IDERA® | Database Tools Whitepaper

FIVE REASONS WHY SQL SERVER HEALTH CHECKS ARE LIFESAVERS

BY JEREMY KADLEC

CONTENTS

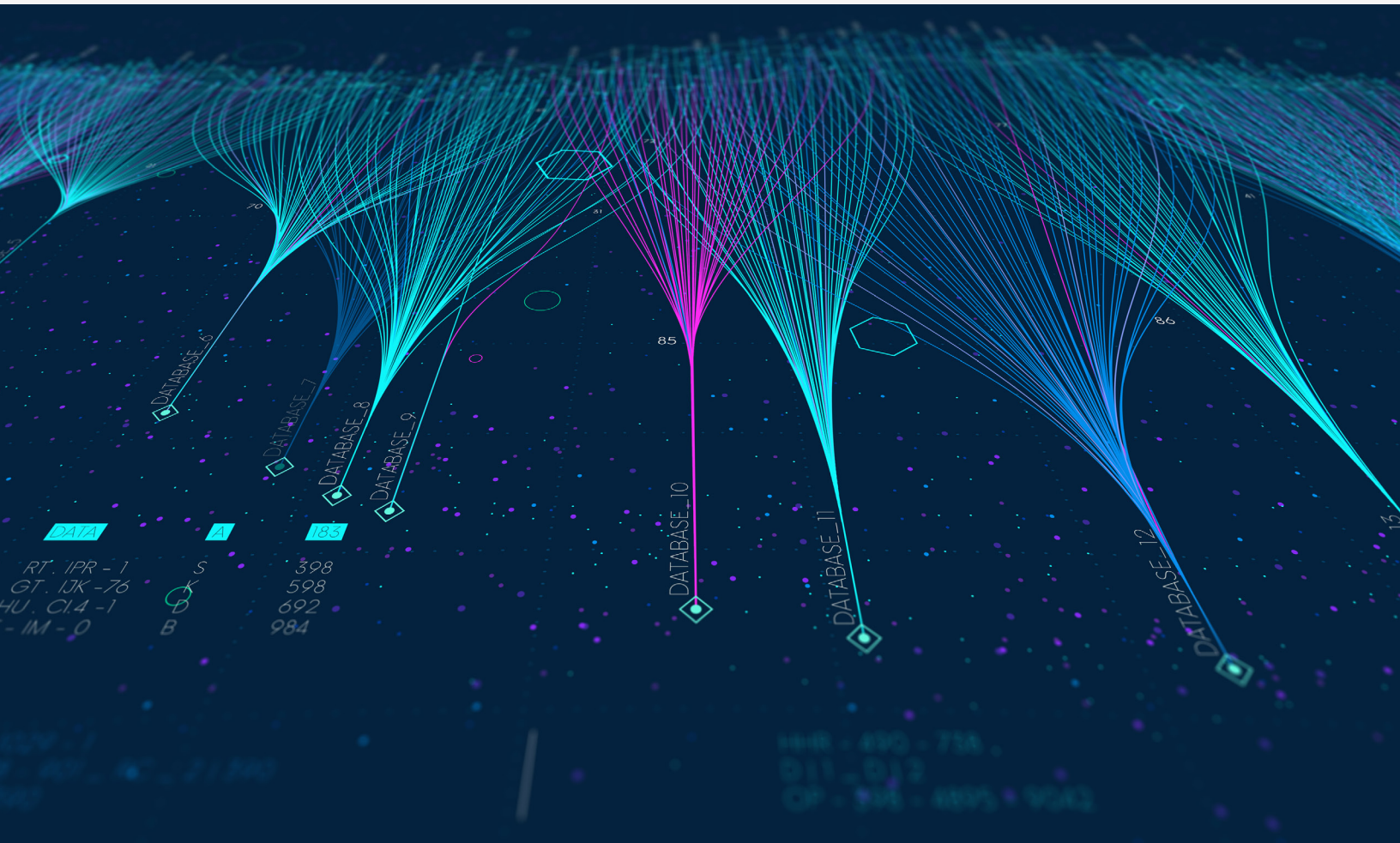
INTRODUCTION

originally a niche offering more than ten years ago has evolved into a mainstream must-have for the SQL Server community. This white paper digs into the common questions about health checks and then deliver a checklist along with some scripts to help you perform your own SQL Server health checks.

Further, this whitepaper intends to address five keys areas that could be considered life-saving. What I mean by life-saving is that they will help you grow, show your value to the organization, and keep your organization running smoothly.

The SQL Server health sanity check

- Avoid backup blunders
- Stay out of the news and secure your SQL Servers
- Make SQL Server maintenance beneficial
- Make performance predictable



WHAT IS A SQL SERVER HEALTH CHECK?

In one word: Inspection. Although, based on my more than ten years of performing SQL Server health checks, no two health checks are the same. It is a matter of analyzing an environment, enumerating issues, and working towards corrective action. However, there are some critical common items you need to address whether you are building new servers, inheriting servers, improving the checks on your existing environment, or reviewing a SQL Server instance that has not had an inspection in a long time.

I can remember one customer distinctly that we worked with for several years. Let us call him John. All the time, John would say "You do not know what you do not know." Sometimes it was said with a smile, and other times it was out of sheer frustration. In many respects, this could be the impetus for a SQL Server health check.

Call it what you want, but a health check is an inspection that is meant to get you a no-nonsense status of your current environment. Often these inspections uncover issues in your environment; other times, they confirm the root cause of an issue and provide concrete recommendations to resolve the issue.

A health check should:

- Validate the architecture, database design, or code for a process.
- Determine problematic code that is causing slowdowns, locking, blocking, and frustrated users.
- Prepare for upgrades of SQL Server, Windows, Storage, or the Application itself.
- Outline considerations when moving to the cloud, virtualization, or new hardware.
- Enumerate operational issues impacting the business and spinning wheels of team members.
- Identify roadblocks to capacity and scalability issues.
- Ensure your SQL Server environment is adequately secured, and your customers are protected.

From these items, and possibly more, you should establish your SQL Server health check goals as you begin to inspect your environment.

1. THE SQL SERVER SANITY CHECK

Getting your arms around an environment is no small feat. You need to inventory and understand the environment to start the process of identifying where you have issues. You also need to understand the SQL Server instance in the context of the infrastructure, application, and business. Let us start down that path of data collection for your SQL Server environment:

Points of contact	
Business owner	
User community	
IT operations	
IT development	

Application information	
Application name(s)	
Application priority	
Application version	
Application architecture	
Application language(s)	
Operating hours	
User location	
Overall server service level agreement (SLA)	
Maintenance window	

Once you have an understanding of the overall context of the SQL Server instance, now you need to take it down a notch and begin to understand the technical aspects:

Infrastructure	
Datacenter	cloud location	
Physical server	virtualized	
Server name	
Purpose: dev, test, user acceptance testing (UAT), production	
IP address	
Windows domain	
Server make and model	
CPU: quantity, speed, cores, hyperthreading	
Storage configuration: volume, size, speed	
Memory: such as quantity, speed	

Windows	
Operating system version	
Service pack	
Patching schedule	
Backup application	
Backup schedule	
Virus protection application	

SQL Server	
Instance count and names	
SQL Server version and edition	
SQL service accounts	
SQL Server port	
SQL Server protocols	
SQL Server character set and sort order	
SQL Server integration services configurations	
SQL Server analysis services configurations	
SQL Server reporting services configurations	
SQL Server databases	
SQL Server backup schedule: full	
SQL Server backup schedule: transaction log	
SQL Server maintenance window	
Failed SQL Server agent jobs	
Problematic SQL Server error log entries	

SQL Server databases	
Database location and size	
Transaction log location and size	
Database owner	
Recovery model	
Compatibility level	
Number of connected users	

As you begin to collect this information, you also need to do a sanity check on the servers. Consider these items as you perform your inspection:

Windows

- Does the server have too much or too little CPU, memory, and disk?
- Does the drive layout make sense?
- Is the version of Windows and the service pack up to date?

SQL Server

- Is the version of SQL Server and the service pack up to date?
- Are the SQL Server configurations (that is, MAXDOP, tempdb, Max Memory) appropriate?
- Are there numerous failed SQL Server agent jobs?
- Are databases not being backed up, including the system databases?
- Is that assumption that system databases will not need to be restored?
- Have the backups been tested?
- Are the transaction logs proportionally too large as compared to the size of the database?
- Is Maintenance running during incorrect periods? Is there no maintenance at all?
- Is the security lacking with many logins with system administrator (SA) rights, shared logins, and no logging?

2. AVOID BACKUP BLUNDERS

Will there ever come a day when the recommendation will be “You do not need to back up your SQL Server databases?” That is not a reality today. For many companies, backups are the last line of defense if an issue occurs. Some companies have a sophisticated high availability and disaster recovery solution. For other companies, backups are the first and last line of defense.

Although just about every company has a SQL Server database backup plan, is it bullet-proof? More often than not, the plan, implementation, and testing can be lacking in several areas. Here are a few that I have seen over the years. Having a rock-solid backup plan is a lot more than just issuing full backups daily. It is a matter of being able to recover from a failure. You need to consider the potential failures to determine ways to mitigate the risks.

SQL Server backup plan or pretend?

Having a rock-solid backup plan is a lot more than just issuing full backups daily. It is a matter of being able to recover from a failure. You need to consider the potential failures to determine ways to mitigate the risks

Make sure your SQL Server Database Backup Plan includes the following:

- System and User Databases: Back up all of your system- and user-defined databases except tempdb and ReportServerTempDB. Getting your instance up and running without the system databases is possible, but it is time-consuming and stressful when a disaster strikes.
- Make It Automatic: Make sure your backup code will automatically include new databases that are added to the instance.
- Risk versus Budget: Work with the business to understand the budget available for backups and the corresponding impact to the business based on the downtime. That is a real risk versus reward system. If the business only provides you sufficient storage for a single set of backups, let them know much how data, person-hours, and downtime the business will suffer. If the business understands the risks, they may be able to find some budget.
- Backup Location: Backup onsite: Good or bad? Backup offsite: Good or bad? Backing up onsite and offsite: probably your best bet to get back online locally as quickly as possible or prevent a more substantial issue from putting you out of business altogether.
- Monitoring: It is incredible to see processes failing for days, weeks, or months without anyone having any idea until it is too late. If your SQL Server database backups are your last line of defense, then make sure they are on your radar, and you know they are successful daily.

- Point in Time: Keep in mind SQL Server has more than just full backups. Depending on your needs, be sure to include differential and transaction log backups to restore to a point in time.

- Protection: Full database backups are a point in time version of your data, so protect them. Keep in mind a hacker does not need to compromise your SQL Server to access your data. They can do so with the backups. So protect your backup directories, encrypt the backup and treat your backups with the same care you treat the online production database.

SQL Server backup plan or pretend?

Here is a simple script to issue a full backup of all your SQL Server databases:

```
DECLARE @name VARCHAR(50) -- database name
DECLARE @path VARCHAR(256) -- path for backup files
DECLARE @fileName VARCHAR(256) -- filename for backup
DECLARE @fileDate VARCHAR(20) -- used for file name
DECLARE @cmd1 VARCHAR(500) -- string together backup command
-- specify database backup directory
SET @path = 'C:\Backup\' -- Change this based on your environment
-- specify filename format
SELECT @fileDate = CONVERT(VARCHAR(20),GETDATE(),112) +
REPLACE(CONVERT(VARCHAR(20),GETDATE(),108),':','')
DECLARE db_cursor CURSOR FOR
SELECT name
FROM master.dbo.sysdatabases
WHERE name NOT IN ('TempDB', 'reportservertempdb') -- database exclusion
OPEN db_cursor
FETCH NEXT FROM db_cursor INTO @name
WHILE @@FETCH_STATUS = 0
BEGIN
   SET @fileName = @path + @name + '_' + @fileDate + '.BAK'
   SET @cmd1= 'BACKUP DATABASE ' +'[' + @name + ']' +
   ' TO DISK = ' + char(39) + @fileName + char(39) +
   ' WITH INIT, COMPRESSION;'
   EXEC (@cmd1)
   FETCH NEXT FROM db_cursor INTO @name
END
CLOSE db_cursor
DEALLOCATE db_cursor
```

For more details, visit Simple script to backup all SQL Server databases.

SQL Server Backup Testing

Testing backups should be simple, straightforward, and habit. To start testing your backups, pull some out of rotation and restore them as a new name, that is, tmp_YOURDATABASE to a Development or Test instance. Be sure to test backups on a weekly or monthly basis from both onsite and offsite file shares. To kick it up a notch, build restore logic to restore backups daily to a development or test instance and issue DBCC CHECKDB commands on them to validate both the backups and the integrity of the database. More information on SQL Server Maintenance is coming up soon.

Do not overlook this critical step and discover that your databases are not restorable after a disaster strikes. The same is valid for working through the process of getting a SQL Server instance up and running. Practice this process in advance of an unforeseen issue. Be proactive and get your scripts together, so you do not fumble around as the downtime clock is ticking.

SQL Server health checks more than a report

A valuable health check is more than a report. It is education and the sharing of knowledge. With every health check we perform at Edgewood Solutions, we strive to educate our customers. The SQL Server community has an abundance of valuable information (tips, tricks, blog posts, articles, videos, forum posts, and more) that should be shared to improve the knowledge of the community as a whole. With resources like these, hopefully, everyone is better educated, and we can collectively grow. Find the issues in your environment and work to correct them to show your value to the organization.

3. STAY OUT OF THE NEWS AND SECURE YOUR SQL SERVERS

It seems like security breaches are making the mainstream news regularly. There are many great sets of legislation intended to protect the general public from having sensitive data compromised and to prevent other financial issues. Whether that legislation impacts you or not, you need to take a common-sense approach to ensure your SQL Servers are secure. Here is a baseline set of items to consider:

- Minimal permissions: users and apps
- Code not susceptible to SQL injection
- Encrypt data and backups
- Secure backups onsite and offsite
- Passwords: strong and changed often
- Auditing: key activity on servers
- No shared logins
- Windows-based authentication
- Obfuscate data for lower environments
- Escalation procedures

In terms of permissions, let us take a look at two essential scripts to start to identify vulnerabilities. First is reviewing which logins have system administrator (SA) rights in SQL Server. SA is an abbreviation for System Administrator, and this login in SQL Server has permissions to perform any operation on the instance. Second, we will review which logins have database owner (DBO) rights. DBO is an abbreviation for Database Owner, and this user has permission to perform any operation in the database. Although there are more permissions in SQL Server that can be problematic, let us start with an inspection of these.

Who has system administrator (SA) rights?

In the script below, we are going to determine the logins with SQL Server system administrator (SA) rights:

```
SELECT SP1.[name] AS Login, SP2.[name] AS Permission
FROM sys.server_principals SP1
JOIN sys.server_role_members SRM
ON SP1.principal_id = SRM.member_principal_id
JOIN sys.server_principals SP2
ON SRM.role_principal_id = SP2.principal_id
WHERE SP2.[name] = 'sysadmin'
ORDER BY SP1.[name]
```

	Role	Login
1	sysadnin	NT Service\MSSQL$SQL19
2	sysadnin	NT SERVICE\SQLAgent$SQL19
3	sysadnin	NT SERVICE\SQLWriter
4	sysadnin	NT SERVICE\SQLWinmgmt
5	sysadnin	QUICK\
6	sysadnin	sa

For more details, visit Auditing SQL Server Permissions and Roles for the Server.

In the script below, we are going to determine the logins with SQL Server system administrator (SA) rights:

```
DECLARE @LoginName sysname
DECLARE @sql NVARCHAR (2000)
CREATE TABLE ##tmp_xp_logininfo
(AccountName varchar(128) NOT NULL,
Type varchar(10) NOT NULL,
Privilege varchar(10) NOT NULL,
MappedLoginName varchar(128) NOT NULL,
PermissionPath varchar(128) NOT NULL)
DECLARE cur_Loginfetch CURSOR FOR
SELECT [name]
FROM master.sys.server_principals
WHERE TYPE = 'G'
AND Name NOT IN ('NT SERVICE\MSSQLSERVER', 'NT SERVICE\SQLSERVERAGENT')
OPEN cur_Loginfetch
FETCH NEXT FROM cur_Loginfetch INTO @LoginName
WHILE @@FETCH_STATUS = 0
BEGIN
```

```
INSERT INTO ##tmp_xp_logininfo
EXEC xp_logininfo @LoginName , 'members'
FETCH NEXT FROM cur_Loginfetch INTO @LoginName
END
CLOSE cur_Loginfetch
DEALLOCATE cur_Loginfetch
SELECT DISTINCT(LEFT(AccountName, 35)) AS LoginName,
LEFT(PermissionPath, 35) AS GroupName
FROM ##tmp_xp_logininfo
WHERE Privilege = 'admin'
ORDER BY 1
DROP TABLE ##tmp_xp_logininfo
```

For more details, visit Auditing Windows Groups from SQL Server.

Who has database owner (DBO) rights?

To find out which users have database owner (DBO) rights, we can execute the following code against each database.

```
EXEC sp_helprolemember 'db_owner';
```

Are all SQL Server health checks just a validation of best practices?

Generally, the best practice is a recommendation that is valuable for the masses. What comes with experience is understanding how to apply best practices based on your specific environment. In theory, particularly best practices should just work. However, the reality is you need to evaluate and test the best practice to validate that the best practice is appropriate in your environment. One traditional best practice is never to use cursors. It has been documented in articles, videos, and heard at conferences.

A cringe-worthy statement is an absolute rule that includes words like always or never. It is undoubtedly the case when it comes to cursors. Would I recommend a cursor for basic create, read, update, and delete (CRUD) operations? No. Are set-based options the way to code your logic the majority of the time? Yes, but I have been in circumstances when I have built cursor-based logic that is faster to build and easier to understand than massive set-based logic that is buggy, locks up numerous tables, and is difficult to troubleshoot.

So are health checks just a validation of best practices? No. They should not be. The best practices can serve as a portion of the foundation for the process, but you should customize your health check for your unique environment.

4. MAKE SQL SERVER MAINTENANCE BENEFICIAL

Do you change the oil in your car or the water filter in your refrigerator? Do you fertilize your lawn? That is maintenance, and if it is a priority in other parts of your life, it needs to be a priority for your SQL Server environment.

- Be smart about these SQL Server maintenance items:
- DBCC CHECKDB
- INDEX REBUILD and REORGANIZE
- UPDATE STATISTICS

DBCC CHECKDB

DBCC CHECKDB is responsible for validating the integrity and consistency of a database. As mentioned in the previous section, one simple means of performing DBCC CHECKDB without impacting your production environment can be accomplished by restoring the database to a Test or Development instance and then issuing the DBCC CHECKDB commands. If you do not have the opportunity to use a test or development instance with sufficient resources, be sure to run the DBCC CHECKDB command against all of your databases regularly during a low-usage period.

Here is a sample script to issue DBCC CHECKDB against all of your databases:

```
SET NOCOUNT ON
-- 1a - Declaration Statements for all variables
DECLARE @DatabaseName varchar(128)
DECLARE @CMD1 varchar(8000)
-- 1b - Database Variables
DECLARE @DatabaseListLoop int
DECLARE @DatabaseListTable table
(UIDDatabaseList int IDENTITY (1,1),
DatabaseName varchar(128))
-- 2a - Loop for populating the database names
INSERT INTO @DatabaseListTable(DatabaseName)
SELECT Name
FROM Master.sys.databases
WHERE Name NOT IN ('tempdb', 'reportservertempdb')
```

```
ORDER BY Name ASC
-- 2b - Determine the highest UIDDatabaseList to loop through the records
SELECT @DatabaseListLoop = MAX(UIDDatabaseList) FROM @DatabaseListTable
-- 2c - While condition for looping through the database records
WHILE @DatabaseListLoop > 0
BEGIN
 -- 2d - String together the final DBCC command
 SELECT @DatabaseName = DatabaseName
 FROM @DatabaseListTable
 WHERE UIDDatabaseList = @DatabaseListLoop
 -- 2e - Header
 PRINT '*************************'
 PRINT @DatabaseName + ' - '+ CAST(GETDATE() AS varchar(25))
 PRINT '*************************'
 -- 2f - String together the final DBCC command
 SELECT @CMD1 = 'DBCC CHECKDB ([' + DatabaseName + '])'
 FROM @DatabaseListTable
 WHERE UIDDatabaseList = @DatabaseListLoop
 -- 2g - Execute the final string to complete the DBCCs
 -- SELECT @CMD1
 EXEC(@CMD1)
 -- 2h - Footer
 PRINT '*************************'
 PRINT ''
 PRINT ''
 PRINT ''
-- 2i - Descend through the database list
SELECT @DatabaseListLoop = @DatabaseListLoop - 1
END
```

Index rebuild and reorganize

Another way to get smart about SQL Server database maintenance is by using index rebuilds and reorganizations. These commands are responsible for correcting fragmented indexes. The two main commands you need to become familiar with are ALTER INDEX REORGANIZE and ALTER INDEX REBUILD. The ALTER INDEX REORGANIZE command is an online operation that reorganizes the leaf nodes of the index to have the logical and physical index order match. However, you are unable to change any of the properties for the index. The ALTER INDEX REBUILD command internally drops and creates the index to correct the fragmentation issues, but also provides the ability to change the index properties.

Below are two sample commands:

```
ALTER INDEX { index_name | ALL } ON <object> REORGANIZE
ALTER INDEX { index_name | ALL } ON <object> REBUILD WITH (
{FILLFACTOR = fill factor
| SORT_IN_TEMPDB = { ON | OFF }
| STATISTICS_NORECOMPUTE = { ON | OFF }
| ONLINE = { ON | OFF }
| MAXDOP = max_degree_of_parallelism
| DATA_COMPRESSION = { NONE | ROW | PAGE }
[ ON PARTITIONS ( { <partition_number_expression> | <range> }
[ , ...n ] ) ]
```

Update statistics

Let us give SQL Server the information it needs to access data efficiently. The best way to do so is by updating the statistics of SQL Server. You can accomplish updating all of the statistics for a single database with a single command. Execute the command below in your desired database:

```
EXEC sp_updatestats;
```

There is one last step - review output

need to review the results for any errors. Be sure to make this a portion of your maintenance process, so a minor issue that you should have discovered does not turn into a major issue you could have prevented.

But I need it! Seriously?

“I need <fill in the blank>.” Name the latest craze in the SQL Server world, and I have heard someone say that they need it. Not because the technology is going to solve a technology or business problem, but because it is the latest and greatest SQL Server feature. I know that I am not alone in hearing these sorts of sentiments. At one of the Baltimore SQL Server Users Group meetings, we had a speaker from Microsoft mention they had a customer who said, “I need BIG Data.”

The reality in many of these situations is that someone has seen an article and now wants to stand up an entire environment to find a problem this new technology will solve. All the while, the remainder of the environment is in shambles. The lesson here is to not lose sight of finding the right solution to correct the core issues impacting the organization.

5. MAKE PERFORMANCE PREDICTABLE

Everyone has performance issues such as slowdowns, locking and blocking, and unexplainable situations. I have seen many of the same types of situations with many customers where significant time and energy is being poured into correcting harmful code, but not the worst offending code. Often the code is long-running but only runs once a day or week. Sure, it takes a few hours, and that is not ideal. However, there is code that runs thousands of times a day and takes seconds for each execution.

In the aggregate, the later set of code is more detrimental than the first. If that code were optimized to run in milliseconds, the users would be much more productive, and the SQL Server instance would require far less CPU, memory, and disk resources.

How do you identify the problematic code?

DBCC CHECKDB is responsible for validating the integrity and consistency of a database. As mentioned in the There are a few different ways to identify the problematic code. Profiler, Extended Events, and the dynamic management views are probably the most popular. Let us take a look at this simplified dynamic management view query to identify the code running most often with the highest aggregate resource count.

```
SELECT TOP 20
    GETDATE() AS “Collection Date”,
    qs.execution_count AS “Execution Count”,
    SUBSTRING(qt.text,qs.statement_start_offset/2 +1,
                (CASE WHEN qs.statement_end_offset = -1
                            THEN LEN(CONVERT(NVARCHAR(MAX), qt.text)) * 2
                           ELSE qs.statement_end_offset END -
                                      qs.statement_start_offset
                )/2
            ) AS “Query Text”,
     DB_NAME(qt.dbid) AS “DB Name”,
     qs.total_worker_time AS “Total CPU Time”,
     qs.total_worker_time/qs.execution_count AS “Avg CPU Time (ms)”,
     qs.total_physical_reads AS “Total Physical Reads”,
     qs.total_physical_reads/qs.execution_count AS “Avg Physical Reads”,
     qs.total_logical_reads AS “Total Logical Reads”,
     qs.total_logical_reads/qs.execution_count AS “Avg Logical Reads”,
     qs.total_logical_writes AS “Total Logical Writes”,
     qs.total_logical_writes/qs.execution_count AS “Avg Logical Writes”,
     qs.total_elapsed_time AS “Total Duration”,
     qs.total_elapsed_time/qs.execution_count AS “Avg Duration (ms)”,
     qp.query_plan AS “Plan”
FROM sys.dm_exec_query_stats AS qs
```

```
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) AS qt
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) AS qp
WHERE
      qs.execution_count > 50 OR
      qs.total_worker_time/qs.execution_count > 100 OR
      qs.total_physical_reads/qs.execution_count > 1000 OR
      qs.total_logical_reads/qs.execution_count > 1000 OR
      qs.total_logical_writes/qs.execution_count > 1000 OR
      qs.total_elapsed_time/qs.execution_count > 1000
ORDER BY
      qs.execution_count DESC,
      qs.total_elapsed_time/qs.execution_count DESC,
      qs.total_worker_time/qs.execution_count DESC,
      qs.total_physical_reads/qs.execution_count DESC,
      qs.total_logical_reads/qs.execution_count DESC,
      qs.total_logical_writes/qs.execution_count DESC
```

For a complete explanation, visit Collecting and Storing Poor Performing SQL Server Queries for Analysis.

This query allows you to see the actual execution plan in the last column of the result set. Just click on it and the execution plan will load in a new Management Studio window.

How do you correct the problematic code?

If you choose to use Profiler\Extended Events instead of the query above, once you have identified the code, your next step is reviewing the Execution Plan. Here is how you review the graphical execution plan:

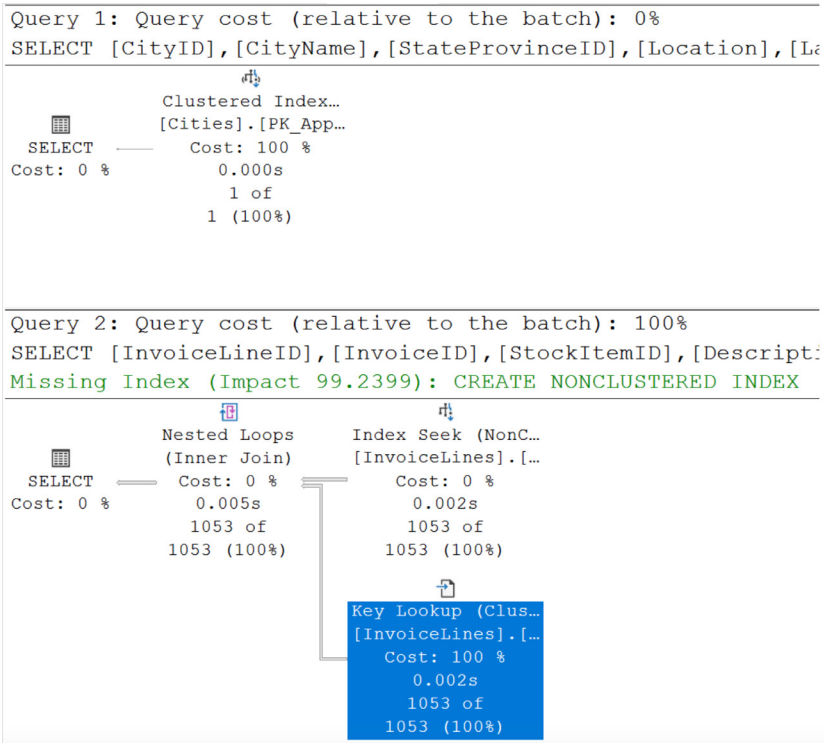
1. Open Management Studio

2. Open a New Query Window and paste your code

3. To include the Actual Execution Plan, press CTRL + M

4. To run the query, press F5

Here is a basic example:



As you review the execution plan, you want to focus on the areas that have the highest usage percentage. From there, you can determine if portions of the code need to be optimized with new or different indexes as well as see if a new coding technique is necessary. The exact solution will vary, but the execution plan should give you the insight to get started.

As a final note, you may even be surprised that the query optimizer may even tell you how to improve the code. That is the case with the bottom half of the execution plan image above. The query optimizer is noting a missing index with an impact of almost 99 percent. Management Studio even makes the code available to copy, modify, test, and implement to improve performance. It does not get much simpler than that.

ABOUT THE AUTHOR

Jeremy Kadlec is the Chief Technology Officer at Edgewood Solutions and co-founder of MSSQLTips.com, where his team solves problems for millions of SQL Server professionals around the globe. Over his 15-year career, Jeremy has served as a distinguished SQL Server consultant, author, speaker, community leader, and trainer. Microsoft has recognized Jeremy as a Microsoft Most Valuable Professional for SQL Server since 2009. He loves spending time with his family and is an avid sport fisherman.

IDERA'S SOLUTION

SQL Doctor

Performance tuning recommendations and health checks for SQL Server

SQL Doctor helps database administrators to tune SQL Server performance, security, and disaster recovery via expert recommendations in physical, virtual, and cloud environments - including managed cloud databases. Unlike its competitors, it provides display health of all SQL Servers, generation of ready-to-run SQL scripts to optimize and undo optimization, limiting of analysis to specified databases, applications, and performance categories, and real-time, as-needed and scheduled checkups.

Start for FREE