SQL Server DBA Training Plan

4th Edition, Winter 2016 - BrentOzar.com/needs

Welcome to Your Training Plan

You're a developer or a sysadmin who wants to become a DBA.
We know. We've been there too, and we're here to share our lessons learned so that your journey is easier than ours. We're Brent Ozar Unlimited®, a consulting company that makes SQL Server faster and more reliable.

I learned to be a DBA the hard way. The hard, crappy way. Our SQL Server was in trouble, and I was the kind of person who would roll up my sleeves and figure out whatever was broken. Next thing you know, I was the one responsible for it.

And boy, did that suck.

I didn't know about any free videos or blogs or e-books. I didn't have the budget to go to a class, and even if I did, I didn't know where to find a good one. DBA wasn't in my job title, and it wouldn't be for years.
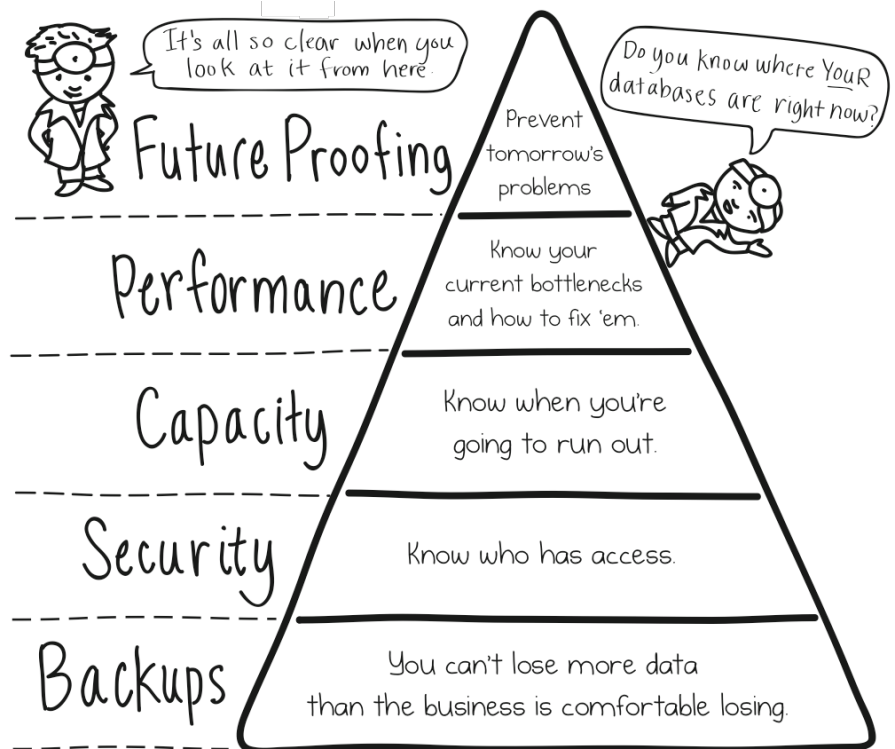
We want to make your learning experience much better than mine was.

This book - part of our free 6-Month DBA Training Plan - covers the basics of the topic, plus link to our favorite free training material on that topic.

Your journey will start with Ozar's Hierarchy of Database Needs, the pyramid shown at right. In the next six months, we'll take you from the bottom of the pyramid up to the top.

You may not be able to fix everything in your environment during those six months, but at least you'll understand the work involved and how to confidently get started.

We've collected thousands of resources over the years - training videos, scripts, blog posts about everything from indexing to careers - and we're excited to share them with you. It's all about making your journey to Professional Database Administrator easier than ours was.

It's all so clear when you look at it from here.

Do you know where YOUR databases are right now?

Future Proofing — Prevent tomorrow's problems

Performance — Know your current bottlenecks and how to fix 'em.

Capacity — Know when you're going to run out.

Security — Know who has access.

Backups — You can't lose more data than the business is comfortable losing.

If You Have Questions

If you have questions about what you're reading, start by Googling your questions. It sounds obvious, but you'd be amazed at how much good stuff there is out there to help. (I'm not being sarcastic. This is exactly how we get started whenever we have our own questions.)

If you have a really short question and you expect a really short answer, and you're on Twitter, tweet the question with #SQLhelp in your tweet. Lots of community members monitor this hash tag, so even if they don't follow you personally, they'll still see the tweet and respond. You can read more about #SQLhelp too.

If you'd like to post a question, try DBA.StackExchange.com or SQLServerCentral's forums. Yes, both of these require registration, but they're totally worth it. On both of these sites, there's hundreds - sometimes thousands - of people who are itching to help answer your questions. They react fast, too - make sure to go back and revisit your question every 10-15 minutes for the first few hours to see what's happening. Answer their clarification questions, and include as much detail as you can. For more instructions, read Getting Help with a Slow Query.

If you still can't get the answers you need, email us at Help@BrentOzar.com. This is a real email address manned by the real people at Brent Ozar Unlimited. This isn't one of those emails where it says, "Don't hit respond because nobody cares." Seriously, we care, and that's why we put these emails together. Just please don't use that as your FIRST resort - we're real people with real jobs and real families, and there's only so many hours per week that we can spend answering questions. By using the above methods first, you'll be able to leverage the whole community's expertise instead of just a few of us.

Build a Server Inventory

Let's start by making an Excel spreadsheet of the servers we've got, the number of databases on each one, and over the coming weeks we'll fix them.

by **Brent Ozar**

At your company, walk into the VP of Sales's office and ask them how many salespeople they have.

NO, I mean, don't actually DO that, because he's going to ask you why the sales app is so slow. But I mean, imagine if you COULD walk into his office and ask him that. I bet he would have an instant answer. He wouldn't wait for a single moment. Or walk into the CEO's office and ask how many employees he has. Or ask the CFO how much the annual budget is.

My point is that when you're in charge, you need to know exactly what you're in charge of.

Make a Spreadsheet Inventory

Let's start by making a spreadsheet. Across the top, make columns for:

- SQL Server Version (2012, 2008, 2005)
- Edition (Standard, Enterprise, Developer)
- Environment (Production, QA, development, disaster recovery)
- Department (sales, HR, accounting, IT, mixed use)
- Business Users Affected (list of people to email when the server dies)
- Application Names (internal or external product names)
- Plan B

That last column gets a little tricky - it means, if this server dies in a fire, what's our Plan B? Are we going to restore the databases from another server? Will we fail over to a log shipped copy? Or will we update our resume and head out for an early lunch? As we go farther into the training, we're going to get much more specific about Plan B.

There's no wrong answers here - week 1 is about understanding where we're at today, not where we'd like to be. We're never where we'd like to be. (Me personally, I'd like to be at a poolside bar right now, but noooo, I'm in a hotel room waiting for my girlfriend to finish blow drying her hair. If you've ever wondered why I write so much, you can thank her full head of hair.)

If you'd like to get ambitious, add additional columns for Core Count, CPU Count, and Memory. The core and CPU counts will get you a head start on licensing, although I have to confess that we're not going to cover licensing as part of our training plan.

What We'll Do With This Spreadsheet

Right now, you probably sleep well at night thinking you know everything that's happening in these servers. Hoooweee, have I got bad news for you. Over the next six months, we're going to progressively add more and more columns to this spreadsheet as we learn more about our environment, uncover problems, and learn how to solve them.

For bonus points, add a column for What Scares Me. Write a quick note about the one

thing that scares you most about this server. Maybe it's blocking problems, maybe it's the failing jobs, maybe it's code you don't understand. Six months from now, I bet you'll be proud of how this column has changed.

How to Survey Your Network for Servers

Put a row in the spreadsheet for every server you have - whether you're in charge of it or not. We want to start with a good inventory of what we have, and there's two good free tools to do it.

Microsoft Assessment and Planning Toolkit - it's actually designed for licensing compliance, but it works great for building server inventories. It scans your network looking for whatever programs you pick, but just confine it to SQL Servers only.

Quest Discovery Wizard for SQL Server - it's a GUI tool that pings all the servers in your network and tries to figure out if they've got SQL Server installed. If you're in a small shop where your account has admin privileges in the domain, you might find a lot more servers than you expected.

Most DBAs don't actually have a list of their servers.

We don't get paid for plugging these products, and we're always on the lookout for similar inventory-building tools, so if you know of a better one, email it to us at Help@BrentOzar.com.

PSST: Ask About This Before Getting Hired

When you take a new job as a DBA, the very first question you should ask the company is, "Do you have a list handy of all the SQL Servers I'll be managing? I don't have to see the list - I understand if you have security concerns - but I just want to know if that list exists."

Most of the time...it won't.

This question serves two purposes: it tells YOU if the company has their act together when it comes to documentation, and it tells THEM that you're the right person to manage their database servers. If they don't have the list, they're going to want that list right away. Now's your chance to explain how you would go about gathering that information (armed with the info in this email.)

Bonus points: create a SQL Server Support Matrix, a document that explains for your developers and end users what's allowed in production, DR, QA, and development. This helps set expectations going forward - if a server's going to be production, then it has to be stable, and that means making sure changes don't happen accidentally.

I created that sample support matrix when I worked as a DBA, and I've shared it so you can do a save-as and get a fast start on your own. Hope that helps!

The Reason We Back Up
All Those Databases

(And yes, this means we're assuming you're backing them up.)

by **Brent Ozar**

The only reason we do backups is so we can do restores.

When I first started out as a SQL Server DBA, I thought things were going well as long as the backup jobs ran successfully. I'd go into SQL Server Agent every now and then, make sure they were still running, and ... that was the end of it. I figured if disaster ever struck, I'd just do a restore. How hard could it be?

In theory, you test our backup strategy ahead of time with Kendra's 5 Simple Questions About Your Backups, and you've memorized the 9 Letters that Get DBAs Fired, along with your company's answers.

In practice, small disasters strike all the time. The most common reasons to do restores aren't to revive an entire server - it's just to get back a few small tables or an individual database. Somebody ran the wrong DELETE statement or dropped a database in production instead of development, and next thing you know, we're all scrambling. Let's think through a few things ahead of time to make the crisis easier.

Where to Do Restores

When you're restoring code (stored procedures, views, triggers, etc) or individual tables, don't restore onto the production server. I don't like touching production servers more than I have to, and let's face it - you're already having a bad enough day as it is. That's why you're doing a restore, remember? So let's do our work on a different server (like dev or QA) and leave production as it is. I've also written about restores in my ideal dev, test, and production environments.

After we've safely restored the right data onto another server, it's easy to copy that data across to other servers. For simplicity and security, you can set up a linked server on the production box with read-only access over to the restore server. Then, from production, you can run INSERT statements using a SELECT sourced from the linked server tables.

However, if you're restoring tables over 10GB, you'll probably want to do the restores directly on the production server to make the data copies faster. Just make sure you're extremely careful with the scripting and the database names - we don't want to restore over the top of your working production database.

This may require adding extra space to the production server. In one emergency, I freed up the necessary space by shrinking all of TempDB's data and log files down to just 1MB. TempDB was on fast drives, perfect for a one-time emergency restore, and that

particular server didn't have any other activity happening due to the outage. We're not always so lucky, but it helps to think out of the box like that.

A word of warning: if referential integrity is involved, like if you're trying to restore tables that have relationships to other tables that you're NOT restoring, then you can be in for a world of hurt here. We're not going to cover that scenario - it really is different in each case.

Doing the Restore

Restore the most recent full backup using the WITH NORECOVERY option - this is really important. This leaves the database in a restoring state so that you can continue to apply additional backups to it. If you forget those two key words, your restore has to start over again from scratch, so please, for the love of all that's holy, double-check that option before you start the restore.

When I'm restoring code or config tables that haven't changed since the last full backups, I don't bother restoring any subsequent differential backups or transaction log backups. The goal is to finish the restore quickly.

Either the backups are tested, or you're about to be tested.

Next, if differential backups are involved, restore the most recent differential WITH NORECOVERY. Differential backups are cumulative - you only have to restore the most recent one.

Next, restore all of the transaction log backups after the differential (or, if you don't have diffs, all of them after the full backup) - again, using WITH NORECOVERY.

Doing all of this with the GUI sucks. The more backups you have, the longer this takes, and the more likely you are to run into errors. Instead, what you need is a script that looks at all of the backups in a folder, plucks out the most recent relevant files,

and restores them for you automatically, in order. We'll talk about automating restores in the next training module.

To learn more about backups and restores, our favorite getting-started articles are:

- Grant Fritchey's SQL Server Backup and Restore for the Accidental DBA
- Brent's DBA Nightmare: SQL Down, No Plans
- Jes's 3 Things You Need to Start Doing to Your Database Server

When I Did My First Restore

I did my first emergency restore when I was working for a photo studio. I'd dropped out of college, and I took a job running their databases. Every morning, I got in bright and early to print out the list of labels for the high school graduates, weddings, babies, and so on that were going to be photographed that day. The photographers would pick up their stacks of labels for their film (FILM! remember that?) and head out into the field.

One morning, as part of my data cleanup process, I ran the DELETE statement. One minor problem - I'd forgotten to put in the WHERE clause, so I deleted all of the photo shoots, ever.

Thankfully, I was also the guy in charge of backups, so after I freaked out, I put last night's tape backups in and started the restores. When the boss came in to pick up his stack of labels, I was able to calmly explain what had happened and what I was doing to fix it.

Orville's temper was legendary, but I dodged a bullet and kept my job. Since then, whenever I handle a database, the very first question I ask is, "Where are the backups, and when was the last time they were tested?" Either the backups are tested, or you're about to be tested.

Doing Faster Database Restores

The GUI isn't going to cut it. We're going to have to roll up our sleeves. As we script out a restore, we'll learn how we need to design backups.

by Brent Ozar

If you're doing transaction log backups, forget using the GUI. Even if you only have one transaction log backup per hour, it'll take you way too long to click through all the files.

Think about what your backup folder might look like if we named our backup files by database, date, time, and a different extension per type of backup (BAK for fulls, DIF for differentials, and TRN for transaction logs):

- MyDatabase_20130718_0000.**bak**

- MyDatabase_20130718_0100.trn

- MyDatabase_20130718_0200.trn

- MyDatabase_20130718_0300.trn

- MyDatabase_20130718_0400.trn

- MyDatabase_20130718_0500.trn

- MyDatabase_20130718_0600.**dif**

- MyDatabase_20130718_0700.trn

- MyDatabase_20130718_0800.trn

In that scenario, I took my full backup at midnight, then hourly transaction logs, with differential backups every 6 hours. (This is never a scenario I'd use in the real world, but

Knowing how you'll restore affects how you back up.

it's the simplest way to get the point across in a tiny email. Hey, you try teaching tough concepts in a page or two, buddy.)

If disaster strikes at 8:15AM and I lose the server, I need to restore the full, the most recent differential, and the transaction logs that follow the differential, like this:

- MyDatabase_20130718_0000.**bak**

- MyDatabase_20130718_0600.**dif**

- MyDatabase_20130718_0700.trn

- MyDatabase_20130718_0800.trn

That's a really easy script to write - and thankfully, MSSQLTips has already done it. Just change the variables for your database name and the path where your restore files live, and presto, the script will restore all of the files for you automatically.

This Script Has Implications For You

You need to back up your databases intelligently. You want to put each database in its own folder, and you want the file names to have the database name and the time in them. I'm a big fan of that anyway - it makes life easier when I want to quickly scan and see what backups have been done.

It assumes the same database path when you restore. If you're the kind of DBA who likes to change drive letters and folders all the time, or you've got files scattered all over the place, and your production and

development database servers have different drive letters, then life is going to be harder for you. There are other restore scripts that can adjust data and log file names at restore time, but they're not quite as elegant when it comes to restoring fulls, diffs, and t-logs.

You can do transaction log backups as often as you want. If the restore process is fully automated, why not do transaction log backups every minute? It doesn't cost you any extra. Rather than incurring extra overhead, it can actually incur LESS overhead. If you're only backing up logs once an hour, your users probably feel that hourly load big time when it kicks off. If you do smaller backups more frequently, they'll be less likely to notice the impact.

You can automate your own fire drill testing. Since this is done with plain old T-SQL, you can set up a SQL Agent job to restore last night's backups onto your staging server. I like setting up a DBA utility server chock full of really cheap (but really large) SATA drives, like 2-4TB drives in a RAID 5. I have a series of Agent jobs to restore a different server's backups every day, and then run DBCC jobs on them. This way I know I'm getting really good backups.

Your Homework for This Chapter

You don't have to go build this whole infrastructure out - but start laying the groundwork by making sure your backup files are named in a restore-friendly way.

Then, try out the MSSQLTips restore script to make sure you can quickly restore a database from scratch into a development or staging environment. If you get errors, leave comments on that blog post describing the error, and ideally, contribute your script improvements back in. Now, when disaster strikes, you won't be clicking around blindly in a GUI - you'll just open a proven script and hit Execute. Bam!

Getting Even Fancier

You can use this technique to build smaller copies of your production databases. For example, at StackOverflow, I built an ultra-compressed backup with even more tricks. On my nightly restore server, after restoring the production backups, I dropped the non-clustered indexes, rebuilt all objects with 100% fill factor, and shrank the data and log files down.

The end result was a database that was over 50% smaller!

I then backed it up with compression, and left those backups in a folder that the developers could access. That made it easier for developers to quickly grab a copy of production data as of last night, copy it to their workstation as quickly as possible, and restore it faster with less drive space required.

Free 30-Minute Videos to Learn More

Log Shipping Part 1: Jes explains how log shipping is really just doing your restores ahead of time.

Log Shipping Part 2: when disaster strikes, what you'll need to do to fail over to the secondary server and manage your jobs.

Backup Planning Video: Jeremiah explains 3 company scenarios and asks you to pick the right backup strategy for each.

Our Backup Video Archive: no, it's not a backup of our videos - it's our videos about backups. We talk database mirroring, clustering, AlwaysOn Availability Groups, and more.

Security: Knowing Who Has Access to the Servers & Data

Before we try to lock things down, find out who has copies of keys.

by **Brent Ozar**

Run our free sp_Blitz™ stored procedure on one of your production servers and pay particular attention to the Security section of the results. It lists the logins who have been granted the sysadmin or security admin roles.

Don't think of them as logins.

Think of them as people who can get you fired.

These people can drop databases, drop tables, change stored procedures, edit data, or even change SQL Server configuration settings like max server memory or maxdop. You're probably not getting alerted when any of these things change - we just can't afford to monitor every single thing in SQL Server and send alerts on it. At some point, we have to be able to trust certain logins, and that's where the sysadmin and security admin roles come in.

Except when we first get started learning database administration, it's usually because we're the only DBA in the shop, and the server is a mess. The front door is unlocked, the key is under the floor mat, and everybody knows we've got a big screen TV in the living room.

How to Get Started Locking Things Down

Before you start removing people's SA rights, be aware that there can be political backlash. In one shop, the head developer's SA rights were removed, and he stormed into the DBA's office the next morning screaming. Turns out one of his apps automatically created a processing database every night, did a bunch of work in it, and then dropped the database. Nobody

Think of sysadmin logins as people who can get you fired.

knew because it was only around for 30-45 minutes. The problem could have been avoided by communicating the security changes ahead of time, and that's where we need to start.

Take the sp_Blitz™ output to your manager - just the security portions - and say something like this:

"Here's the list of people who can do anything on the production server - delete data, drop databases, or change performance settings. If they do, here's the list of applications that will be affected, including accounting and payroll. I don't want to take away all of their permissions - I just want to start by giving them full permissions over their database, but not in any other databases, and not at the server level. Can I talk to them about doing that?"

Note that we're only going to TALK to them, not actually do it, because we need to communicate with them first. Then, go to the end users or developers involved and say:

"We're starting to lock down the production server, but I want to make sure you have all the permissions you need. I'm going to make you a complete database owner inside your database so you can do anything you want in there, but I'm going to take away your rights to the other databases (like accounting and payroll), and I'm going to remove your permissions to change server-level settings like how much memory the server can use. I'm planning on doing it next weekend, and I want you to have my email and phone number so that if anything breaks on that date, you can call me and I can audit what's happening on the server to see if it's related to the permissions change."

When You Get Resistance

When - not if - you get pushback from developers or users, go to the project managers or business people who have a large stake in the database. For example, if the accounting database is on the server, go to the CFO and say:

"Here's the list of people who can take down the accounting system. They have the permissions to drop the database at any given time, and there's nothing I can do to stop it. I'd like to get that changed - can I schedule a short meeting with you and the development manager to get everyone on the same page?"

You want to turn it into a business problem, not a technology problem, and the CFO will very much be on your side. She can't afford to have her entire department go down just because some developer didn't put a WHERE clause on a T-SQL statement.

I Know, This Chapter Isn't Fun

Database administration isn't all bacon and roses. Sometimes it's boring politics and paperwork, and this is one of those weeks.

In the very first chapter, we built a spreadsheet inventory of our servers, and now it's time to fill in a little more details. Since we're analyzing security, we need to know which applications live on each server, and who's in charge of each of those applications. You don't have to fill in the specifics of who has read or write permissions in each database, but we want to at least know the business purpose and

the business co[...]

The business co[...] one who really helps us get the [...] ocked down because their jo[...] line if this data is lost or unavailab[...] e to think of it, you're someone w[...] get THEM fired!) In the coming week[...] w[...]rking more with them [...] rmance, too, so [...] shing out tha[...]

Datab[...]

Sure, [...] here to pro[...] at we

do involves sitting in meetings, convincing people to do what we want, how we want.

It's made more challenging because we often don't have any real authority. Sometimes the DBAs report to completely different managers than the developers - and sometimes, it's even different companies! We might be administering a database server that houses a third-party application, and the vendor's support team demands to have SA access.

Consulting Lines:

After consulting for a while, I wrote a series of posts with lines you can use as a DBA:

"Sounds like you've got it all under control."

"What happens if that doesn't work?"

"Would you mind driving?"

"SQL Server needs a dog."

Risk isn't just a board game.

Top tip: when you play Risk, go for South America. It's only 4 countries with 2 access points, so it's easy to defend. See, security matters.

by **Brent Ozar**

Download the SQL Server Compliance Guide. It's an old 92-page whitepaper circa 2008, but it might just be the best (and most timeless) technical document that Microsoft has ever produced. **JC Cannon** and **Denny Lee** deserve a big thank-you for a great job.

I only need you to read pages 7-13. In those six pages, you'll understand the differences between risk management, governance, and compliance. Risk management means knowing what risks the company is taking with the data, governance means the actions taken to address the risks, and compliance means someone is double-checking that we're actually doing the governance stuff we say we're doing.

Your job as a DBA involves all three, but when you're just getting started with compliance, focus on risk management. We need to get a quick idea of the different ways we could lose data, or that supposedly secure data could get into the wrong hands. (Yes, your developers are probably the wrong hands, but that's a different story.)

Homework: Look for Risky Business

Odds are, nobody in the company has an inventory of the data we're storing in ridiculously unsafe ways. Here's a quick way to check your own database looking for dangerous fields:

```
SELECT * FROM
INFORMATION_SCHEMA.COLUMNS
WHERE COLUMN_NAME LIKE '%password%'
OR COLUMN_NAME LIKE '%social%'
OR COLUMN_NAME LIKE '%credit%'
```

Feel free to change out those keywords for other terms that are relevant to your business - fields that are sensitive, and that would hurt if they got out into the wild. Then look at the contents of those tables - is the data being stored unencrypted? Who has access to it?

If we're storing unencrypted passwords in the database, for example, then every database backup we've ever done is dangerous. If anybody gets access to any backup file, like our offsite tape backups, then we could be on the front page of the news tomorrow.

When you find it, email the developers and the business contact for that database. Explain that the data is not encrypted, and use examples from the SQL Server Compliance Guide to show how this is a risk for the business.

Compliance: Knowing Who Accesses What

It's unusual for me to assign homework first in the email, and then go on to talk about other things, but I want you to be aware of other things that companies usually do around their secure data.

Often management will say, "We need to audit everyone who changes or accesses secure data." Technically, SQL Server has features that can accomplish this goal - things like SQL Server Auditing. In practice, though, this is a problem for hard-core security teams because the very same DBA who manages the servers also has permissions to change the SQL Server audits. A greedy DBA could easily disable auditing, get the necessary data, and then enable it again.

Know whether your company just wants to check a box on an auditor's form, or really be secure.

With seriously ambitious work, you can lock auditing down so that can't happen, but ... it's really, really hard.

As a result, most seriously secure companies end up with a completely different auditing solution that lives outside of the DBA's realm. The security team buys third party hardware appliances like IBM Guardium or Imperva that act like a network firewall between everyone and the SQL Server. The appliance logs everything that happens in the SQL Server, and then the security team can review those logs without you (the DBA) knowing about it.

These solutions are typically six figures and up. They're expensive because they have to be absolutely bulletproof - if they fail, you can't have the access to the database suddenly stop. Plus, you'll need them in place for not just your production environment, but also your disaster recovery environment.

If you just want to check a box and make the auditors think you're secure, that's easy and cheap, but seriously good security is seriously expensive.

Big Companies Love Security

The bigger the company, the more interested they get in security.

In small shops without a dedicated security department, everybody is often a domain administrator, and everybody knows where all the passwords are stored.

In big shops with a dedicated security team, the security team usually makes requests to disable the DBA's sysadmin access. They want the DBA to only have enough permissions to do their job, but not enough permissions to see the database contents. (After all, the DBA shouldn't see credit card data, health data, etc.)

This isn't a realistic request in SQL Server 2012 and prior, but SQL Server 2014 brings new server roles that will make this request easier. For more info, check out the security enhancements section of What's New in 2014.

Learning More About Security

Check out Denny Cherry's book Securing SQL Server (paperback - Kindle). Denny manages to bring a very dry topic to life.

Every shop with PCI, HIPAA, SOX, or other compliance needs should definitely own a copy of this book. It's a tougher sell for small shops with only one full time DBA, though, because a lot of the security features discussed take time to digest and implement.

It's Time to Level Up.

In Ozar's Hierarchy of Database Needs, we've touched on the bottom two layers, backups and security. Before we move up a level, let's pause.

by **Brent Ozar**

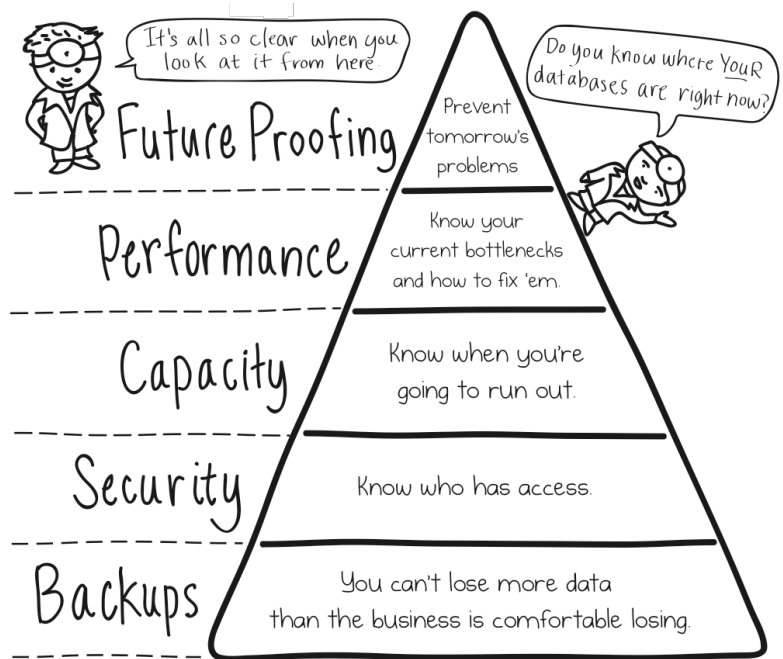We've covered backups and security, except...

Well, we haven't.

We could go on for hundreds of pages about those two levels alone. We could fill entire ebooks, perhaps even libraries, with cool concepts like how the differential bitmap works, how to secure stored procedures with certificates, how to configure SQL Server when using VMware-based backups for storage replication, how to recover from a corrupted nonclustered index, yadda yadda yadda.

Our point of this ebook is a fast start.

Your journey in SQL Server database administration is never really done. You're going to be learning for the rest of your life, and even when you think you're done, you're only mistaken. Heck, I'm a Microsoft Certified Master, and I still learn things in almost every presentation I attend. (Except Jeremiah's, because I'm not smart enough to digest that stuff.)

So we're going to move up a level to capacity, but please don't email us with cries of complaints about how you wish we would have gone deeper into clustering setups, replication internals, or whatever other

It's all so clear when you look at it from here.

Do you know where YouR databases are right now?

Future Proofing — Prevent tomorrow's problems

Performance — Know your current bottlenecks and how to fix 'em.

Capacity — Know when you're going to run out.

Security — Know who has access.

Backups — You can't lose more data than the business is comfortable losing.

Trivial Pursuit category you're hot on this week. We just can't cover it all in one ebook.

When you find an area that you want to dig deeper into, check out:

Our video training courses - from our $29 90-minute courses all the way up to our 5-6 hour monsters at $299, we can go much deeper. Some even include quizzes!

Our in-person classes - we have 2-3 day courses for developers and DBAs who need to make SQL Server faster and more reliable.

Our SQL Critical Care™ - we work together with you over WebEx or in person to diagnose your SQL Server's pain points and train you on the fixes.

How to Monitor Drive Space

Just when you thought it was safe to let your guard down, we throw a trick chapter at you.

by Brent Ozar

Don't.

It's not your job.

No, seriously, hang on a second. I'm not saying that you should ever say things like "It's not my job," but I do want you to understand when you should avoid reinventing the wheel. Your Windows administrators should be monitoring every Windows server you have, making sure it doesn't run out of drive space. This isn't some wild, unique need that only SQL Server has - it's everywhere in your data center.

(You might actually even BE a Windows admin, just here because you need to learn about working with SQL Server. It might actually BE your job to monitor this stuff. That's cool - that's how I got started too.)

In our journey from the base of our Hierarchy of Database Needs to the top, we do indeed need to talk about capacity - but I don't want to monitor drive space from inside SQL Server, and I don't want to buy SQL-specific tools in order to pull it off. Yes, you can technically use commands like xp_fixeddrives to list all of the SQL Server's drive letters and how much free space they have, but that doesn't work everywhere. For example, you might have mount points or databases on UNC paths, neither of which will show up in xp_fixeddrives. So leave the drive space monitoring to the pros.

Why You Shouldn't Build a Monitoring Tool

If you want to be a professional developer, you should build things. You should learn what exceptions could happen, how to trap them, and how to fail elegantly. It's hard work, but if you get good at it - really good - you can build amazing things and make a killer living for yourself.

But if you want to be a professional DBA, you should leave the developing to the developers.

I'm not saying you shouldn't learn the dynamic management views (DMVs), how to dig through them to find the evidence you need, and how to turn that data into actions. Hopefully, I'm going to teach you how to do a lot of that over the course of the next six months. Take another look at the Hierarchy of Database Needs again, and think for a second about all the things we're going to be learning over the next six months. Just in the last five weeks, I've had you build an inventory of your servers, automate your database restores, start locking down security, and enumerate your database risks. The next few months are chock full of things that will help you become a hero in the eyes of your users and your developers.

Building a crappy monitoring tool in your spare time will not give you that same level of respect. (And yes, if you've only got your spare time at work, your monitoring tool is going to be crappy. If you've got so much time that you can build a great monitoring

tool, you need to focus on providing results to the business fast, because you're about to be laid off as pure overhead.)

How to Pick a Monitoring Tool

There's basically three kinds of monitoring tools out there for SQL Server.

Hardware Alerting - when you buy a server from big companies like Dell, HP, or IBM, they come with built-in management tools. These tools are surprisingly powerful - for example, check out our post on How to Use HP System Management Homepage.

Up/down Alerting - these tools make sure the SQL Server service is running, and that it has all the basic needs taken care of. If the server runs out of drive space, if the service stops, if the event log starts filling up with errors, or zombies attack, these tools will let you know. The most popular software in this category is Quest Spotlight, Idera SQL Diagnostic Manager, and Red Gate SQL Monitor.

Performance Diagnostics - these tools try to derive insight from SQL Server's dynamic management views (DMVs) to help performance tuners wring the most throughput out of their hardware. The most popular software here is SQL Sentry Performance Advisor, Quest Foglight Performance Analysis, and to some extent, the tools in the above category too.

If I was you, I'd start by asking the Windows team if they've got any software that handles the up/down alerting for services, drive capacity monitoring, etc. If so, get them to start monitoring your servers. I'm being selfish here - my real reason is that I want to save my monitoring budget for tools in the Performance Diagnostics category. Surprisingly, all of these tools are around the same price - around $1,000-$2,000 USD per monitored instance.

Next, it's time to evaluate them. Don't just install a bunch of random tools in production and see what they find. Instead, no matter which category of tool you're buying, make a list of the last 4-5 things that have caused your phone to ring after hours. Here's some of my personal favorites:

- Deadlocks
- Out-of-control query running for hours
- Long-running job, like slow backup
- Queries that desperately need an index
- SQL Server not accepting connections

Figure out how to recreate those same symptoms in your development environment, and then get a free trial of a couple of the tools I mentioned. (All of them provide free 10-14 day trials.) Reproduce the problem, and watch how the monitoring software reacts. Does it lead you to the root cause quickly, or does it just have a bunch of flashing red lights on the console? The best ones will save you time by getting you right to the solution.

After you've done your evaluation and picked a favorite, get a quote from them - and get a quote from the other vendors as well. Share the quotes with the competing salespeople.

If you're building a monitoring tool in your spare time at work, it's either going to be crappy, or you're about to be laid off because you have too much spare time.

You'd be surprised how willing they are to negotiate, and you can use the cheapest quote to get the tool you really want.

Your Homework

I don't expect you to magically get budget approval for a tool this week, but I want you to start making a list of day to day problems as they strike. If you waste four hours troubleshooting a deadlock problem, make a note of the date, the time required, and a few words about the root cause. In a month or two, you'll be able to present this list to your boss as proof of why the right monitoring tool could save you money. Down the road, I'll also link you to a video showing you how to get the budget approval for that tool.

I'm not selling you vendor tools, by the way. I often link to vendor tools, but I don't receive any compensation. I'm just a huge believer in using the right tool to look like a hero fast - instead of building your own hammer every time you change jobs.

How This Philosophy Cost Me a Job

I actually failed a DBA job interview once for giving the same answer in my email here. The lead DBA asked me, "If you were going to monitor your servers, how would you do it?" I said I'd buy a tool off the shelf, and he was furious. He wanted me to use an open source monitoring framework.

Today, several years down the road, now that I'm a Microsoft Certified Master, my answer is still the same. I don't reinvent the wheel, and neither should you. Focus on things that provide the business tremendous value, and your career will take care of itself.

If your business would find tremendous value in a killer monitoring system, then become a developer and give it to them, or consider contributing to the huge number of open source monitoring products. Don't build a new one from scratch, and definitely know which metrics to monitor.

Bobcats per 100 Orders and Other Spurious Metrics

by Jeremiah Peschka

Did you know that you can ship a bobcat 1/30th of the time and still maintain 97% positive feedback on ebay?

What other statistical lies are lurking out there for you to find in Perfmon? Cache Hit Ratio, Disk Queue Length, Page Life Expectancy, Page Splits, and User Connections can be bogus. Learn why.

Monitoring SSD Performance

by Jeremiah Peschka

What if we could watch SSD wear in real time? Many vendors offer SMART status codes to return detailed information about the status of the drive. Rotational drives can tell you how hot the drive is, provide bad sector counts, and a host of other information about drive health. Read more in this post.

Before you build or buy, check out the best free SQL Server downloads:

BrentOzar.com/go/freebies

Manage Space Inside Databases

While I don't want you reinventing the "Is the C Drive Full?" wheel,
I very much want you to know what's going on inside each database.

by **Brent Ozar**

Inside each of your database data files (MDFs), SQL Server stores your data in 8KB pages. That's kilobytes - not megabytes, not gigabytes, but just mere kilobytes.

Say that we create a table:

```
CREATE TABLE dbo.Employees
(EmployeeID INT IDENTITY(1,1),
 EmployeeName VARCHAR(200))
```

First off, yes, I understand, I shouldn't do EmployeeName in one field - I should have FirstName, LastName, MiddleName, Suffix, Prefix, yadda yadda yadda, but I'm trying to keep this email short. Now see what you did? It's long again. Doggone it, it's hard teaching this stuff in an email.

Anyhoo, in this table, each record takes up just a little bit of space. EmployeeID is an INTeger, which takes 4 bytes. It's the same 4 bytes whether that number is 1 or 1,000,000. EmployeeName is a VARCHAR(200), which means we can store up to 200 characters in here, and each character takes a byte. If we insert 'BRENT OZAR', that's 10 characters (and boy, am I a character), so we need 10 bytes to store it.

If all of our employees average about 10 characters in their name, that means we could fit about 500-600 records per 8KB database page. (In reality, there's some overhead because SQL Server also needs to use some parts of the page to store metadata, and we'll talk about that later in the training.)

Brent Ozar Unlimited is a small company, so we can keep all of our employees on a single 8KB page. As we insert, update, and delete employees, SQL Server fetches that 8KB page off disk, brings it into memory, makes the necessary changes, and then writes that data page back to disk. The 8KB page itself is the smallest unit that SQL Server will cache - it doesn't cache individual rows/records - and each page belongs exclusively to just one object.

A Word About Objects

You'll notice that I avoid using the word "table". Tables are cool, but as we start to dig into what SQL Server's doing under the hood, we want to start thinking about these three object types:

Heap - a table with no clustered index. In my dbo.Employees table, I didn't specify in what order SQL Server should store my data, so it's just going to slap the data down on my 8KB page in any old order.

Clustered Index - what we normally think of as a table. If I'd have created my table like this:

```
CREATE TABLE dbo.Employees
(EmployeeID INT IDENTITY(1,1) PRIMARY
KEY CLUSTERED,
 EmployeeName VARCHAR(200))
```

Then SQL Server would store my data in order of EmployeeID. That way, when I search for EmployeeID #42, it can go directly to that number without scanning through all of my employees. The clustered index is sorted in the order of the EmployeeID field, but it's actually the full copy of our table, including all of our fields - in this case, just EmployeeName.

Nonclustered index - If I tell SQL Server to:

```
CREATE NONCLUSTERED INDEX IX_Name
ON dbo.Employees(EmployeeName)
```

Then SQL Server will create a second copy of my table sorted by EmployeeName. This copy of my table will only include the fields specified in my index (EmployeeName), plus whatever fields it needs to get back to the clustered index (in this case, my clustering key, EmployeeID).

Armed with this knowledge of pages, you'll make better data modeling decisions, too. (MAX) ain't free.

All three of these objects - heaps, clustered indexes, and nonclustered indexes - will be stored on separate sets of pages. We won't have the clustered index and nonclustered index for the same table on the same page - they're split. That's why when we're doing space analysis, we have to think in terms of indexes, not tables. To learn more index basics, read Jes Borland's SQL Server Index Terms.

Pages & Rows on Disk

The sys.dm_db_index_physical stats Dynamic Management Function (DMF) returns the number of rows and pages stored in each database object. It takes parameters for database ID and object ID, or you can pass in NULLs to get information across all of your database objects. Scroll down to the examples link in that Books Online page, and you'll get queries to examine a single table - I'd strongly recommend starting with a small table, because some of the parameters for this DMF will cause SQL Server to actually look at every page in the object. That means if all of the pages for that object aren't cached in memory, SQL Server will go pull those pages off disk, and that can slow down your running SQL Server.

This DMF also includes average record size and max record size. This makes for fun spelunking: how big is each record, really? Just because we make everything a VARCHAR(8000) doesn't mean we're actually storing 8,000 characters in each field. Now, don't go changing your database structure just yet - you can easily break applications when datatypes change. Let's leave that for later.

You can get similar metadata much faster by using sp_spaceused, but it doesn't get the cool info about average record size, and I wanted to encourage you to go spelunking here.

Learning More About Pages

In my 90-minute session How to Think Like the Engine, I explain pages, indexes, joins, SARGability, and more. I use real database pages from the StackOverflow.com database for demos, and you actually get PDFs to print out and follow along as we go.

Your Homework

Let's start thinking about what objects are taking up space in our databases. Check out

our free sp_BlitzIndex™ stored procedure that analyzes the indexes in your database from a psychologist's point of view. Is your database a hoarder, clutching on to a bunch of nonclustered indexes that aren't getting used, and are just taking up space? The details columns in the results will show how big each index is, and whether it's getting used or not.

This is important because the more indexes you have:

- The longer your backups take
- The longer index rebuilds take
- The more memory you need to cache everything

And most importantly, the slower your inserts/updates/deletes go, because SQL Server has to maintain more copies of your table.

As we start to move up the Hierarchy of Needs from capacity into performance, you'll start to see how these foundational items are so important.

Want to See What a Page Looks Like?

Check out the DBCC PAGE command. You pass in a database name, file number, and page number, and SQL Server will return the nearly-raw contents of that page along with diagnostic information. It's a fun way to get a peek under the hood.

This topic is a good example of how knowing the basics of database internals can come in handy when you step back and think about database server performance. You don't have to use DBCC PAGE as part of your job as a DBA, but just knowing what an 8KB page is helps you understand the output of various Dynamic Management Views (DMVs) when they report back units in pages.

We've got tons of blog posts and videos about our favorite index tips, tricks, and free tools:

brentozar.com/go/index

What Pages are In Memory?

If everything's stored on an 8KB page, and each page has only one object on it, then we can figure out what objects (tables/indexes) are getting cached.

by **Brent Ozar**

We like to think SQL Server is using all of our memory to cache data, but that's just part of it. SQL Server uses memory for lots of things:

- Caching database objects
- Sorting data for your query results
- Caching execution plans
- Performing system tasks

Often, we're surprised by how little data is being cached for each database.

Last section, we looked at the 8KB pages in our database. Those pages are the same whether they're on disk or in memory - they include the database ID and the object ID, so if we looked at all of the pages in memory, we could figure out which tables are being cached in memory right now. The below query gives us the magic answers, but be aware that the more memory you have, the longer this will take. It won't block other users, but it could take a minute or two if you've got >64GB memory, several minutes if you've got a terabyte or more:

You might be surprised at how little memory is used for caching data.

```
SELECT CAST(COUNT(*) * 8 / 1024.0 AS
NUMERIC(10, 2)) AS CachedDataMB ,

CASE database_id WHEN 32767 THEN
'ResourceDb' ELSE DB_NAME(database_id)
END AS DatabaseName

FROM sys.dm_os_buffer_descriptors

GROUP BY DB_NAME(database_id) ,
database_id

ORDER BY 1 DESC
```

Compare the size of each database versus how much is being cached. Often in the field, I'll see 100GB databases that just have 8-12GB of data cached in memory. That might be completely okay - if you only regularly query just that amount of data - but what if we constantly need all 100GB, and we're constantly pulling it from disk?

This Leads to Cool Questions

This DMV query leads to so many cool performance tuning questions! I get so excited by these concepts.

How fast are the cached pages changing? From the moment we read an 8KB page off disk, how long does it stay in memory before we have to flush it out of the cache to make room for something else we're reading off disk? This concept is Page Life Expectancy, a Perfmon counter that measures in seconds how long things stay in RAM. The longer, the better, as I explain in my Perfmon tutorial.

Do the results change based on time of day? This is a one-time snapshot of what's in memory at the moment, but it can change in

a heartbeat. If you have automated processes that run a bunch of reports in a single database at 2AM, then the memory picture will look completely different then.

Are we caching low-value data? If you mix vendor apps and in-house-written apps on the server, you'll often find that the worst-written application will use the most memory. Thing is, that might not be the most important application. Unfortunately, we don't have a way of capping how much memory gets used by each database. This is why most shops prefer to run vendor applications on separate virtual machines or servers - this way, they don't hog all the memory on a SQL Server that needs to serve other applications.

Do we have enough memory? If you're running SQL Server 2008/R2/12 Standard Edition, you're limited to just 64GB of physical RAM. If you're running SQL Server on bare metal (not a VM) and you've got any less than 64GB, go buy enough to get to 64GB. It's the safest, easiest performance tuning change you can make. If you're in a VM or running Enterprise Edition, the memory question gets a lot tougher. To learn more, read [A Sysadmin's Guide to SQL Server Memory](#).

Are we using memory for anything other than SQL Server? If we've got Integration Services, Analysis Services, Reporting Services, or any other applications installed on our server, these are robbing us of precious memory that we might need to cache data. Don't remote desktop into your SQL Server and run SSMS, either - it's a memory pig. Put your management tools on a virtual machine in the data center, and remote desktop into that instead.

Can we reduce memory needs with indexes? If we've got a really wide table (lots of fields) or a really wide index, and we're not actively querying most of those fields, then we're caching a whole bunch of data we don't need. Remember, SQL Server is caching at the page level, not at the field level. A nonclustered index is a narrower copy of the table with just the fields/columns we want. The less fields, the more data we can pack in per page. The more we can pack in, the more data we're caching, and the less we need to hit disk.

When I tune indexes on a server I've never seen before, sys.dm_os_buffer_descriptors is one of the first places I look. The database with the most stuff cached here is likely to be the one that needs the most index help.

It's Probably Not a SAN Problem.

When I was a junior DBA, I focused a lot on the storage. I kept complaining to my SAN administrators because my storage didn't respond fast enough - my drives were taking 50ms, 100ms, or even 200ms in order to deliver data for my queries.

The SAN admin kept saying, "It's okay. The SAN has a cache." Thing is, the size of the SAN's cache is typically 32GB-128GB - which at first sounds like a lot - but divide it between all of the servers connected to the SAN. Often, we find that an individual SQL Server might get only a couple of gigabytes of SAN cache. That's way less than what the SQL Server has in memory. What are the odds that, when we need data for a query, it's not going to be in SQL Server's 64GB of memory, but it IS going to be in the SAN's miserly 2GB of cache? Not gonna happen.

SAN caching is still great for writes, especially for the transaction log, but don't count on it helping for SELECT speeds.

To learn more, check out [our training course on Storage, Virtualization, and Hardware for DBAs](#).

Indexes: What Goes First?

In which we pretend the phone company, but instead of giving everybody unlimited calling, we just organize the data.

by **Brent Ozar**

For our evil training purposes, let's say we work for the phone company, and we need a database table with phone numbers. We need to track:

- Phone number (required)
- Billing contact last name (required)
- Billing contact first name (required)
- Business name (optional)
- Business category (restaurant, dog groomer, auto dealer, etc)
- Address 1
- Address 2
- City
- State
- Zip
- Service start date

No, Brent's home number is not in this chapter.

(Sometimes a person or a business will have multiple phone numbers, but for the sake of this training, let's keep it a simple flat table.) We will never have two records in here with the same phone number. We have to tell our database about that by making the phone number our primary key.

When we make the phone number the primary key, we're telling SQL Server that there can be no duplicate phone numbers.

That means every time a record is inserted or updated in this table, SQL Server has to check to make sure nobody exists with that same phone number. As of the year 2000, there were about 360,000 people in Miami. Throw in businesses, and let's say our table has 500,000 records in it.

That means by default, every time we insert one eensy little record, SQL Server has to read half a million records just to make sure nobody else has the same phone number! Every 1 write = 500,000 reads. Well, that won't work, will it? So let's organize our table in the order of phone number. That way, when SQL Server inserts or updates records, it can quickly jump to the exact area of that phone number and determine whether or not there's any existing duplicates. This is called setting up a primary CLUSTERED key. It's called clustered because - well, I have no idea why it's called clustered, but the bottom line is that if you could look at the actual hard drive the data was stored on, it would be stored in order of phone number.

Now we have the table organized by phone number, and if we want to find people by phone number, it'll be very fast. While our computer systems will usually need to grab people's data by phone number, our customers and end users often need to get numbers by other ways. That's where non-clustered indexes come in.

The White Pages: A Non-Clustered Index

Our customers constantly need to find people's phone numbers by their name. They don't know the phone number, but they know the last name and first name. We would create an index called the White Pages:

- Billing contact last name
- Billing contact first name
- Phone number

That index would save people a ton of time. Think about how you use the white pages:

1. You scan through pages looking at just the letters at the top until you get close
2. When you get close, you open up the full book and jump to the right letters
3. You can quickly find the right single one record

Now think about how you would do it without the White Pages. Think if you only had a book with 500,000 records in it, organized by phone number. You would have to scan through all 500,000 records and check the last name and first name fields. The database works the same way, except it's even worse! If a developer wrote a SQL query looking for the phone number, it would look like this:

```
SELECT PhoneNumber FROM Directory WHERE
LastName = 'Smith' AND FirstName =
'John'
```

That doesn't say select the top one - it says select ALL of them. When you, as a human being, go through that list of 500,000 phone numbers, you would stop when you thought you found the right John Smith. The database server can't do that - if it finds John Smith at row #15, it doesn't matter, because there might be a few John Smiths.

Whenever you do a table scan and you don't specify how many records you need, it absolutely, positively has to scan all 500,000 records no matter what.

If the database has an index by last name and first name, though, the database server can quickly jump to Smith, John and start reading. The instant it hits Smith, Johnathan, it knows it can stop, because there's no more John Smiths.

Covering Fields: Helping Indexes Help You

But that's not always enough. Sometimes we have more than one John Smith, and the customer needs to know which John Smith to call. After all, if your name was John Smith, and the phone book didn't include your address, you'd get pretty tired of answering the phone and saying, "No, you want the John Smith on Red Road. He's 305-838-3333." So we would add the Address 1 field in there too.

- Billing contact last name
- Billing contact first name
- Address 1
- Phone number

Do we absolutely need the address in our index for every query? No, but we include it for convenience because when we DO need it, we need it bad. And if we DON'T need it, it doesn't really hurt us much.

This is called a covering index because it covers other fields that are useful. Adding the address field to our index does make it larger. A phone book without addresses would be a little thinner, and we could pack more on a page. We probably don't want to include the Address 2 field, because the Address 1 field is enough to get what we need. The database administrator has to make judgement calls as to which fields to use on a covering index, and which ones to

skip. When building covering indexes, the covering fields go at the end of the index.

Obviously, this index would suck:

- Billing contact last name
- Address 1
- Billing contact first name
- Phone number

We don't want all of the Smiths ordered by their address, and then a jumbled mess of first names. That wouldn't be as fast and easy to use. That's why the covering fields go at the end, and the names go first - because we use those.

Selectivity: Why the Last Name Goes First

If you wanted to search for Brent Ozar in the phone book, you look in the O's for Ozar first, and then you'll find Ozar, Brent. This is more efficient than organizing the phone book by first name then last name because there are more unique last names than first names. There are probably more Brents in Miami than Ozars. This is called selectivity.

The last name field is more selective than the first name field because it has more unique values.

For lookup tables - meaning, when users need to look up a specific record - when you've narrowed down the list of fields that you're going to use in an index, generally you put the most selective field first.

Indexes should almost never be set up with a non-selective field first, like Gender. Imagine a phone book organized by Gender, Last Name, First Name: it would only be useful when you wanted a complete list of all women in Miami. Not that that's a bad thing - but no matter how much of a suave guy you think you are, you don't really need ALL of the women in Miami. This is why non-selective indexes aren't all that useful on lookup tables.

This rule is really important for lookup tables, but what if you aren't trying to look up a single specific record? What if you're interested in a range of records? Well, let's look at...

The Yellow Pages: Another Index

When we need to find a dog groomer, we don't want to go shuffling through the white pages looking for anything that sounds like a dog groomer. We want a list of organized by business category:

- Business Category
- Business Name
- Address 1
- Phone Number

Then we'll look at the list of businesses, see which name sounds the coolest and which address is closest to ours, and we'll call a few of them. We'll work with several of the records. Here, we're searching for a range of records, not just a single one.

Notice that we didn't put the most selective field first in the index. The field "Business Name" is more selective than "Business Category". But we put Business Category first because we need to work with a range of records. When you're building indexes, you not only need to know what fields are important, but you have to know how the user is fetching records. If they need several records in a row next to each other, then it may be more helpful to arrange the records like that by carefully choosing the order of the fields in the index.

Learning More About Indexes

Indexes are really important, so we'll be covering these in more depth in the next two emails. In the meantime, here's a few great resources on getting started with indexes:

Our index resources page - where we've got posts and videos about heaps, indexing for deletes, partitioning, and more.

Our Indexing videos - free 30-minute videos on indexing mistakes, DBA Darwin Awards, and how to design smarter indexes with the DMVs.

SQLServerCentral's Index Stairway - a 15-part series by David Durant that goes all the way to indexing internals.

Expert Performance Indexing by Jason Strate and Ted Krueger (book) - covers how indexes work and how to pick the right one. Also available on Kindle.

And finally, if you'd like a video training session with our very own Microsoft Certified Master Kendra Little, we've got a 6-hour set of videos complete with quizzes and demos:

How to Tune Indexes and Speed Up SQL:

6-Hour Training

BrentOzar.com/go/indextunin
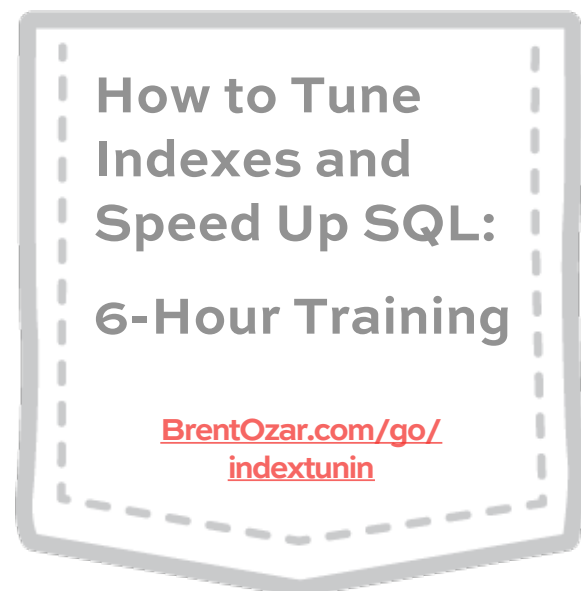
More Kinds of Indexes

Covering, filtered, full text, XML, heaps, 3x5 index cards, you name it.

by **Brent Ozar**

Last time, we talked about the two most common types of indexes - clustered and nonclustered. In this week's episode, we're going to spend just a paragraph or two covering the other types of indexes, starting with covering indexes. HA! Get it? We're covering covering indexes. Oh, I kill me.

Covering Indexes

Covering indexes aren't actually a different kind of index - it's a term that is used in combination with a query and an index. If I have this query:

```
SELECT FirstName, LastName, PhoneNumber
FROM dbo.People WHERE LastName = 'Ozar'
```

And if I have this index:

```
CREATE INDEX IX_LastName_Includes ON
dbo.People (LastName) INCLUDE
(FirstName, PhoneNumber)
```

Then the index covers all of the fields I need to run this query. SQL Server will start by looking up all of the Ozars by last name, and then the index includes the FirstName and PhoneNumber fields. SQL Server doesn't have to go back to the clustered index in order to get the results I need. This means faster queries, plus less contention - it leaves the clustered index (and the other nonclustered indexes) free for other queries to use.

Covering indexes are most effective when you have very frequent queries that constantly read data, and they're causing blocking problems or heavy IO.

Filtered Indexes

Say we're a big huge online store named after a river, and we constantly add records to our dbo.Orders table as customers place orders. We need to query orders that haven't been processed yet, like this:

```
SELECT OrderNumber FROM dbo.Orders
WHERE OrderProcessed = 0
```

The vast majority of the Orders records will have OrderProcessed = 1, because we keep all of our order history in this table. If we create an index on the OrdersProcessed field, it's going to have a lot of data - but we're never going to run queries looking for OrderProcessed = 1. Starting with SQL Server 2008, we can create an index with a WHERE clause:

```
CREATE INDEX IX_OrderStatus ON
dbo.Orders (OrderNumber) WHERE
OrderProcessed = 0
```

Now, we have an index on just a few records - a small subset of our table!

Full Text Indexes

Let's say we have a table called dbo.MoviePlots, and it had a Description field where we put each movie's plots. We know we liked this one movie where a guy was afraid of snakes, but we couldn't remember the exact table. We could write a query that says:

```
SELECT * FROM dbo.MoviePlots WHERE
Description LIKE '%snakes%'
```

But that wouldn't be very efficient. SQL Server would have to look at every movie's description, and scroll through all of the words one character at a time looking for snakes. Even if we index the Description field, we're still going to have to scan every row.

Why did it have to be snakes?

Full text indexes break up a text field like Description into each word, and then stores the list of words in a separate index. They're blazing fast if you need to look for specific words - but only as long as you rewrite your query to use the full text search commands like this:

```
SELECT * FROM dbo.MoviePlots WHERE
CONTAINS(Description, 'snakes')
```

You can even do fun stuff like look for synonyms or variations on a word. To learn more about full text indexing, check out:

- Books Online on Full Text Indexing - seriously, stop laughing, the manual's really good. This link is for SQL 2014/2012, but keep in mind that there were changes from 2008 to 2012.
- Understanding Full Text Indexing by Robert Sheldon - covers SQL Server 2008, plus the differences between 2005 and 2008 (which were huge).

XML Indexes

You can store XML data natively in SQL Server tables using XML fields. SQL Server is aware of the contents - in the sense that SQL Server knows the content is valid XML - but it's not necessarily smart about searching the data.

When you run XML queries, SQL Server has to roll up its sleeves and parse the XML data. Every. Single. Time. That's CPU-intensive, and a recipe for slow performance. Instead, we can create pre-processed versions of the XML so we can rapidly jump to specific nodes or values.

- Primary and Secondary XML Indexes - Books Online.
- Selective XML Indexes - instead of wasting a ton of space indexing values we never use, SQL Server 2012 can create the equivalent of filtered indexes on XML.

Heaps

Heaps are tables with no clustered index whatsoever. They're tables stored in random order, data slapped in any old place that fits. When you want to query a heap, SQL Server scans the whole freakin' thing. Every. Single. Time.

Sounds bad, right? Most of the time, it is - except for a couple of very niche uses. If you have a log-only table, meaning theres inserts but never any updates, deletes, or selects, then a heap can be faster. If you have a

staging table in a data warehouse, where you shove a lot of data in quickly and then need to get it all out at once, then delete all of it, a heap can be faster. Just make sure you test it to make sure it's actually faster under your application's needs.

Picking the Right Indexes for Your Apps

SQL Server's Dynamic Management Views (DMVs) surface a lot of useful instrumentation like which indexes are getting used, which ones aren't getting used, and which ones SQL Server wishes it would have had. Unfortunately, they don't give you a holistic overall picture - they're just raw data that has to be manually combined and interpreted. We'll talk about that in coming lessons, but you need to have this ground knowledge of index options first.

Dude, Who Stole My Missing Index Recommendation?

by [Kendra Little](#)

Recently, [Jes](#) asked the team an index tuning question: "If a query has an index hint in it, will the optimizer ever suggest a missing index for that query?"

I immediately loved the question because I'd never really thought about it before. I typically think of index hints as being a very risky game and avoid them whenever I can– after all if someone drops the index you've hinted, any query hinting a non-existent index will start to fail. (That's [a really bad day](#)!)

Even so, some people love index hints…

Read the full story: [www.brentozar.com/archive/2013/07/dude-who-stole-my-missing-index-recommendation/](#)

How to Master Index Tuning in One Step

by [Kendra Little](#)

I'm going to tell you a secret. Index tuning is complicated, but it's something you can become great at. You just need to practice it regularly.

Here's that one step: stop thinking index tuning is a problem for Future You.

That's it. Really. If you read this headline and didn't skip the post, your job probably involves helping an application using SQL Server go faster. If that's the case, index tuning is a problem for Present You. It's a problem for you now, it's a problem for you next month, and it's still a problem the month after that.

Index tuning isn't something you do once a year. It's something that you need to do iteratively– that means every month. Over time, data sizes change, user activity changes, and the SQL Server optimizer changes. Each of these things mean that indexes that are best for an application will alsochange. As you tune indexes, your query plans will change and you're very likely to see more opportunities to add, drop, and combine indexes emerge. Because of this, you want to do a few changes every month.

I hear a lot of reasons why people don't tune their indexes…

Read the full story: [www.brentozar.com/archive/2013/08/how-to-master-sql-server-index-tuning/](#)

The Parable of Index Maintenance

Indexes are like cars. You have to maintain them, you're probably not doing it, and if you take them to a mechanic, you'll probably get overcharged.

by [Kendra Little](#)

Once up on a time, there was a database server with 500GB of data and a heavy read workload of dynamic queries. Data was updated frequently throughout the day and index tuning was a serious challenge. At the best of times, performance was dicey.

Things went bad

Application performance plummeted. Lots of code changes had been released recently, data was growing rapidly, and the hardware wasn't the absolute freshest. There was no single smoking gun-- there were 20 smoking guns!

A team was formed of developers and IT staff to tackle the performance issue. Early in the process they reviewed maintenance on the database server. Someone asked about index fragmentation. The DBA Manager said, "Of course we're handling fragmentation!" But a few queries were run and some large, seriously fragmented indexes were discovered in production.

The DBA explained that fragmentation wasn't the problem. He didn't have automated index maintenance set up, but he periodically manually defragmented indexes that were more than 75% fragmented.

Bad, meet ugly

The whole performance team flipped out.

Trust disappeared. Managers squirmed. More managers were called in.

The DBA tried to change the subject, but it was just too late. More than a week was wasted over Fragmentation-Gate. It was a huge, embarrassing distraction, and it solved nothing.

Here's the deal-- the DBA was actually right. Fragmentation wasn't the root cause of the performance problem. But he made a miscalculation: he should have set up occasional automated index maintenance to align with his team's normal practices and standards.

Why you need automated index maintenance

When performance gets bad, one of the very first things people look at is whether systems involved are configured according to best practices. If you're not following a best practice, you need a good reason why.

Regular index maintenance still has a lot of merit: even in Shangri-La, where your data all fits into memory and your storage system is a rockstar with random IO, index maintenance can help make sure that you don't have a lot of empty space wasting loads of memory.

It's still a good idea to automate index maintenance. Don't go too crazy with it-- monitor the runtime and IO use and run it only at low volume times to make sure it helps more than it hurts.

How much downtime can you spare?

Before you implement index maintenance, find out how much time tables can be offline in each of your databases.

If you've got SQL Server Standard Edition, index rebuilds are always offline.

Even with SQL Server Enterprise Edition, you can specify an online rebuild unless the index contains large object types. (This restriction is relaxed somewhat in SQL Server 2012).

Partitioned tables are especially tricky. You can rebuild an entire partitioned index online, but partition level rebuilds are offline until SQL Server 2014.

Maintenance plans or custom scripts?

You can go the easy way and use SQL Server Maintenance Plans, but unfortunately they're very simplistic: you can only say "rebuild all the indexes" or "reorganize all the indexes". You cannot say, "If the index is 45% or more fragmented, rebuild it-- otherwise do nothing." If you don't spend much time with SQL Server and you've got downtime available every weekend, this can be a decent option.

If you need to minimize downtime, custom index maintenance scripts are the way to go. Our favorite: Ola Hallengren's maintenance scripts. These are super flexible, well documented, and … free! The scripts have all sorts of cool options like time boxing and statistics maintenance.

Download and configure them on a test instance first. There's a lot of options on

Our Best Free SQL Downloads includes video tutorial on Ola's script setup.

BrentOzar.com/go/freebies

parameters, and you'll need to play with them.

Get used to the 'cmdexec' job step types. When you install the scripts you'll see that the SQL Server Agent jobs run index maintenance using a call to sqlcmd.exe in an MSDOS style step. That's by design!

Use the examples on the website. If you scroll to the bottom of the index maintenance page you'll find all sorts of examples showing how to get the procedure to do different useful things.

Find out when maintenance fails

Don't forget to make sure that your maintenance jobs are successfully logging their progress. Set up Database Mail and operators so jobs let you know if they fail.

Tell your boss you did a good thing

Finally, write up a quick summary of what you did, why you chose custom scripts or maintenance plans, and why. Share it with your manager and explain that you've set up automated index maintenance as a proactive step.

Having your manager know you're taking the time to follow best practices certainly won't hurt-- and one of these days, it just might help you out.

Learning More About Fragmentation

5 Things About Fill Factor - Including what it's for, what it's NOT for, and why you shouldn't play with that.

Stop Worrying About Fragmentation - defragging everything can cause more problems than it solves.

Pop Quiz!

Time to find out if you've been reading, or just "scanning".

Backup & Recovery Questions:

1. How many production SQL Servers do you have?

2. What's the RPO and RTO of your most important production server?

3. Did your backups take the normal amount of time last night?

4. When was the last time DBCC successfully finished in production?

Security Questions:

1. How many different people are sysadmins in production?

2. Do they each know that they're sysadmins, and take care to avoid accidents?

3. How many of your databases hold personally identifiable data like credit card numbers, social security numbers, and passwords?

4. If someone gets hold of one of those database backups, can your company's data go public?

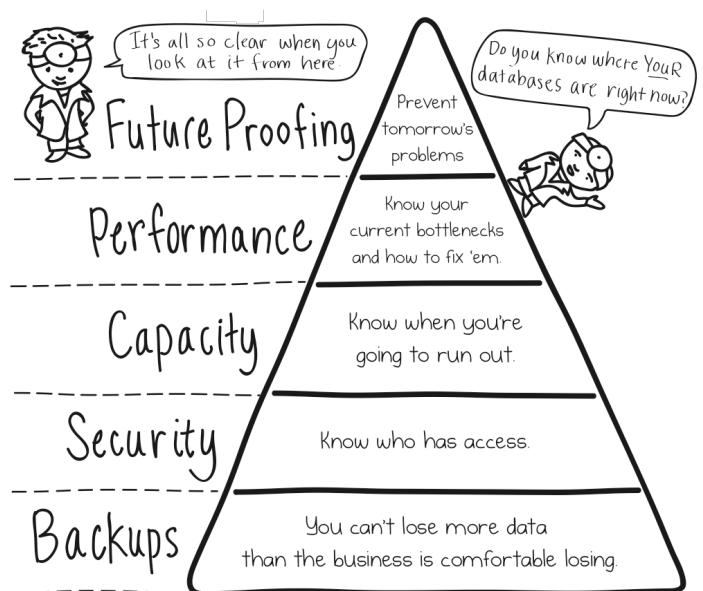5. Have you informed your managers of that risk, or will they blame you?

Monitoring Questions:

1. When a database server runs out of drive space, who gets emailed?

2. Do at least two different people get the email in case one is on vacation or unavailable?

3. What actions will you take to fix the situation?

4. Are those actions documented so that everyone who gets the email can take action quickly?

Index Questions:

1. Does every table in production have a clustered index?

2. For any exceptions (heaps), do you have a plan to fix them?

3. Which table in production has the most indexes, and why?

4. Which frequently queried tables in production have the least indexes, and why?

5. How are you managing index fragmentation?

It's all so clear when you look at it from here.

Do you know where YouR databases are right now?

Future Proofing — Prevent tomorrow's problems

Performance — Know your current bottlenecks and how to fix 'em.

Capacity — Know when you're going to run out.

Security — Know who has access.

Backups — You can't lose more data than the business is comfortable losing.

The Right Answers

There's no one right answer for any of these questions, but some answers are more wrong than others. Database administration is a journey, and not everyone in the company is going to appreciate the work you're putting in. You can spend weeks or months trying to improve your answers on these. No matter how big your company is and how many database administrators you have, you're probably never going to be truly happy with your answers here.

You're not aiming for perfect.

You're aiming for good enough that your managers accept the base of your Database Hierarchy of Needs pyramid, and that you feel confident in tackling the higher levels like performance and future-proofing.

Next week's email is going to start digging into performance, and we're going to ignore the lower levels of the pyramid - but that doesn't mean that part of your journey is

Don't make changes without a tested safety net.

over. Print out this email, cut out the question list, and scribble in a few thoughts. Pin it up on your wall, and a few months from now, when you're feeling overconfident that your environment is awesome, check that list again. Refresh your memory with the links on the right side of this email.

When an outsider comes in, like a support engineer or a consultant, they're going to start at the base of your pyramid first. Before they start to help, they have to make sure your data is backed up and checked for corruption. They can't go making changes without having a safety net.

And you shouldn't either.

Remember that as we start talking about changing server and database settings next.

What is SQL Server Waiting On?

As SQL Server runs queries, it constantly tracks what it waits on.
Ask SQL Server for these wait statistics, and tuning is easy. Easier, anyway.

by [Brent Ozar](#)

You probably got into database administration by way of development or systems administration. You're used to monitoring stuff from the OUTSIDE using things like Performance Monitor counters.

SQL Server has a way, way, way better tool.

When SQL Server starts running your query, your query consumes CPU. It sits on a CPU scheduler, using as much CPU as it can, all to its greedy self. SQL Server doesn't carve up a core and say you can run for 3 seconds until someone else gets to run - oh no. Your query runs until it's done, burning CPU the whole time.

Until it has to wait for something.

The instant your query has to wait - like if SQL Server needs to read data from hard drives, or wait for someone else to let go of a lock - then your query steps off the CPU and goes into a waiting queue. SQL Server tracks how many milliseconds your query spends waiting, and what resource it's waiting on.

You might be surprised at how little memory is used for caching data.

The Crappy Way to Check Waits

Run this simple query:

```
SELECT * FROM sys.dm_os_wait_stats
```

And you'll get back a list of wait types, plus how many milliseconds the server has spent waiting on this wait type. The time is cumulative, measured over time since the SQL Server instance was started - or since someone manually cleared the wait stats. (Don't do that.)

There's a few problems here: the wait list is really cryptic, it's filled with irrelevant system wait types, and it's measured over time. What you really want is a quick snapshot of waits over a 30-second period, with the system wait types filtered out.

The Better Way to Check Waits

Hit BrentOzar.com/go/waitsnow and get our free script. This returns 3 result sets - the first showing how long the server has been up, the second showing the waits since they were last cleared, and the third is a fun one. The third shows a running sample of waits over the last 30 seconds.

If your server is busy, you'll see MORE than 30 seconds of waits on the biggest bottlenecks. That's totally normal because your SQL Server has multiple cores, each of which may have multiple queries lined up waiting for something.

If your server isn't busy, your waits will probably add up to 30 seconds - or much

less. This just means the server's mostly sitting around idle. Don't make big performance tuning decisions based on small samples like that - aim for sampling when the server's really busy.

To learn more about wait types and what they mean, check out our wait types resources page.

Once you get started with wait stats, you'll want to capture this data all the time and trend it. Don't reinvent that wheel: every modern performance monitoring tool tracks wait stats already.

Correlating Waits Stats and Perfmon

Once we think we've got a bottleneck, we need to double-check those numbers by gathering server-level metrics about that particular bottleneck.

In our Performance Monitor tutorial, we explain how to set up Perfmon, gather the right metrics, and export them to a spreadsheet. Depending on the wait stats you're seeing as a bottleneck, here's the Perfmon counters to collect:

To double-check CXPACKET and SOS_SCHEDULER_YIELD waits, collect:

System: Processor Queue Length
(for each individual core, not the total)

This wait type indicates challenges with parallelism. Parallelism isn't a bad thing - it means SQL Server is breaking out your large queries into multiple tasks, and spreading that load across multiple processors. Learn more about CXPACKET waits.

For PAGEIOLATCH* and WRITELOG waits:

Physical Disk: Avg Sec/Read

Physical Disk: Avg Sec/Write

Physical Disk: Avg Reads/Sec

Physical Disk: Avg Writes/Sec

The top two counters are about response time - how fast the storage is returning results. The bottom two counters are about how much work we're giving to the storage. Much like you, the more work you're asked to do, the slower you get. The top two are the SAN person's fault, the bottom two are your fault. You want to make sure the bottom two numbers trend down over time by doing better indexing.

You need to figure out how to help an ailing SQL Server.

Our SQL Critical Care™ helps.

brentozar.com/go/help

For LCK_* waits, collect:

SQLServer: Locks - Lock Waits/sec

SQL Server: Locks - Avg Wait Time

SQL Server: Access Methods - Table Lock Escalations/sec

SQL Server: Transactions - Longest Running Transaction Time

SQL Server: Access Methods - Full Scans/sec

These counters help you determine how much locking is going on, and where the source is. For example, when you're having locking problems, and the Full Scans/sec reports a high number, maybe you've got a lot of table scans going on, and those are grabbing locks across tables while they work. Or maybe Longest Running Transaction Time is reporting a few minutes - meaning someone is running a really long transaction, and it's starting to block lots of other users.

Got Other Waits?

Wait types can be so cryptic - there's so many of them, and they're often not documented well. To learn more about wait types and what they mean, check out our wait types resources page.

Learning More About Perfmon Counters

No matter what your biggest wait type is, the idea here is to correlate that wait type with underlying Perfmon counter measurements to drill down deeper and find the root cause of the problem. To learn more, here's our favorite resources:

Perfmon Counters of Interest Poster - from Quest Software, listing the best counters by type.

Our Perfmon tutorial - explaining how to collect the data and analyze it.

Jimmy May's Perfmon Workbook - an Excel spreadsheet with Jimmy's favorite counters and thresholds.

Watch Brent explain wait stats while he's dressed up as Richard Simmons. Click here.

Get More Help from Us

We've got more tricks than a pony

by Brent Ozar Unlimited®

We love - no, we LOVE - helping people get relief for data pains, and we've got lots of options to help.

FIRST AID: TOTALLY FREE STUFF

We build cool troubleshooting tools and give them away for free:

- sp_Blitz® - fast SQL Server health check.
- sp_BlitzIndex® - identifies indexing madness dragging down your SQL Server.
- Our blog - thousands of articles on performance tuning, availability, and career development.
- Much more - like our posters, YouTube videos, and weekly webcasts.

IN-PERSON TRAINING CLASSES ACROSS THE UNITED STATES

Our classes are taught by real experts with hands-on knowledge - specifically, us. We share the latest cutting-edge tips and tricks that we've learned in real-life deployments.

We're available for questions and answers - it's your chance to talk face-to-face and get personal advice on your tough challenges.

Join us in-person at our classes!

VIDEO COURSES:
$29-$299 FOR IN-DEPTH KNOW-HOW

- How to Think Like the SQL Server Engine, $29 - Kick start your performance tuning with insight into SQL Server's brain.
- DBA Job Interview Question and Answer Kit, $29 - Practice questions and more.
- Virtualization, SANs, and Hardware for SQL Server, $299 - 5 hours of subsystem secrets.
- How to Tune Indexes and Speed Up SQL Server, $299. 6.5 hours of quizzes, scripts, and more!

You can watch our high-definition training from your desktop, laptop, or even your iPad for 18 months.

That's just a sample - check out the full list.

SQL CRITICAL CARE®:
A FASTER, SAFER SERVER IN 4 DAYS.

Tired of struggling with a slow, unreliable SQL Server?

In just 4 days, we'll work together with you to get to the root cause, explain your options, and give you a simple, prioritized action plan to make the pain stop.

We don't keep secrets: you get to keep our scripts, and you watch us work. It's like the best conference training, but in your own environment.

Schedule a free 30-minute consultation with us and learn more about our SQL Critical Care®: email Help@BrentOzar.com.