IDERA® | SQL Server Whitepaper

5 DBA MISTAKES THAT CAN COST YOU YOUR JOB

BY ROBERT DAVIS

TABLE OF CONTENTS

Introduction

1 A Lack of Good Backups

2 Undetected Corruption

3 Disabling Page Verification

4 Giving Away Elevated Permissions

5 Sharing Passwords

Summary

Don’t Despair!

INTRODUCTION

Everyone makes mistakes. Database administrators (DBAs) are no exception to that rule. When DBAs make mistakes, they are often the ones who find it first and have to fix it. In the past, I have been accused of never making mistakes, but in reality I've merely gotten really good at finding and fixing my mistakes before anyone else notices.

There are some mistakes that are often deemed inexcusable for a DBA to make. These are the things that can cost businesses irreparable damage. DBAs will often be coerced into allowing things that they know they are wrong. When things go bad, it is almost always the DBA who is held responsible for failing to protect the data—no matter who signed off on the risky decisions.

If a manager insists we do something that we know is wrong, it is incumbent upon us to educate that manager about why it is the wrong thing to do—and press for doing the right thing. While there are certain scenarios with a lot of grey areas that could justify allowing a risky behavior, there are 5 things that a DBA should never allow to happen.

These 5 mistakes outlined below endanger the recoverability, the integrity, and the security of the data that we protect. These 5 mistakes may be deemed unforgivable, and they could cost you your job.

1 A LACK OF GOOD BACKUPS

A DBA's number one priority is backups. I can't stress this enough. If everything else goes wrong, you should always be able to rely on backups as your last resort during any disaster event. Too many times I meet people who are experiencing a disaster who discover that their backup job has been failing, or that all of their backups contain corruption and can't be restored. This can lead to taking drastic measures to preserve what data can be saved and then dealing with the repercussions of losing data. Failing to have reliable backups can cost you your job.

In a perfect data world, DBAs should be aware of the requirements for recovery time and data loss. These two requirements are often referred to as Recovery Time Objective (RTO) and Recovery Point Objective (RPO), and are used to design a disaster recovery plan (and the resulting backup plan) to support recovery.

In the event of a disaster, RTO is the length of time you are allowed to be down and RPO is the amount of data you are allowed to lose. The goal of a DBA should always be to have as close to zero data loss as possible. When devising a backup plan for disaster recovery, you should assume that all other levels of protection have failed—restoring from backups is your last line of defense.

When you are down to your last line of defense, the last good backup you have is the amount of data you will lose. This will help you determine your backup frequency. If you have a low data loss requirement, you will need to take frequent backups. The only backup type you should be doing with a high frequency is log backups. This generally means running in either full recovery model or bulk-logged recovery model.

Having reliable backups means more than just having backups, though. It means knowing that the backups can be restored and knowing how to restore them. This is where testing backups comes in to play. As a bare minimum, you should be using the BACKUP VERIFYONLY command to test that a backup is restorable.

In addition to verifying a backup, I also highly recommend using the CHECKSUM option with all backups and restores. The CHECKSUM option performs extra checks when possible (more on this later) to determine if a database is corrupted. If the extra checks find data corruption, the backup operation will fail and alert you that corruption exists. Additionally, it will perform a checksum of the entire backup file, which will help detect if the backup file itself is corrupted after it is created.

The big advantage to the final CHECKSUM operation for the entire backup file is that if corruption has occurred in the file, a restore of the file will fail right away, rather than when it encounters the corruption. This is especially important for very large databases where a restore operation can take hours. If the backup file has been corrupted and it has the extra checksum value in the header, the checksum will be recalculated at the start of the restore operation and will fail very quickly. Finding out at restore time that your backup file is corrupted is very bad, but finding out that it is corrupted after a couple of hours of running the restore is far worse.

The best thing a DBA can do to ensure that the backups are restorable is to test them by performing actual restores. Ideally, I prefer a combination of automatically restoring backups—to ensure they are valid—and running disaster drills, where a downtime scenario is practiced and DBAs run through the entire recovery process. Planning and practice are the keys to achieving your recovery requirements when an actual disaster occurs.

I like to say that the first and last things a DBA should do are backups. If I inherit a new server or environment, the first thing I do is make sure all servers have backups running successfully. Later, I will revisit the backup situation and develop a disaster recovery plan based on realistic RPO and RTO requirements. I could forgive someone once for not catching the second part of that, but not having reliable backups is an unforgivable mistake if you are a DBA. If disaster strikes and a DBA does not have reliable backups, their job is at risk.

Data is valuable and important.
And it is your responsibility as a DBA
to protect it.

Failing to do so can ultimately
cost you your job.

2 UNDETECTED CORRUPTION

Corruption can happen to anyone at any time—there is no way to guarantee that it will not occur. As DBAs, all we can do is to find corruption as quickly as possible and to be prepared to deal with it when it happens. Corruption is but one of the disaster events I referred to in the previous section, but it is critical enough to warrant discussion on its own. Most DBAs are not highly experienced in dealing with corruption because it's not something you normally encounter on a regular basis. When it does happen, one of the key things you can do to defend against it is to find it quickly. Failing to do so can result in not being able to recover from corruption without data loss—sometimes massive data loss.

SQL Server provides many built-in methods for detecting corruption. The CHECKSUM option for backups and restores was discussed in the previous section, and I will talk about the page verification options in the next section. In this section, I want to discuss running basic integrity checks using the DBCC CHECKDB commands—or other DBCC CHECK commands.

At a bare minimum, DBAs should regularly run integrity checks of all databases. The part that is often overlooked is the "doing it regularly" part. Recovering from corruption without data loss and with minimum downtime often means using backups for a partial or full restore of the database. However, if you back up a database that is corrupted (without using the CHECKSUM option), you get a corrupted backup. If corruption exists undetected for a long time, you are less likely to have a good, uncorrupted backup that is restorable to the current time.

Even with long-term storage on tape, you may have to go way back in time to find a backup free of corruption, and it is not likely that you will have all the log files from that time, which would allow you to restore to bring the database current. This could mean a very large amount of data loss. At the very least, having to resort to restoring from offsite storage is likely to cause an extended outage. Worst case scenario, a lot of (or all) data may be lost. These are the kinds of events that cause companies to go out of business.

Despite the attractiveness of simply running DBCC CHECKDB WITH REPAIR_ALLOW_DATA_LOSS, automatically repairing corruption should also be considered a last resort option. This option fixes corruption by deallocating corrupt pages, but once data pages are gone, they're done for good. This is the scenario many DBAs face when they failed to find corruption quickly enough and no longer had a valid backup without corruption. This is a severe case of DBA neglect, and failing to detect corruption in a timely fashion can cost you your job.

3 DISABLING PAGE VERIFICATION

SQL Server provides some very critical built-in functionality for detecting corruption called "page verification." When you're running a query and get those annoying alerts about corruption, page verification is what is detecting that corruption. Best of all, the best page verification option is the default setting for new database. To protect databases with this option, all you have to do is nothing.

Checksum page verification is also what enables the extra checks I talked about when using the CHECKSUM option for backups and restore. When this option is enabled, SQL Server calculates the checksum over a whole page and writes the value to the page header. When a page is read into memory, the checksum value is calculated again and compared to the value in the header. If the two checksums do not match, the page is corrupted and an 824 error is raised alerting the user of corruption. If the checksum values exist in the header of a page, the value is recalculated during backup and restore operations when using the CHECKSUM option to detect corruption on any data page protected by checksums.

There are three page verification options available: checksum, torn page detection, and none. There is very negligible overhead in using the page verification options, but not using them can have a disastrous impact on your database. The only time you have to take any action to ensure you are protected by page verification is when you upgrade a database from SQL Server 2000 or earlier. CHECKSUM is the preferred page verification option, and has been the default option since SQL Server 2005. In SQL Server 2000, the default option was torn page detection and, prior to torn page detection, page verification was not a setting you could configure. In other words, "none" has NEVER been a default option for page verification. If you have a database that has page verification disabled, someone has changed it to that setting.

In my opinion, if a DBA is setting page verification options to none, it is either an act of deliberate sabotage or incompetence. Either way, that DBA should not be allowed to administer SQL Server. This mistake can and should cost you your job as it requires willful action to occur.

If disaster strikes and a DBA does not have reliable backups, **their job is at risk.**

4 GIVING AWAY ELEVATED PERMISSIONS

Security is very critical in SQL Server and is often overlooked by many SQL professionals. As administrators of the databases, we are protectors and guardians of the data they contain. We cannot protect the data if we don't control access to it.

Giving higher permissions to people or applications than they justifiably need, can result in data loss or data theft, both of which can damage the business in a very public way. Giving away too much access can also impact performance, as users are not always careful with the types of queries they run. Additionally, when governmental compliance policies (e.g., PCI, HIPAA, SOX, etc.) are applicable, failing to follow access guidelines can also result in penalties including steep fines or even incarceration.

This is why we practice the Principle of Least Privilege. This principle states that only the lowest level of permissions required and justified are to be granted. The key words to keep in mind are "required" and "justified". DBAs field a lot of requests for production access that must be denied. When DBAs grant access to production databases, they should be able to defend why they granted those permissions. You should be able to say why it is required and why it is justified.

You will often be told that the non-production environments do not have the functionality or the data required to support the queries that they want to run. Or that it is harder to perform the tests or investigations that they need to run. If non-production is inadequate, the correct approach is to fix non-production. And, if doing your job the correct way is too hard, find an easier job. There are plenty of highly-skilled professionals doing in non-production what many will tell you can't be done except in production.

I was once asked to temporarily take over operations for a large business intelligence application. Shortly after taking over operations for the application, I discovered that the developers had SysAdmin access to production and deployed new builds themselves. I immediately removed all access from production for everyone in development and then notified them of the removal. There were a lot of unhappy people in the short-term, but they also realized what I did was the right thing to do and surprisingly put up very little argument against it.

In the three months I ran operations for the application, the engineering team showed they had the capacity to be a real enterprise-class team. Unfortunately, six months after a permanent DBA was found to run their operations, I learned that the team had reverted to their old ways and once again had SysAdmin access and were doing their own deployments. The DBA defended this by saying that they wouldn't take no for an answer and that they forced him to do it. This shows you the difference a strong DBA can make in this regard. You can make a difference.

It is the responsibility of every DBA to judge the appropriateness of granting permissions and making sure the principle of least privilege is followed. This is a mistake that I have seen DBAs terminated for making.

5 SHARING PASSWORDS

This final mistake applies to all professionals who work with SQL Server, not just DBAs. Sharing passwords bypasses audit controls and can be used to cover up who really committed an act that violated the integrity of the data. If I were a disgruntled employee who wanted to steal data or sabotage the company, I would first try to learn the password for a different account so it couldn't be traced back to me.

When I learn that someone has shared their password for an account, whether their own account, a service account, or a SQL login, I immediately disable and deny all access to the account. This can have a severe business impact, but it is necessary in order to protect the data from theft and sabotage.

I was running operations for a critical application that processed large quantities of credit card transactions—not only for our own line of business, but also for other lines of businesses. Processing and storing credit card transactions places you under strict PCI (payment card industry) compliancy regulations. Since we were already PCI compliant in a company with many diverse lines of business, it was simpler and smarter to let other lines of businesses tie into our payment processing APIs and avoid redesigning the wheel.

One of the downstream BI teams wanted to deploy reports that used our BI database as source. They had not yet set up a service account, and asked to use one of our service accounts until they got their own as some high-level executives were expecting the reports to be available that day. This would have required sharing the password for one of our service accounts. I told them there was absolutely no way I would risk our payment processing servers being shut down by an auditor because we were caught sharing passwords. The executive would have to wait.

The point of this story is there is no reason that could justify sharing passwords— not even a high-level executive wanting immediate access. If we had shared the password, and were caught doing so, the executive is not the one who would have lost his or her job when the company lost millions of dollars due to not being able to process credit cards. The executive is not the one who would be held accountable for knowing better. If you do something you know is wrong just because a manager or executive tells you to do so, you will be held accountable for not making that person aware of why it is wrong. It is your job on the line for doing it, not theirs.

SUMMARY

As I've iterated multiple times above, Database Administrators are the protectors and guardians of the data in the databases. I could throw out lots of clichés like "data is king" or "data is power", but they are all really just pointing out the same thing: data is valuable. Data is valuable and important. It is your responsibility as a DBA to protect it. Failing to do so can ultimately cost you your job.

The 5 mistakes I have outlined above all center around protecting the data. These mistakes jeopardize the recoverability, integrity, and security of the data. The mistakes you want to avoid are:

- A lack of good backups
- Undetected corruption
- Disabling page verification
- Giving away elevated permissions
- Sharing passwords

Making any of these mistakes could easily land you in hot water and even cost you your job. Protect yourself by protecting your data. I cannot guarantee that you won't get into trouble by taking a hard line on these topics when management wants you to compromise, but that's not a job I would want to keep anyway. Even if management signs off on risky behaviors, if it results in public embarrassment for the company, having failed to protect your data could hurt your ability to find a new job as well.

DON'T DESPAIR!

The good news is, if you have made these mistakes and no major issues have occurred as a result, it is not too late to correct it. Now that you know the 5 mistakes DBAs make that can cost you your job, you can stop making them. You must take a proactive stance on correcting any problems you have in this area.

If you do not currently know the status and recoverability of your backups, look into that now. Make sure all of your databases are getting backed up. Make sure that the backups are getting verified and tested. Add the CHECKSUM option to all backup and restore commands so that you can avoid creating corrupted backups without knowing it. And most of all, make responding to backup failures a critical priority.

Institute a regular process for checking the integrity of your databases. Create a SQL job or a maintenance plan, or use a third party tool for checking and reporting the integrity of your databases. Make sure that you are taking full advantage of all of the mechanisms for identifying corruption. Create alerts that notify you when an 823 (possible corruption detected at the OS level), 824 (possible corruption detected at the SQL Server level), or 825 (823 or 824 error occurred but was successful on a retry) error occurs.

And of course, make sure your databases are all using the CHECKSUM page verification option. Go check all of your databases right now and change them to CHECKSUM if they are using torn page detection or no page verification. You need to understand that changing to CHECKSUM page verification does not mean you are instantly protected. The CHECKSUM values are written to the header when the page is next written to disk. I generally recommend the next time you perform index maintenance in the database; you do a rebuild of every index and every table rather than selectively rebuilding or reorganizing indexes. You can spread this over several nights if you need to. This will cause every page to be rewritten to disk and ensure all data is protected.

Next thing to do is check to see who has elevated permissions in your production databases. Who is in a server role that doesn't need to be? Who has permissions to write to or execute procedures in your databases? Start with the highest levels of permissions and work your way down. Prune the permissions where you can. Work with your engineering teams to find out what permissions are actually required and get down to just those permissions. Start requiring any requests for access to provide a valid justification for why the access is required.

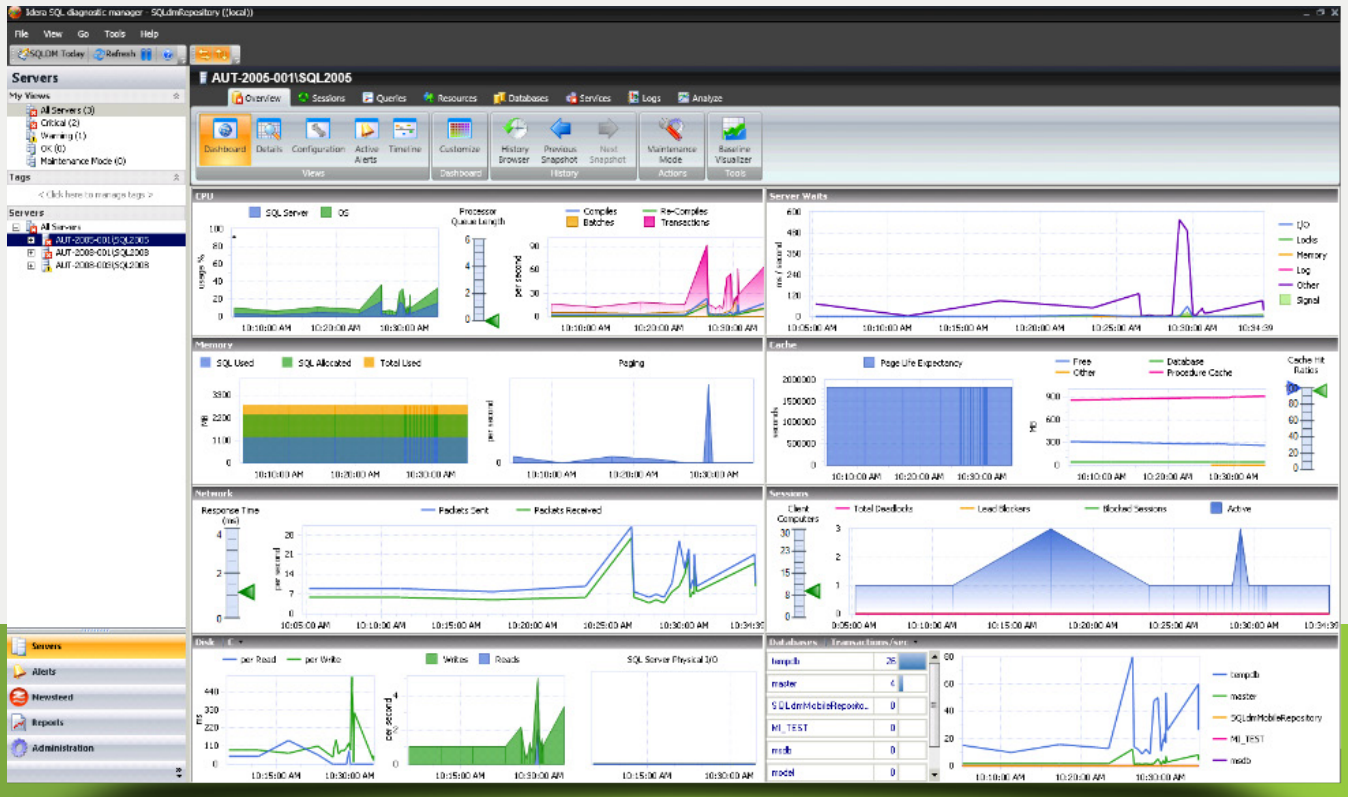
If you know of any shared passwords, get those accounts out of your SQL Server. If you have shared passwords to any accounts, change the password to that account and do not share it again. Require that everyone who accesses the database use credentials that are unique to them.

ACHIEVE 24/7 SQL MONITORING

WITH SQL DIAGNOSTIC MANAGER

- Monitor performance for physical, virtual, and cloud environments.
- Monitor queries and query plans to see the causes of blocks and deadlocks.
- Monitor application transactions with SQL Workload Analysis add-on.
- Optimize SQL queries to maximize application performance with SQL Query Tuner add-on.
- View expert recommendations from SQL Doctor to optimize performance.
- Alert predictively with settings to avoid false alerts.
- View summary of top issues and alerts with the web console add-on.

Start for FREE



IDERA

IDERA.com