White Paper

Avoiding the Top DBA Mistakes

SQL Server Best Practices

By Tim Radney and Mike Walsh

For

Embarcadero Technologies

October 2014

INTRODUCTION

At Linchpin People, we perform a lot of WellDBA™ exams to help clients troubleshoot and optimize their SQL Server environments. This whitepaper summarizes a few of the top mistakes we find organizations making. We call them mistakes, but in most cases they are really things that DBAs just may not be aware of. The big exception to that is not properly backing up the data so that it is recoverable. As IT professionals, we should all be fully aware that we need to be able to recover data, especially databases. With that background in mind, we've focused this whitepaper on some aspects of data recovery, data consistency, maintenance operations, memory optimizations, alerting, instance tuning, power management, tempdb best practices, and baselining, that we find most frequently in our WellDBA exams. We give insights into why these areas are important and guidance on how to correct these mistakes in your environment. We hope you will find this information useful, and if you have any questions make sure to look us up on Twitter.

MISTAKE 1: BACKUPS THAT ARE NOT RESTORABLE AND DO NOT MEET SLAS

Having proper backups is critical for the safety of your data. Without proper backups, you cannot restore your database in the event your data becomes corrupt, someone accidently messes up the data, or some other disaster occurs.

Having proper backups means several things. First you must have backups that are restorable. I highly recommend having a restore validation process in place. This is typically addressed by regularly restoring a copy of your backups to a secondary location. Many organizations have a dedicated server and storage that they use to perform this function. This is a process that you can be easily automate or that requires very little manual processing.

Another part of having proper backups is having backups that can meet or exceed the service level agreement (SLA) that has been established. The SLA is this case will be for the recovery point objective (RPO) of the system. If you are taking hourly transactions log backups, the RPO for that environment would be 1 hour. This means that the acceptable risk of data loss for that environment is 1 hour. A system with an RPO of 15 minutes could expect a transaction log backup cycle of 15 minutes.

WHAT TO DO: IMPLEMENT A PLAN

When you take over a new environment, it is always a good to ensure that the current backup plan fits within what the customer or business is expecting and make any adjustments if necessary.

Having a scheduled job that runs and completes successfully each night is not an adequate validation that you have successful backups. Often times I have found that databases have been excluded from the scheduled backup job. To help protect against such surprises, you will need to have a process in place to check to make sure each database has a recent backup. You can accomplish this in several ways, however one of my favorites is to use a script that I blogged about here(http://www.linchpinpeople.com/verify-sql-server-backups/) with the database name, date of the last full backup, differential backup, and the last two transaction log backups, as well as the recovery model. With this script, you will be able to determine what type of RPO you could meet with a given backup strategy, as well as check to make sure the right types of backups are occurring based on the recovery model.

Mistake 2: Not Checking for Database Corruption

What is database corruption? The most basic way to explain database corruption is to think about the individual data pages being stored on disk becoming unreadable. In nearly all cases, this is related to problems at the I/O subsystem level. This could be due to a hardware failure, driver or firmware issue, controller, or any number of other physical or logical errors. When this happens, the data on disk becomes damaged.

Checking for database corruption is very important. Speaking to DBAs from all over the world, one of their biggest fears is finding corruption in one of their critical databases. If you do not have a process in place to check for corruption, then the likelihood of you finding out about the corruption while you can still recover the data is very slim.

When corruption is detected, I recommend that you contact your systems engineers and vendors to have them start looking at the infrastructure environment while you deal with the database corruption. You will want the underlying issue that caused this error to be corrected to help avoid future issues.

What to Do: Use CHECKDB

The best way to detect and check for data corruption is to schedule a job to run DBCC CHECKDB against your databases. This is an I/O intensive process, so schedule it to run during low activity times. Many organizations perform this checking over the weekend. Some organizations have built a process to validate their backups by restoring them to a secondary server and also perform their CHECKDB scans there, as well. This offloads the I/O impact from production to a secondary server to avoid any effect on the customer experience.

I typically utilize DBCC CHECKDB [DB_NAME] WITH NO_INFOMSGS, ALL_ERRORMSGS when running CHECKDB. I want to suppress regular messages but see any and all error messages. DBCC CHECKDB is very robust and has various options you can chose from when running the command. For more information on DBCC CHECKDB and its associated commands, you can read this article http://msdn.microsoft.com/en-us/library/ms176064.aspx.

In most cases, SQL Server will back up a corrupt database. Using WITH CHECKSUM specifies that the backup process will verify each page for checksum and torn page. This is the default behavior for compressed backups, however using backups with checksum is not a replacement for running DBCC CHECKDB.

When corruption is found, depending on the type of corruption, your backups from before the corruption occurred may be your only option to be able to recover the data. The output of the CHECKDB scan will give you some guidance on what repair options may be available. If the corruption is on a non-clustered index, then that index could be dropped and recreated to fix the corruption issue. However, if the corruption is on a clustered index, you are not as lucky. A restore of the full backup, differential, and logs is most likely your option. Or perhaps you can do a page-level restore.

You don't want to be in a situation where you find non-repairable corruption in your database if you don't have backups from before the corruption occurred. This situation would mean you now have data loss. Depending on what table the corruption is found in, you could lose critical customer data.

READER NOTE: "DBCC CHECKDB WITH REPAIR_ALLOW_DATA_LOSS" does exactly what it says: You *will* lose data. Run this only with extreme caution and when you have no other options.

MISTAKE 3: NOT RUNNING MAINTENANCE TASKS

When you're responsible for SQL Server—just like with owning a house or automobile—you need to perform certain maintenance tasks to keep SQL Server running smoothly. These maintenance tasks include index maintenance, backup history, error log cleanup, and ensuring statistics accuracy. You can automate many of these tasks within SQL Agent jobs and simply monitor them.

WHAT TO DO: INDEX MAINTENANCE

Index maintenance is an important maintenance task with SQL Server. Index fragmentation can have a negative impact on SQL Server performance. Fragmentation exists when your indexes have pages where the logical ordering does not match the physical ordering within the data file. According to a Microsoft whitepaper (http://technet.microsoft.com/en-us/library/cc966523.aspx), fragmentation can have a negative impact ranging from 13 to 460 percent, depending on the level of fragmentation and size of the environment.

You can reduce index fragmentation by reorganizing, rebuilding, or dropping and recreating the index. The two most common methods are to reorganize or rebuild the index. When an index is rebuilt, the index is essentially dropped and recreated. The index reorders the index rows in contiguous pages, and disk space is reclaimed by compacting the pages based on the existing fill factor setting. When an index is reorganized, the leaf levels of the clustered and nonclustered indexes are defragmented by physically reordering the leaf-level pages to match the logical order of the leaf nodes. This also will reclaim disk space by comparing the index based on the existing fill factor. Reorganizing uses minimal system resources compared to rebuilding.

You can determine how fragmented your indexes are by using the system function sys.dm_db_index_physical_stats. This function does not include the index name, so you might want to join it on sys.indexes on object_id in order to display the name of the index. You will need to pass the system function a couple of parameters. In the following example, I provide the DB_ID and the table I want the fragmentation stats on. Run the following script against AdventureWorks.

```
SELECT  ps.index_id, i.name, ps.avg_fragmentation_in_percent, ps.page_count
FROM sys.dm_db_index_physical_stats (DB_ID(), OBJECT_ID(N'sales.store'),
       NULL, NULL, NULL) AS ps
JOIN sys.indexes AS i ON ps.object_id = i.object_id AND ps.index_id = i.index_id;
GO
```

In the real world you will have higher page count values than I am showing in this example. The values that become important for determining when to rebuild or reorganize depend on the fragmentation level and the number of pages. A good rule of thumb is: If fragmentation is less than 5 percent, does nothing; if it's between 5 percent and 30 percent, reorganize; and if it's greater than 30percent, rebuild. For page count, an old value was 1000 pages. In my experience, that value will vary depending on the environment. These values however are a great starting point.

As you can probably imagine right now, with the number of indexes within your tables, the number of tables within your database, and the number of databases on your server, creating a process to handle all this maintenance would be very time consuming. The good news is there are a multitude of ways to automate this maintenance.

A common approach for smaller environments has been to use database maintenance plans. The issue with database maintenance plans is that your choice is either to reorganize or rebuild indexes. It is an all-or-nothing choice. If you selected to rebuild indexes for the database AdventureWorks in a database maintenance plan, then each index would be rebuilt, regardless of the amount of fragmentation. The same would occur if you selected to reorganize: Each index in the database would have its index reorganized. This creates a lot of unnecessary overhead.

It would be much better to have a process that allows you to choose whether to rebuild or reorganize depending on the amount of fragmentation and the size of the table. Such a process already exists and is widely used by data professionals across the world. Ola

Hallengren built a robust maintenance solution that you can use to perform backups and integrity checks and perform index maintenance. You can read more about this process and download the scripts at https://ola.hallengren.com/.

Regardless of the method you use to rebuild or reorganize your indexes, the important concept here is that you need to perform the action. If you are using a method that is unnecessarily rebuilding or reorganizing indexes that are not overly fragmented, consider updating your process to one that uses more advanced logic.

WHAT TO DO: KEEP UP WITH BACKUP HISTORY MAINTENANCE

SQL Server logs all backup and restore history in MSDB. By default, this data is not deleted or purged automatically. As a database professional, you need to make sure this information is cleaned out. Some third-party backup applications will do this maintenance for you if you select that option. Often times though, I find that backup history is not being purged.

Microsoft includes the stored procedure sp_delete_backuphistory. You can pass it the parameter of the oldest date to retain the history for. For example, if you want to delete all backup and restore history prior to January 1, 2014, you can run the following script:

```
USE msdb;
GO
EXEC sp_delete_backuphistory '01/01/2014'
```

Scheduling this to run in a SQL Agent job and passing it a value that meets your data retention needs is fairly easy. Most organizations use either 30, 60, or 90 days.

WHAT TO DO: CLEAN UP ERROR LOGS

The SQL Server log is a great place to find out about what is happening with your SQL Server database. Each time SQL Server is restarted, the log is recycled. The current log is renamed to errorlog.1, the existing errorlog.1 becomes errorlog2, and so on. Since SQL Server instances are not often restarted, it is very common for the SQL Server log to become large. It can be time consuming if you have to view the contents of the SQL Server log, and it contains tens of thousands of entries.

To avoid having an unmanageable SQL Server log, you can execute the system stored procedure sp_cycle_errorlog, which will recycle the log. You will want to schedule a job to execute this stored procedure to some value that meets your organizations requirements.

```
EXEC sp_cycle_errorlog
```

The default number of logs to keep is rather low, so if you begin recycling your error log each day you will lose some historical data. The way you prevent that is by increasing the maximum number of error logs. By default this value is 6 which would mean 7 total logs. You can increase this value to 99. To do so, simply right click on "SQL Server Logs" and choose Configure to increase the value.

WHAT TO DO: CHECK THE ACCURACY OF STATISTICS

Statistics are metadata that the SQL Server Query Optimizer uses to help select the execution plan for a query. The accuracy of the statistics will have direct impact on the Query Optimizer's ability to create an effective execution plan. By default, SQL Server has a feature called Auto-Update Statistics turned on. This is a great feature; however its limitation is that typically 20 percent plus 500 rows have to be updated before it will update the statistics. For small tables, this works well. However for larger table or tables with skewed data sets, this can be less than optimal.

Because of this limitation, relying on SQL Server to keep your statistics up to date might not be effective enough unless you are a small shop and don't have major data fluctuations. If you find that your statistics are out of date (numerous scripts are available on the web to help you determine the accuracy of your statistics), then you need to implement a method of updating your statistics. You can accomplish this by using a t-sql task with a custom SQLAgent job that runs sp_updatestats, (This approach works for a lot of people but could be updating stats that might have been recently updated by index maintenance.) Or you can use a free index and statistics maintenance script that was mentioned above in the section about Index maintenance. Ola Hallengren includes the ability to update statistics and avoids updating statistics on any statistics that were just updated with an index rebuild.

When you're rebuilding an index, the statistics for that index are updated. However, the column indexes are not. When you're reorganizing an index, the statistics are not modified.

MISTAKE 4: NOT PAYING ATTENTION TO MEMORY SETTINGS

By default when you install SQL Server, the min and max memory settings are not configured for your environment. The default min memory setting is 0 MB, and the max memory default setting is 2147483647 MB. That max memory setting calculates to 2 petabytes. These defaults could leave your SQL Server instance being starved for memory or starving the OS of any available memory. You can easily check to see what the values are of your memory settings by running the following script:

```
SELECT   *
FROM     sys.configurations
WHERE    configuration_id IN ( '1543', '1544' )
```

WHAT TO DO: MONITOR AND ADJUST

Many years ago when I learned about setting the min and max memory values, I was taught that a good rule of thumb for the values is 30 percent of available memory for min when the server is dedicated to SQL Server. For max you need to use some math: For a server with 16 GB of RAM or less, reserve 1 GB for the OS and 3 GB for SQL Server. For servers with more than 16 GB of RAM, use 1 GB for the OS per 8 GB of RAM. With this equation, a server with 16 GB of RAM would have 4 GB dedicated to the OS and 12 GB dedicated to SQL Server. A server with 64 GB of RAM would have 10 GB allocated to the OS and 54 GB allocated to SQL Server.

The values listed above are not set in stone. If you have heavy usage of SSIS, SSRS, SSAS, or if you have an application installed on your server, you will most likely have to dial back the max memory value a bit. How will you know if you need to decrease or increase the max memory value? You can monitor the Memory\Available MBytes performance counter. The values set by Microsoft say that this counter should always remain above 150-300 MB. For me, this value is way low. However most of the servers I work with have larger amounts of memory. I like to see the available MBytes counter closer to 1 GB. If I find a server with a value too small, I back off the max memory setting. At the same time if I encounter an MBytes counter well over 1 GB; I know I am leaving valuable memory on the table that SQL Server could be utilizing, so I increase the max memory value.

MISTAKE 5: NOT BEING AWARE OF SQL SERVER ALERTS

The overriding point here is that a happy DBA is a proactive DBA. It's very important for DBAs to maintain a proactive stance towards their SQL Server environment. Part of that is a proactive approach to reactive events.

How do you know when your server is experiencing issues? How do you know when errors are being thrown indicating that a log file is running out of space someplace? Hopefully following all of the advice here will decrease the impact of such situations, but the point is: Things happen even with the best of intentions and preparation. You want to know about those issues as soon as they occur, and you want to receive an email about it instead of a customer complaint.

Enter SQL Server alerts. While we recommend DBAs consider using third-party monitoring products to monitor their SQL Server environment, we also understand that such products aren't always in the budget. A good first step to gaining full visibility into your SQL Server instance is to create SQL Server Agent operators, set up database mail to email these operators, and then set these operators up to receive alerts when a job fails or when certain error conditions occur.

What to Do: Set Up Database Mail and Operators

The first thing you need to do is ensure Database Mail is set up appropriately and working. Once that is done, you can create SQL Agent operators that will receive the notifications. The operator can be individual users or an email distribution list that is sent to everyone responsible for the SQL Server instances.

For a detailed walkthrough on how to create these, you can see instruction details and a video on our blog here (http://www.linchpinpeople.com/how-to-set-up-sql-server-alerts/).

What to Do: Understand Agent Alerts

Next you create a set of standard alerts for SQL Server. SQL Server has various errors that can be raised for various situations. These errors have severity levels that describe the type and seriousness of an error. Errors with a severity of less than 16 typically have to do with application logic or syntax errors, or they are warnings. You don't need to be alerted about these, typically. Instead, setting up alerts for severity 16 through severity 25 errors will enable you to be notified anytime something happens that affects a database seriously (like a log file filling up), or when an instance-wide concern occurs (like serious resource issues that could affect availability of your databases on the instance).

We also recommend creating an alert for Error 825, which is severity 10 so it would not show up on the alerts starting at severity level 16. Error 825 is a warning, but it is an early sign of potential I/O issues that leads to SQL Server corruption. You can read more about this error message on Paul Randal's blog (http://www.sqlskills.com/blogs/paul/a-little-known-sign-of-impending-doom-error-825/).

Mistake 6: Accepting the Default Values for Query behavior

For small workloads, SQL Server may perform quite well with default values set for Max Degree of Parallelism and cost threshold for parallelism, however as workloads increase these values will likely begin to hinder performance.

What to Do: Understand MAXDOP and CXPACKET

SQL Server is built to allow parallel execution of queries. This has been a tremendous advantage for queries such as larger operations dealing with larger sets of data that benefit from parallel execution. No one specific type of workload **always** benefits from parallelism, but typically queries that run in reports or large scans, sorts, or joins of data are the ones that benefit most.

By default, SQL Server is configured with a Max Degree of Parallelism (MAXDOP) of "0". This setting controls the degree of parallelism – the number of CPU cores that can be used to execute a parallel region of a query. The default setting of 0 means "unlimited," or uses all of the cores available to SQL Server.

This default has rarely ever made sense. It makes less sense now in the days of 12 and 18 cores and multiple-processor machines. There is no magic answer on what number is right, but a few considerations to help out here are as follows:

1. 0 probably isn't the right setting in multi core or multi-processor environments
2. 1 probably isn't as right as often as it is recommended on the Internet
3. Microsoft says if you have more than 8, try stopping at 8 and see how it works
4. This is an online setting. You can change it without restarting SQL Server if you need to change it again in the future
5. If you have more than 4 cores total but less than 8, stopping at 4 may make sense so that not all cores can be in use for processing parallel regions of a query that has run away. If you have more than 8, stopping at 8 may make sense

You can change this setting over time as performance conditions dictate. You can dial it in by watching the performance of larger queries.

Often SQL Server users will go straight to MAXDOP when they see or experience excessive CXPACKET waits. CXPACKET waits indicate threads are waiting for other parallel threads to complete their work. This is often not the real cause of those issues, though. Often excessive CXPACKET waits are a good sign that further query tuning could be done, or they are a sign that other resource issues (for instance I/O or memory) are causing delays that are making all other threads wait.

WHAT TO DO: UNDERSTAND THE COST THRESHOLD FOR PARALLELISM

SQL Server uses the cost threshold for parallelism setting to determine when a query that could benefit from parallelism is triggered to apply parallelism.

This setting refers to the estimated subtree cost of a query. You can see this when looking at a query plan and hovering your mouse point over the first operator on the top and left most position. This shows the cost of all of the operators that fed data to the final query.

This cost is an older calculation based on the number of seconds operations took on certain hardware at a certain time during SQL Server development. The only problem is that this cost method never changed with more recent hardware or versions of SQL Server. As a result, the calculation is still based on technology around the time that SQL Server 2000 was new or in development.

The default value is 5. This may have made sense at some point, but it sometimes can trigger unnecessary parallelism, which can cause performance concerns and make queries go parallel that may not benefit as much from it.

This setting is a bit like MAXDOP in that there are some guidelines, but it is a setting best experimented with and researched for your environment. We typically like to see this at least set to 20 or 25 in most environments. In many environments, we may go higher. It is rare, however, that we see this number lowered back to 5.

This is a dynamic setting, No SQL Server instance reset is necessary. No server restart is necessary. You can explore the plan cache and see what costs look like to get a sense for your distribution. But we typically find increasing slowly and starting with 20-25 is a good approach.

This setting change, like MAXDOP, does not eliminate the need for good query tuning practices. You should still review your workloads and indexes for performance optimizations to get the most out of your environment over time.

Mistake 7: Not Understanding Power Settings

Microsoft and CPU manufacturers and server manufacturers have introduced power saving options. You notice the effects of these options in various forms, but the results typically are a slowdown of database performance when it comes to SQL Server.

At the very least, the power saving means a CPU could be under clocked, run at a slower speed for most of the time, and only ramp up when it feels a certain need for it as the CPUs become less idle. The problem here is that SQL Server typically doesn't just peg a CPU high and leave it there. It tends to need CPU increases, then not need them. And go on in a cycle like that. In SQL Server, the end result of a setting that degrades CPU performance to save some power is poorer performance. This performance degradation depends on the workload, but it is not insignificant.

There are also other settings at work in the power-saving schemes of servers nowadays. For instance, NICs can go to sleep, resulting in connectivity issues or iSCSI performance issues.

What to Do: Learn about Server Power Saving Options

The best setting for a Windows machine running SQL Server is "High Performance," not "Balanced," which is the default power scheme for many installations nowadays. Confirm the power mode settings for all servers and place them into High Performance mode.

Also several server manufacturers have BIOS settings that work in concert with or in addition to the Windows settings. Check with your system administrators to ensure that these settings are correct.

On our blog (http://www.linchpinpeople.com/power-saving-settings-killing-sql-server-performance/) we also have a video and set of instructions showing you how to check for power saving modes being enabled and how to use a tool called CPU-Z to see if you are experiencing the effect of CPUs running in a power saving mode.

MISTAKE 8: NOT UNDERSTANDING TO RIGHT SIZE TEMPDB

Often times we find that clients or users of forums are having performance issues related to tempdb. The number one issue we find is that tempdb has not been optimized from the default settings upon installation of SQL Server. When this is the case, the performance issue with tempdb is primarily with latch contention on the allocation pages.

You might now be wondering what allocation pages are. Allocation pages are pages in the data files that manage and track extent allocations. With regards to tempdb, the three allocation pages of primary concern are Page Free Space (PFS), Global Allocation Map (GAM), and Shared Global Allocation Map (SGAM).

PFS starts on page 1 and repeats every 8,088 pages, or 64 MB of data file. As the name indicates, PFS tracks the amount of free space each page has.

GAM starts on page 2 and repeats every 511,232 pages, or 4 GB of data file. GAM tracks which extents have been allocated.

SGAM starts on page 3 and repeats every 511,232 pages, or 4 GB of data file. SGAM tracks which extents are uses as shared (mixed) extents.

By default, if you have not modified tempdb or the model database, tempdb is recreated and modeled every time SQL Server Service is restarted. By default, this is a single data file and single log file, and both are in the low MB range. Since there is only one tempdb per instance, this would leave you with a single PFS, GAM, and SGAM page for the entire instance.

WHAT TO DO: LEARN HOW TO SIZE TEMPDB APPROPRIATELY

To help solve this, the current recommendation from Microsoft is to increase the number of data files for tempdb. The recommendation on the number of files is based on the number of cores your systems has, and then you adjust up if you still have contention.

The general rule is that if you have 8 logical processors or fewer, then use 1 file per core. If you have 8 or more logical processors, then start with 8 files. If you still have contention, increase the number of files in increments of 4. These files should be of equal size. If you determine that you have 4 logical processors and your tempdb workload size is 8 GB in size, then you create 4 data files of 2 GB size. You will also want to set the auto growth size of these files to a fixed size rather than percentage. (http://support.microsoft.com/kb/2154845)

Mistake 9: Not Knowing What Normal Behavior Is

Part of being proactive is keeping an eye on how your server is used and growing over time. Part of making life easier when you're brought into an emergency is knowing what "good" looks like. Part of knowing when it's time to consolidate or split off databases into their own instance is keeping on top of utilization over time.
All of these require good baselines. There are many ways to obtain baselines of your SQL server environment, from the simple roll-your-own perfmon-based approach to using a third-party tool.

What to Do: Monitor Your Environment to Know How Your System Is Performing

For now, it doesn't matter how you do it. Just start obtaining baseline metrics. You can't know when something is bad until you know what it looks like normally. There are many approaches to base lining, but the recommended approach is that we have to just get you going.

Use a free tool from Microsoft called Performance Analysis of Logs (PAL – pal.codeplex.com). This tool can create a perfmon template for SQL Server, which you can then put on your server with perfmon. After it is there, you can then run your perfmon trace to file and bring the perfmon log file down to PAL to analyze. PAL will give you a set of graphs and alerts with full details of what is going on in your environment, how SQL Server is responding, and how your resources look.

We have a video walkthrough of setting up and using PAL on our blog. (http://www.linchpinpeople.com/how-create-sql-server-baseline-using-pal/) This is the quickest way to get up and running.

Once you get familiar with the tool, get in the habit of running a baseline every quarter or month, comparing the same period of days each quarter or month. This can show how your environment trends over time, and you can see where the next bottleneck could be or get a sense of capacity planning as time goes on.

We also suggest looking into the SQL Server I/O and Wait statistics that are tracked on an instance. John Sterrett has scripts for both wait stats (http://johnsterrett.com/2013/10/08/benchmark-sql-server-wait-statistics/) and file stats (http://johnsterrett.com/2013/02/25/benchmark-sql-server-disk-latency/) on his blog, along with a full description of how to interpret the results.

Hand in hand with the PAL reports, this will show you how your server is running day over day.

Conclusion

With the information you have just read, you will be able to tune and correct many possible issues affecting your SQL Server environment. You'll also be able to take a proactive approach to monitoring and resolving any future issues.

Having proper backups and database integrity will help you sleep at night knowing you can recover your databases and have confidence that the data is good. Having optimized indexes and up-to-date statistics will help your query optimizer be more efficient and improve performance. Following the memory best practices will help ensure your OS and SQL Server are not constantly competing for memory. Configuring alerts will help you be aware of issues before certain events turn catastrophic. Making some minor tweaks to maxdop and cost threshold for parallelism can greatly improve your query performance, and changing your power settings can give much needed CPU resources back to your SQL Server instance. Making sure to right size tempdb for your environment can be the key to eliminating latch contention on tempdb. Lastly, if you are not monitoring it, then you cannot measure it. You need a baseline to know what normal behavior verses abnormal behavior is. Otherwise when someone says things are slow, how do you know if it is as slow today as it was last week?

This is by no means a comprehensive list of everything that could impact your server environment. However the mistakes we've covered are some of the top issues that we see when performing WellDBA™ exams. Getting your environment cleaned up and following the best practices outlined in this whitepaper will go a long way towards making sure you have a reliable and stable environment.

Next Steps

For more information about Embarcadero award-winning database tools and to try them for free, visit www.embarcadero.com/products/database-tools.

About the Authors

Tim Radney is a partner with Linchpin People and a data professional for a financial services company. Tim is a published author and trainer and has been in the role of systems administrator, production DBA, manager, and information security engineer over the past 16 years. Tim holds certifications across many aspects of IT and is dedicated to paying it forward within the SQL Server community. Microsoft recently recognized Tim as a SQL Server MVP. He serves as the chapter leader for the Columbus, GA SQL Users Group, is a PASS regional mentor for the Greater Southeast USA, and is a regular speaker at national and regional conferences and SQL Server events. Follow Tim on twitter @tradney or his blog http://www.timradney.com.

Mike Walsh is a partner with Linchpin People and our WellDBA™ Practice Lead. He is an experienced SQL Server professional with 15 years of experience and has worked with every version of SQL Server since version 6.5. He has been in the role of DBA, developer, architect, and performance lead but always leans on his DBA and performance tuning experience. He loves providing training and coaching to clients and colleagues and has been recognized as a SQL Server MVP by Microsoft for four years in a row, since 2011. He speaks at national and regional conferences and SQL Server events on DBA, performance tuning, and professional development topics. Mike blogs regularly on SQL Server topics. Follow Mike on twitter @mike_walsh or his blog http://www.straightpathsql.com.