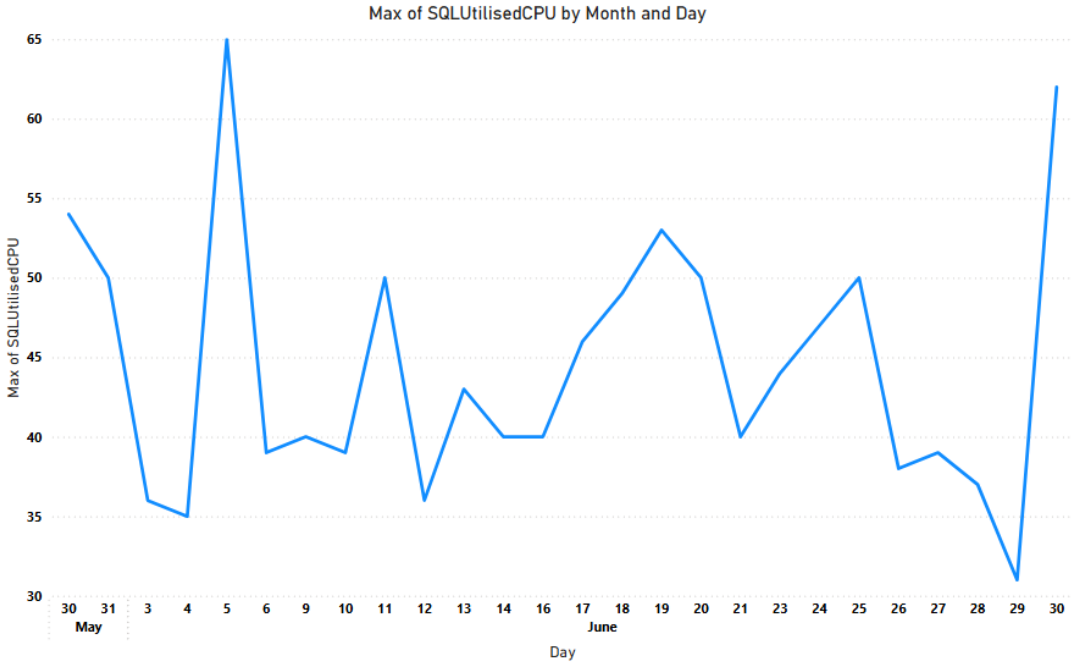
Property	Value
SQL Server Collation	**SQL_Latin1_General_CP1_CI_AS**
SQL Server Edition	**SQL Azure**
SQL Server Machine Name	**prod-lg-ci-mssql-mi-1.8b554cd229a5.database.windows.net**
Database Name	**DBLgLens, DBLoanguard, DBLoanguardHistory**
SQL Server Clustered	**NO**
SQL Server Version	**12.0.2000.8**
SQL Server Version Level	**RTM**
CPU count	**16**
RAM	**87GB**
Configured Mem GB	**82GB**
Availability Group	**NO**
In-Memory Optimization	**YES**

1.Performance metrics and Analysis

1.1. CPU Utilization by SQL SERVER

The CPU utilization over the period was generally moderate, with the majority of observations (857 times) recorded at 30%. Utilization at 35% was seen 282 times, followed by fewer instances at higher levels—40% (70 times) and 45% (23 times). Peak CPU usage reached a maximum of 65%, but only on two occasions. This pattern indicates consistent and controlled CPU consumption, with occasional moderate spikes that are unlikely to cause significant performance degradation.

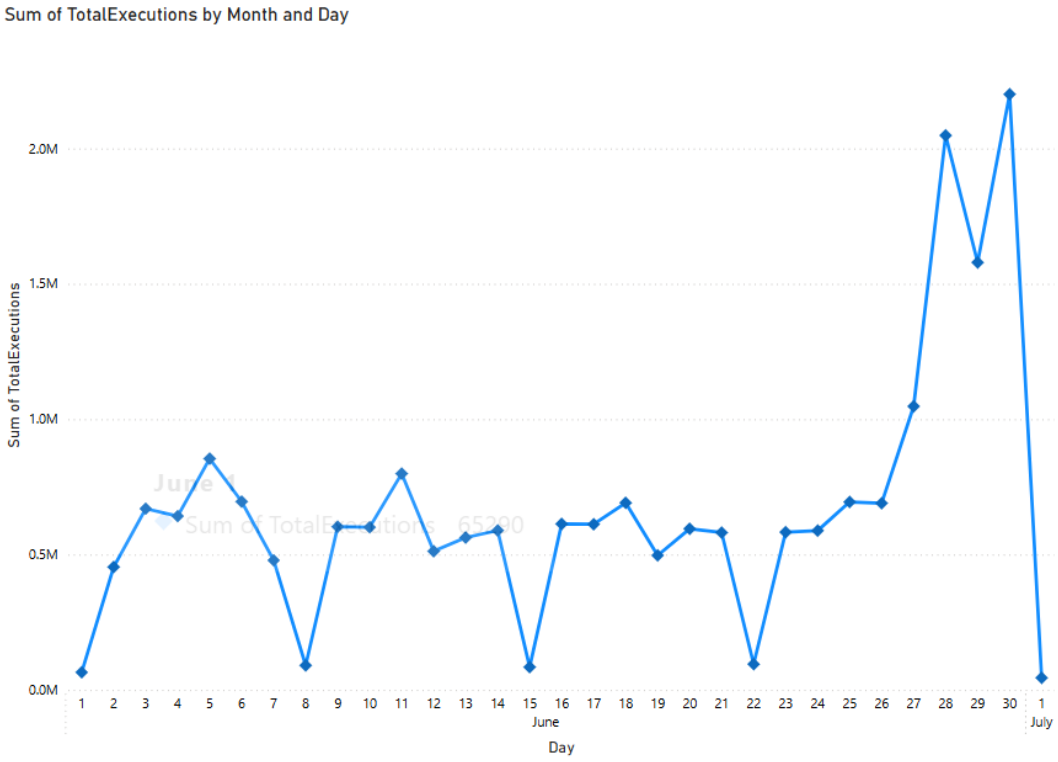
Month	Average of SQLUtilisedCPU
May	
30	37.70
31	36.08
June	
30	40.00
5	35.91
18	35.70
25	35.42
19	35.25
17	35.04
13	34.77
20	34.71
11	34.66
27	34.56
16	34.39
24	34.13
6	34.08
23	34.06
14	33.40
10	33.21
21	33.16
3	33.14
26	33.13
9	33.06
4	33.00
28	32.90
12	32.50
29	31.00



1.2. Total Number of Execution per day:

Between June 1 and July 1, query executions showed fluctuating patterns. Executions peaked on June 30 at over 2.2 million, following other high-load days like June 28 (2.04 million) and June 29 (1.57 million), suggesting sustained heavy activity during month-end processing. A noticeable dip occurred on June 22 (95k), June 15 (84k), and June 8 (90k), indicating reduced usage on weekends or non-business days. Overall, the system handled over half a million executions per day on most weekdays, pointing to consistent workload levels across the database during this period.

Year	Month	Day	Sum of TotalExecutions
2025	July	1	44998
2025	June	1	65290
2025	June	2	454322
2025	June	3	669733
2025	June	4	642088
2025	June	5	854416
2025	June	6	695768
2025	June	7	478972
2025	June	8	90760
2025	June	9	603051
2025	June	10	600984
2025	June	11	799102
2025	June	12	512480
2025	June	13	562950
2025	June	14	589330
2025	June	15	84573
2025	June	16	613384
2025	June	17	612027
2025	June	18	691236
2025	June	19	497174
2025	June	20	595394
2025	June	21	581278
2025	June	22	95463
2025	June	23	582888
2025	June	24	588505
2025	June	25	694455
2025	June	26	689842
2025	June	27	1047685
2025	June	28	2048175
2025	June	29	1579037
2025	June	30	2200324



1.3. Top Queries and Wait Types

Top Consuming Queries by Physical Reads:

TotalPhysicalReads	execution_count	AvgPhysicalReads	AvgElapsedTime	AvgCPU	QueryText
3194784	1	3194784	390837280	51768121	select MAX(CREATEDON) as NewWatermarkvalue from dbo.BANKSTATEMENTTRANSACTIONDETAILS
2838003	3	946001	276348144	24897470	select MAX(CREATEDON) as NewWatermarkvalue from dbo.BKPBANKSTATEMENTTRANSACTIONDETAILSVALIDA
2811274	3	937091	276285110	25244425	select MAX(COALESCE(MODIFIEDON, '1970-01-01 00:00:00.000')) as NewWatermarkvalue2 from dbo.BKPBANKS
1524324	1	1524324	159730485	36521553	select MAX(COALESCE(MODIFIEDON, '1970-01-01 00:00:00.000')) as NewWatermarkvalue2 from dbo.BANKSTA
1350235	1	1350235	148813191	35774265	select MAX(CREATEDON) as NewWatermarkvalue from dbo.BANKSTATEMENTTRANSACTIONDETAILSVALIDATE

Top Consuming Queries by Duration:

TotalDuration	execution_count	AvgDuration	TotalPhysicalReads	AvgPhysicalReads	AvgElapsedTime	AvgCPU	Query Text
45699860044	9961	4587878	571973	57	4587878	4023902	insert into @tblFinalSetOfRules(DocumentRuleM
16954638768	383778	44178	0	0	44178	38811	select @TotalNoofSFTPApplicationsPendingforDataEntry = count(distinct
16950550223	383778	44167	0	0	44167	38826	select @TotalNoofAPIApplicationsPendingforDataEntry = count(distinct APN.ApplicationNo) from
16843512528	383778	43888	0	0	43888	38804	select @TotalNoofApplicationsPendingforDataEntry = count(distinct
15518153891	59	263019557	3	0	263019557	735	WAITFOR(RECEIVE conversation_handle, service_contract_name,

Top Consuming Queries by CPU Time:

TotalCPUTime	execution_count	AvgCPUTin	Query
40082088672	9961	4023902	insert into @tblFinalSetOfRules(DocumentRuleMessagesId,DocValMessageId
14900694647	383778	38826	select @TotalNoofAPIApplicationsPendingforDataEntry = count(distinct APN.ApplicationNo) from @AppCountsModified
14894950505	383778	38811	select @TotalNoofSFTPApplicationsPendingforDataEntry = count(distinct APN.ApplicationNo) from @AppCountsModified
14892165709	383778	38804	select @TotalNoofApplicationsPendingforDataEntry = count(distinct APN.ApplicationNo) from @AppCountsModified
7439496681	8732	851980	insert into @AppCountsModified select
1370554973	11913	115047	insert into @tblFinalSetOfRules(DocumentRuleMessagesId,DocValMessageId

Key Observations:

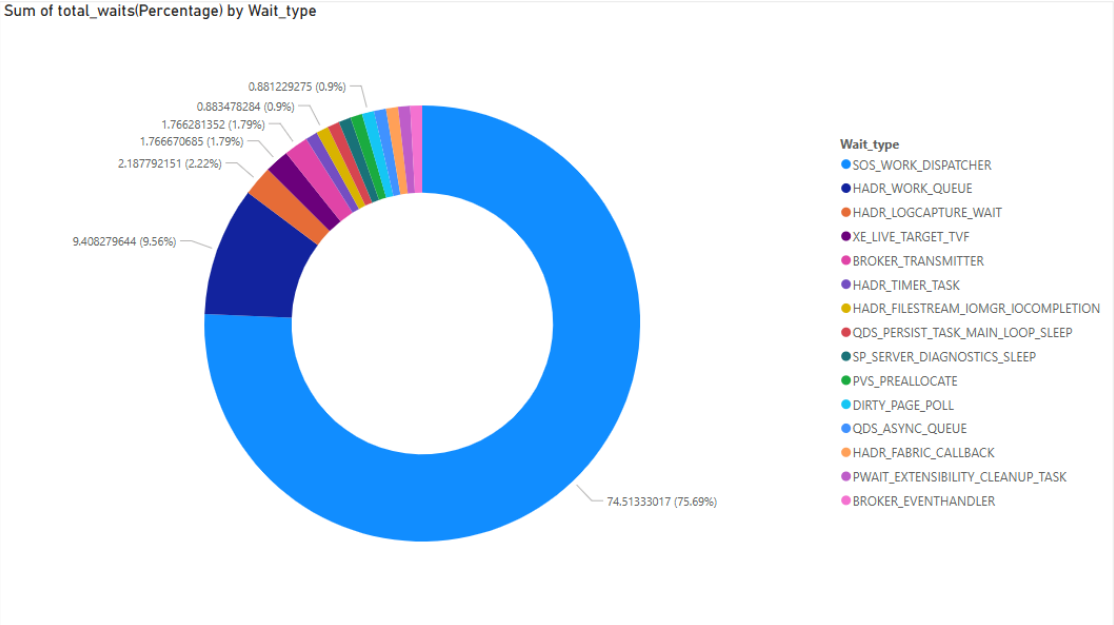
- Several queries have extremely high execution counts (up to 383K) and long durations, indicating frequent and slow operations.
- Some queries show very high physical reads (over 3 million), causing significant disk I/O pressure.
- CPU consumption is heavy, with average CPU time per execution reaching over 4 million microseconds.

Top 15 Wait Types:

The wait statistics reveal that SOS_WORK_DISPATCHER is the dominant wait type, accounting for 74.51% of total wait time, which typically indicates idle CPU scheduler threads and suggests that there is no CPU performance bottleneck. Notable waits include HADR_WORK_QUEUE (9.41%) and HADR_LOGCAPTURE_WAIT (2.19%), both associated with Always On availability group operations, hinting at overhead in log capture and secondary synchronization. Other background waits like XE_LIVE_TARGET_TVF, BROKER_TRANSMITTER, and QDS_PERSIST_TASK_MAIN_LOOP_SLEEP each contribute around 1.76% to 0.88%, reflecting background diagnostics, broker messaging, and query store persistence activity. These are generally benign unless they grow over time. Overall, the system shows healthy CPU availability.

Wait_type	Wait_time_sec	Waiting_tasks_count	total_waits(Percentage)
SOS_WORK_DISPATCHER	14537597.93	6406518	74.51333017
HADR_WORK_QUEUE	1835561.32	7111010	9.408279644
HADR_LOGCAPTURE_WAIT	426839.635	1624610	2.187792151
XE_LIVE_TARGET_TVF	344678.57	24361	1.766670685
BROKER_TRANSMITTER	344602.611	2671456	1.766281352
HADR_TIMER_TASK	172538.373	380469	0.88435578
HADR_FILESTREAM_IOMGR_IOCOMPLETION	172367.173	343244	0.883478284
QDS_PERSIST_TASK_MAIN_LOOP_SLEEP	172338.489	2873	0.883331262
SP_SERVER_DIAGNOSTICS_SLEEP	172203.907	21962	0.882641454
PVS_PREALLOCATE	172195.489	593389	0.882598307
DIRTY_PAGE_POLL	171928.39	1570917	0.881229275
QDS_ASYNC_QUEUE	171382.01	8705	0.878428772

HADR_FABRIC_CALLBACK	171003.627	589	0.876489347
PWAIT_EXTENSIBILITY_CLEANUP_TASK	171000.718	96	0.876474437
BROKER_EVENTHANDLER	169720.527	86	0.869912741



3.Index Fragmentation:

The index fragmentation analysis across multiple databases reveals varied levels of index fragmentation. In the **DBLoanGaurd** database, a substantial number of indexes exhibit fragmentation, with 9 indexes having fragmentation greater than 90%, 3 between 80-90%, and 1 between 60-70%. Additionally, there are 2 indexes in the 50-60% range, 5 in the 40-50% range, 7 between 30-40%, 10 between 20-30%, 14 between 10-20%, and a significant 211 indexes with minimal fragmentation (0-10%).

In the **DBLoanGaurd_History** database, 11 indexes fall within the 0-10% fragmentation range, indicating low fragmentation levels.

The **DBLgLens** database shows 3 indexes with 0-10% fragmentation, 1 index between 30-40%, and 1 index with fragmentation above 90%, which may require immediate attention.

It is important to note that several highly fragmented indexes—particularly those above 30% fragmentation—may also have high page counts, potentially impacting query performance by increasing I/O operations. Targeted index maintenance, such as rebuilds for high fragmentation and reorganizations for moderate fragmentation, is recommended to enhance database performance and reduce resource overhead.

Database_Name	Fragmentation_range	Index_count
DBLoanGaurd	0-10%	211
	10-20%	14
	20-30%	10
	30-40%	7
	40-50%	5
	50-60%	2
	60-70%	1
	80-90%	3
	Greater than 90	9

DBLoanGaurd_History	0-10%	11
DBLgLens	0-10%	3
	30-40%	1
	Greater than 90	1

4.Missing Index:

The audit noted that several databases could benefit from additional indexing to enhance query performance. While not currently critical, the lack of key non-clustered or covering indexes may result in slower query execution, especially on tables with a high number of rows and elevated scan-to-seek ratios. This performance impact may be due to missing indexes, insufficient covering strategies, or outdated statistics. Implementing the recommended indexes and maintaining up-to-date statistics can significantly improve efficiency and ensure optimal database responsiveness.

Database name	Table_name	create_index_statement	User seeks	Last user seek	Avg user impact
DBLgLens	applicationDocuments	CREATE INDEX [IX_applicationDocuments_ApplicationNo, LoanguardId_] ON [dbo].[applicationDocuments] ([ApplicationNo], [LoanguardId])	1697	6/27/25 12:48 PM	70.42
DBLgLens	applicationDocuments	CREATE INDEX [IX_applicationDocuments_ApplicationNo, LoanguardId_] ON [dbo].[applicationDocuments] ([ApplicationNo], [LoanguardId]) INCLUDE ([DocumentTypeId], [LGDocumentName], [DocumentName], [LGDocId], [DocumentDescription], [DocumentStatus], [Status])	1154	6/27/25 12:48 PM	70.38
DBLoanguard	Applications	CREATE INDEX IX_Applications_ScoreReportStatusAppStatus ON ([ScoreReportStatus], [AppStatus]) INCLUDE ([ApplicationNo], [ProductId], [ClientId], [ModifiedOn], [ReverseAPIStatus], [CallBackPostServiceStatus], [CallBackPostAttemptedOn])	2820	6/30/25 12:38 AM	100

DBLoanguard	Applications	CREATE INDEX IX_Applications_AppStatusCreateMode_ClientId ON ([AppStatus], [CreateMode], [ClientId]) INCLUDE ([ProductId], [CreateStatus], [ScoreStatus])	2585	6/30/25 12:40 AM	80.45
DBLoanguard	AadharCard	CREATE INDEX IX_AadharCard ON ([ApplicationNo], [LoanguardId], [DocumentIndex])	9256	6/30/25 12:11 AM	99.95
DBLoanguard	ITRValidate	CREATE INDEX IX_ITRValidate ON ([LoanguardId], [DocumentIndex])	1259	6/29/25 6:33 PM	99.94
DBLoanguard	TextractProcessInfo	CREATE INDEX IX_TextractProcessInfo ON ([DocValMessageId])	8701	6/30/25 12:11 AM	99.94
DBLoanguard	DocumentOriginalityCheck	CREATE INDEX IX_DocumentOriginalityCheck ON ([DocValMessageId])	5570	6/29/25 8:43 PM	99.9
DBLoanguard	SalaryTransactionDetails	CREATE INDEX IX_SalaryTransactionDetails ON ([SalaryTransactionID], [HeadType]) INCLUDE ([HeadValue], [HeadName], [YTDHeadValue])	3396	6/29/25 8:40 PM	99.9
DBLoanguard	PanCard	CREATE INDEX IX_PanCard ON ([ApplicationNo], [LoanguardId], [DocumentIndex])	5741	6/30/25 12:11 AM	99.77
DBLoanguard	MSEBBillValidate	CREATE INDEX IX_MSEBBillValidate ON ([ApplicationNo], [LoanguardId]) INCLUDE ([MSEBName], [MSEBAddress], [MSEBDate], [MSEBConsumerNo], [MSEBBillingUnit], [DocumentIndex], [MSEBBillingAmount])	2316	6/29/25 10:44 PM	99.72

DBLoanguard	VoterIdCardValidate	CREATE INDEX IX_VoterIdCardValidate ON ([ApplicationNo], [LoanguardId]) INCLUDE ([VoterIdCardNo], [VoterIdCardHolderName], [VoterIdCardHolderAddress], [VoterIdCardDateOfIssue], [VoterIdCardNoBackSide], [DocumentIndex], [VoterCardBirthDate])	2316	6/29/25 10:44 PM	99.61
DBLoanguard	VehicleRCValidate	CREATE INDEX IX_VehicleRCValidate ON ([ApplicationNo], [LoanguardId]) INCLUDE ([VehicleRCNumber], [VehicleOwner], [DocumentIndex], [VehicleChassisNumber], [Hypothecation], [RegistrationDate], [EngineNo], [Address], [VehicleModel], [MftgYear])	2316	6/29/25 10:44 PM	99.47

Table2: Missing Index Impact

5.Unused Index

Unused indexes in SQL Server refer to indexes that are not being utilized by queries. These indexes consume storage space and can impact the performance of data modification operations (such as INSERT, UPDATE, and DELETE) as well as the overall database performance.

Database Name: **DBLoanGaurd**

Table_Name	Index_Name	UserSeeks	UserScans	User Updates
dbo.DocumentValidatingMessages	DVDocIdDocAppSts	0	0	85991
dbo.DocumentValidatingMessages	DValDoAppSts	0	0	85759
dbo.DocumentValidatingMessages	DocAppStsApcntIdAppId	0	0	85759
dbo.TimeAnalysis	TADIdUId	0	0	73549
dbo.TimeAnalysis	TAUID	0	0	73549
dbo.TimeAnalysis	TimeAnaDocIdUserId	0	0	73549
dbo.Applications	ApnCreatStaAppNoProdIdAppDtAppSt	0	0	9993
dbo.Applications	AppnProdCliAppNoAppDtAppSts	0	0	6921
dbo.Applications	APPLication_CreateMode	0	0	2820
dbo.SalaryTransactionDetailsValidate	SalTraDetValHeadType	0	0	474
dbo.BankNarrationCategories	IndBNarrCNarrLables	0	0	229
dbo.bkp_BankNarrationCategories	IndBNCNarrLables	0	0	0

Table3: Unused Index Impact

6.Disk I/O Performanace:

The disk latency analysis of the SQL Server instance reveals significant I/O performance issues, especially in databases such as **DBLoanguard**, **DBLoanguardHistory**, and **msdb**. These databases show very high average total latencies—above 100ms—with **DBLoanguard** experiencing **122ms** for reads and **111ms** overall. This suggests substantial delays in data retrieval operations. In contrast, **tempdb** performs optimally with consistently low latencies (under 2ms), confirming that temporary objects and sort operations are not contributing to the I/O bottleneck. Additionally, high average bytes per read in certain databases indicate possible full table scans, likely caused by missing or suboptimal indexes.

Database_name	Physical Name	Size of Disk	Average Read latency	Average Write latency	Average Total Latency	Average Bytes Per Read	Average Bytes Per Write
DBLoanguard	https://wasdprodince1aprsmi446.blob.core.windows.net/managedserver-0272052c-1b52-45d6-91ee-87bd091f976f/data/f9a03225-9566-4c62-ad72-6b71534322c2_1.mdf	360897	122.4	43.2	111.8	68326	35517
DBLoanguardHistory	https://wasdprodince1aprsmi446.blob.core.windows.net/managedserver-0272052c-1b52-45d6-91ee-87bd091f976f/data/2221aa4c-06d9-4736-819c-89362a6bb4b8_1.mdf	43927	106.4	25	104.2	91980	15882
msdb	https://wasdprodince1aprsmi446.blob.core.windows.net/managedserver-0272052c-1b52-45d6-91ee-87bd091f976f/data/msdb_1.mdf	304	43.1	36.3	37.7	42236	10793
DBADB	https://wasdprodince1aprsmi446.blob.core.windows.net/managedserver-0272052c-1b52-45d6-91ee-87bd091f976f/data/726e9ebc-9cb8-4647-90c3-25f7f156ea3c_1.mdf	112	20.8	41	36.8	98066	18215
DBLgLens	https://wasdprodince1aprsmi446.blob.core.windows.net/managedserver-0272052c-1b52-45d6-91ee-87bd091f976f/data/0a5bb2b7-079a-42f2-beb2-94053ae604df_1.mdf	5200	8.3	37	17.7	82404	33656
DBLoanguard	https://wasdprodince1aprsmi446.blob.core.windows.net/managedserver-0272052c-1b52-45d6-91ee-87bd091f976f/data/f9a03225-9566-4c62-ad72-6b71534322c2_2.ldf	8816.25	81.7	4.8	5.1	3876748	17227
DBLgLens	https://wasdprodince1aprsmi446.blob.core.windows.net/managedserver-0272052c-1b52-45d6-91ee-87bd091f976f/data/0a5bb2b7-079a-42f2-beb2-94053ae604df_2.ldf	136	8.8	4.6	4.8	536451	22688
msdb	https://wasdprodince1aprsmi446.blob.core.windows.net/managedserver-0272052c-1b52-45d6-91ee-87bd091f976f/data/msdb_2.ldf	65.25	12.3	4.5	4.6	570693	4659
DBADB	https://wasdprodince1aprsmi446.blob.core.windows.net/managedserver-0272052c-1b52-45d6-91ee-87bd091f976f/data/726e9ebc-9cb8-4647-90c3-25f7f156ea3c_2.ldf	104	7.4	3.7	3.8	275148	9074
DBLoanguardHistory	https://wasdprodince1aprsmi446.blob.core.windows.net/managedserver-0272052c-1b52-45d6-91ee-87bd091f976f/data/2221aa4c-06d9-4736-819c-89362a6bb4b8_2.ldf	2179.9375	6.8	3.3	3.6	101520	8606
tempdb	C:\SFApplications\Worker.CL.WCOW.SQL22_App18\work\data\tempdb_14A307E1-326A-444B-B1BF-3DC428F4A532.ndf	272	0.9	1.5	1.5	56850	16309
tempdb	C:\SFApplications\Worker.CL.WCOW.SQL22_App18\work\data\tempdb_C3DD8DD0-AB3B-4437-9E3F-07B87C73A7BA.ndf	272	1	1.2	1.2	56782	17398
tempdb	C:\SFApplications\Worker.CL.WCOW.SQL22_App18\work\data\tempdb_230ED559-9C9F-4CA7-BD76-B17EB3CE430D.ndf	272	1	1	1	55895	16684
tempdb	C:\SFApplications\Worker.CL.WCOW.SQL22_App18\work\data\tempdb_70CE94E6-D15D-4B0A-BAFC-051A0F151AB2.ndf	272	1	1	1	56285	16956
tempdb	C:\SFApplications\Worker.CL.WCOW.SQL22_App18\work\data\tempdb_CADD0982-9F46-4A13-91F7-3C1789A96259.ndf	272	0.9	1.1	1	56056	17081
tempdb	C:\SFApplications\Worker.CL.WCOW.SQL22_App18\work\data\tempdb_71462BE5-F668-4B88-84A7-61CCBBAE3EE9.ndf	272	0.9	0.9	0.9	56125	17747
tempdb	C:\SFApplications\Worker.CL.WCOW.SQL22_App18\work\data\tempdb_5FAF6303-A8E5-478D-8C8C-A3D57FBF177B.ndf	272	1	0.9	0.9	56243	16835
tempdb	C:\SFApplications\Worker.CL.WCOW.SQL22_App18\work\data\tempdb_E8886635-A091-4F60-8C55-EE291D3EAF2B.ndf	272	0.9	0.9	0.9	56770	16376
tempdb	C:\SFApplications\Worker.CL.WCOW.SQL22_App18\work\data\tempdb_F072D6B3-B66B-4A59-80CF-9DF3093A7756.ndf	272	1	0.9	0.9	56114	16555
tempdb	C:\SFApplications\Worker.CL.WCOW.SQL22_App18\work\data\tempdb_637DA76F-D460-4B30-8860-DC5C6A05C088.ndf	272	1	0.9	0.9	56911	17274
tempdb	C:\SFApplications\Worker.CL.WCOW.SQL22_App18\work\data\tempdb_9D561CCE-D9B9-4ED0-A26D-DFBE1BCA98EF.ndf	272	1	0.9	0.9	56601	16976
tempdb	C:\SFApplications\Worker.CL.WCOW.SQL22_App18\work\data\tempdb.mdf	272	0.9	0.9	0.9	54267	15005
master	https://wasdprodince1aprsmi446.blob.core.windows.net/managedserver-0272052c-1b52-45d6-91ee-87bd091f976f/data/replicatedmaster_1.mdf	5.1875	4.9	0.4	0.5	19728	14460
tempdb	C:\SFApplications\Worker.CL.WCOW.SQL22_App18\work\data\templog.ldf	1040	0.5	0.4	0.4	65161	61251
model	https://wasdprodince1aprsmi446.blob.core.windows.net/managedserver-0272052c-1b52-45d6-91ee-87bd091f976f/data/managed_model_1.mdf	8	0.5	0.3	0.4	141259	10146
master	https://wasdprodince1aprsmi446.blob.core.windows.net/managedserver-0272052c-1b52-45d6-91ee-87bd091f976f/data/replicatedmaster_2.ldf	6.125	0.3	0.3	0.3	10240	10387
model	https://wasdprodince1aprsmi446.blob.core.windows.net/managedserver-0272052c-1b52-45d6-91ee-87bd091f976f/data/managed_model_2.ldf	8	0.1	0.2	0.2	167253	9286

7. Cumulative Recommendations

7.1 Query Performance & Wait Types

- Optimize queries by ensuring they are selective and avoid using SELECT *; retrieve only the necessary columns.
- Evaluate and enhance indexing to reduce high physical reads contributing to PAGEIOLATCH_SH waits.
- Increase SQL Server memory allocation to improve data caching and reduce disk I/O.
- Adjust the server's parallelism settings by setting **MaxDOP** to **4** and increasing the **Cost Threshold for Parallelism** to **50**, to control excessive parallel query execution.

7.2 Index Optimization

Index Fragmentation

- Rebuild indexes with fragmentation greater than 30%, especially those associated with large page counts to improve performance.
- Reorganize indexes with moderate fragmentation (10%–30%) as part of routine maintenance.

Missing Indexes

Implementing recommended missing indexes will significantly reduce disk I/O, addressing one of the key performance bottlenecks observed during analysis.

Unused Indexes

Identify and remove persistently unused indexes, especially on write-intensive tables, to lower maintenance overhead and enhance DML performance.

7.3 Disk I/O Performance

- Optimize indexing and refine queries that trigger full table scans.
- Implement missing indexes and apply selective filters to minimize read/write latency.
- Monitor latency metrics and consider storage improvements if high stall times persist, especially for files with consistently high read/write delays.