Global Knowledge ®

Expert Reference Series of White Papers

Top Ten Things Every DBA Should Know About SQL Server

Top Ten Things Every DBA Should Know About SQL Server

Brian D. Egler, Global Knowledge Course Director, MCITP, MCT 2010

Introduction

Microsoft SQL Server has evolved over the years as a scalable, robust database management system and is now competing in the VLDB (Very Large Database) space with Oracle and IBM. The market share for the product continues to grow, based on total cost of ownership and ease of use. This white paper outlines some of the important fundamentals of Microsoft SQL Server that every DBA should know.

Top Ten Things Every DBA should know about SQL Server:

1. A Full Backup Does NOT Truncate the Transaction Log
2. Transaction Log Not Shrinking? OK, Shrink it Twice
3. Yes, You Can Recover your Data Right up to the Point of Failure
4. Data Partitioning Improves Performance for Large Tables
5. Security - Ownership Chains Are a Good Thing
6. Has Microsoft Deprecated your SQL Code?
7. DTS Has Been Deprecated, Get on the SSIS Bandwagon before 2011
8. Data Compression Improves Performance, too
9. Change Data Capture Simplifies the Incremental Load
10. Books Online Is Still the Best

1. A Full Backup Does NOT Truncate The Transaction Log

A frequent question that students bring to class when I am teaching is this: "My transaction log is growing out of all proportion - what can I do?" I always answer with another question "Are you backing it up?" The student usually replies, "Of course, we backup the entire database every night." "But are you backing up the LOG?" At this point the student lets me know that if we solve this dilemma, the boss will feel the cost of the class will be worth it right there and then.

Backing up the transaction log will not only backup the latest committed transactions within the log but will also truncate the log file. Truncation means that the transactions that were backed up are removed from the log file, freeing up space within the log file for new transactions. The truth is, if you don't backup the transaction log, it will continue to grow forever, until you run out of disk space. New DBAs to SQL Server assume that the Full Database Backup truncates the transaction log - it doesn't. It does take a snapshot of the transaction log

at the very end, so that transactions committed during a long-running full backup are actually backed up, too (which is quite clever), but it does not truncate the log. So students come to class with a 100MB Database that has a 16GB Transaction Log. Yikes. Well, the long-term solution is to backup the transaction log frequently. This will keep the transaction log "lean and mean." But how frequent is frequent? Well, it depends. Generally, we try to keep the transaction log under 50% of the size of the data files. If it grows beyond this, then we back it up more frequently. This is why, for some very active databases, we may backup the log every 15 minutes. As I said, it depends.

This discussion assumes a database Recovery Model of "Full," which is the normal recommendation for production databases. This setting allows the backup of the transaction log so that you can recover up to the point of failure by restoring log backups in order. A Recovery Model of "Simple" will automatically truncate the transaction log periodically, but you are not allowed to backup the log in this mode, so you can only recover using the last database backup, which implies potential loss of transactions. This setting is usually reserved for test or read-only databases.

2. Transaction Log Not Shrinking? OK, Shrink It Twice

OK, so let's get back to the 16GB Transaction log file. What should we do to get this back to a reasonable size? Well, backup the log first of all. It contains some valuable updates. That will truncate the log but will not make the file any smaller. Secondly, perform a DBCC SHRINKFILE operation. Now, when we do this, we may not see any shrinkage, so let's refer to Books Online, which states, in its wisdom: "it may take two backups to actually free the space." Yes, if at first you don't succeed, try again. The actual solution is to run the BACKUP LOG/DBCC SHRINKFILE sequence twice!

The second time around, you should see significant shrinkage.

Reference: Managing the Size of the Transaction Log File (on MSDN)
http://msdn.microsoft.com/en-us/library/ms365418.aspx

3. Yes, You Can Recover your Data Right up to the Point of Failure

SQL Server is architected so that you can recover ALL of the committed transactions right up to the point of failure. This fact did not go unnoticed by financial services companies on Wall Street during the early days of distributed applications - the early days of SQL Server. To these companies, even a few minutes of lost transactions can mean millions of dollars - yes, time is money. But we still need to practice disaster recovery, so that when disaster does strike, we are ready to play the hero. As a colleague of mine always says, you want to avoid any "resumé-generating events!"

The point of failure recovery assumes that you can back up the "tail" of the transaction log AFTER the failure. The tail of the log contains "gold dust" - all those latest transactions that have committed since the last backup. That is why we store the log file on RAID-1 mirrored volumes so that even in the event of disk failure, we can

get a copy so we can back it up. In this way, the log file is the most important file in the database; the data files can always be recovered from a full backup, but those last minute changes before a failure are only contained in the active transaction log. So, whatever you do, make sure you backup the active transaction log before a restore operation. In fact, when teaching, I always tell students to remember to do two things before attempting a classic database restore:

1. Restrict access to the database (stop people updating the database!)
2. Backup the "tail" of the transaction log (capture those last-minute changes!)

And with the Enterprise Edition, you can now perform Online Restores; you may not have to restrict access at all. This means that you only need to remember ONE thing before a restore operation: Backup the "tail" of the transaction log!

If you remember this, then you can rest assured that you will be the hero.

Reference: Tail-Log Backups on MSDN
http://msdn.microsoft.com/en-us/library/ms179314.aspx

4. Data Partitioning Improves Performance For Large Tables

Data Partitioning was a new feature of SQL Server 2005 that enables you to store large tables across multiple devices, giving performance and manageability benefits. It's a feature that has been in Oracle and DB2 for a while, but now Microsoft is competing with the big guys in the VLDB space so this scalability feature is very important. Multi-Terabyte databases are becoming more common, especially in the Business Intelligence arena. The idea is to be able to store a table's rows on different devices based on a column value within that row, for instance, Calendar Month, so that all similar rows within a table are stored in one place. So we may choose to store each month's data on a separate "partition," which could be on a separate physical device. When we need data from one particular month, in this example, the system would only need to access one device. If we query data across many months, we have multiple disk volumes working in parallel giving better performance, too. SQL Server uses the Filegroup object in order to define partitions. A "partition function" defines the boundary values of the partitions, a "partition scheme" defines the Filegroups that represent partitions and the CREATE TABLE statement identifies the partition scheme (and its associated function) to be used. The system will then work out, based on a single column value, which partition a new row should be stored on.

However, the manageability features, although not so obvious, may be just as important in the long run. Say we have to store the last 12 months of order data for a large multi-national retail company in a table. We may have millions or even billions of rows of data. We also want to archive data older than 12 months to an archive table. What we would be faced with, without partitioning, is copying millions of rows at the end of each month to an archive table with the same structure. Well, with partitioning, we can just flick a switch (actually the ALTER TABLE SWITCH statement) and the partition instantly becomes part of the archive table with no data movement necessary. That could save a lot of time in our already busy 24x7 environment. To create a new empty partition for the new month's data, we can use the SPLIT RANGE clause of the ALTER PARTITION FUNCTION statement. To remove unused empty partitions we can use the MERGE RANGE clause of the same statement. In this way, we can define a "Sliding Window" of data with no need to copy and delete huge amounts of data. Also, because in-

serting data into a new partition is an order of magnitude faster than to an existing one, this strategy is popular for incremental loads for a Data Warehouse.

Reference: Designing Partitions to Manage Subsets of Data

http://msdn.microsoft.com/en-us/library/ms191174.aspx

5. Security – Ownership Chains Are A Good Thing

Ownership Chaining enables managing access to multiple objects, such as multiple tables, by setting permissions on one object, for instance a view. It is a powerful technique used for controlling access to data, enabling the creation of a "security layer" defined by Views and/or Stored Procedures, such that authorized users can access a View or execute a Stored Procedure without being able to access the base tables directly. In this way, Ownership Chaining simplifies the security model greatly. This technique has been used in previous releases and is now also supported using schema object in SQL Server 2005 and beyond.

The rule is that if all the database objects referenced in a View, or Stored Procedure, or Function are owned by the same user, then SQL Server will only check permissions at the highest level. SQL Server assumes that because all the objects have the same owner, that the owner knows what they are doing when they set permissions on the top level object. This is an unbroken ownership chain and is a good thing. It also has the effect of creating a security layer because the underlying database objects are protected and are not accessible by the calling user. An easy way to guarantee an unbroken ownership chain is to have all objects in a database created by the same user, preferably the database owner. Conversely, if any of the objects are owned by other users, we have a broken ownership chain (not good); SQL Server will be forced to check permissions at all levels and, therefore, access will be denied, unless we set permissions at all these levels. This is not only difficult to administer, but it also opens up access to lower level objects that we wanted to protect.

With the introduction of the Schema object in 2005, the owner is no longer apparent in the qualified name of the object; however, ownership chaining still applies. If you own a Schema, you effectively own all the objects contained within it. Once again, as long as all the objects are owned by the same owner, we have an unbroken ownership chain and creating security layers is a piece of cake.

6. Has Microsoft Deprecated Your SQL Code?

When it comes to new releases of Microsoft SQL Server, what can you expect? New features, we hope. But one critical thing everyone should want to know is: Which old features were put onto the dreaded "Deprecated Features" list? "Deprecated" sounds like an awful word but it merely refers to features that are "going away." This list, issued by Microsoft, has two categories: 1) "Features Not Supported in the Next Version of SQL Server" and 2) "Features Not Supported in a Future Version of SQL Server."

For instance, two of the most important features gone from SQL Server 2008 are:

BACKUP LOG WITH NO_LOG

and

BACKUP LOG WITH TRUNCATE_ONLY

These two statements, which are functionally equivalent, have been around since the early days of SQL Server and were typically used to truncate the transaction log when it became full (the infamous error code 9002). The beauty of these statements was that they would not actually backup the transaction log at all, they would just truncate the log and free up space. The downside of the statements was that they would break the "chain" of log backups, exposing the database to potential data loss until a full database backup could be made. Now Microsoft is trying to protect us from ourselves and has outlawed these statements in 2008.

Now, to be fair, Microsoft always gives advance warning to the SQL Server community of the features that will be removed in future versions. As you can imagine, sometimes the list can cause a bit of a panic for database developers. But speaking from experience, the panic is usually unfounded. Microsoft has done a reasonable job with backward compatibility in the past so, in general, they do tend to preserve older features even as new ones are introduced.

And when I say fair warning, I mean it. For instance, in the "Deprecated Database Engine Features in SQL Server 2005" document on MSDN (which was first issued when SQL Server 2005 was released), number one in the list of "Features Not Supported in the Next Version of SQL Server" is the DUMP statement. This means that the DUMP statement is supported in SQL Server 2005, but we have been served notice that it won't be supported in the next release, i.e., SQL Server 2008. This is interesting for me, since I used to work for Sybase in the mid-90s; you may or may not know that Sybase invented SQL Server in the '80s and with it the DUMP statement to "dump" or backup databases.

Now bear with me as we go through a history lesson. Microsoft was looking for a database product at the time, so an agreement between the companies allowed Microsoft to port the SQL Server product to its own operating systems instead of the Sybase-preferred OS of UNIX. Microsoft SQL Server was born. However, Microsoft decided to terminate this agreement because they wanted control over the source code, which had been previously dictated by Sybase developers. The first major Microsoft-only release was Microsoft SQL Server 6.0. In Microsoft SQL Server 7.0, released in 1998, the DUMP statement was replaced by the BACKUP statement. Microsoft obviously did not like the word DUMP; therefore, they wanted it "deprecated." That is when the DUMP statement first appeared on the "Deprecated List" under the "Features Not Supported in a Future Version of SQL Server" category. So according to my calculations, it took around ten years for the DUMP statement to actually disappear from the SQL Server product after first appearing on the infamous "Deprecated List."

In other words, don't panic until it actually happens. (But you have been warned!)

References: Deprecated Database Engine Features in SQL Server 2005
　　　　　http://msdn.microsoft.com/en-us/library/ms143729(SQL.90).aspx

Deprecated Database Engine Features in SQL Server 2008
　　　　　http://msdn.microsoft.com/en-us/library/ms143729.aspx

7. DTS Has Been Deprecated, Get On The SSIS Bandwagon Before 2011

Microsoft has told us that DTS has been deprecated. It means we have been warned that it will no longer be supported in a future release. When SQL Server 2008 comes out shortly, Microsoft will only support two previous releases namely SQL Server 2000 and 2005. So any DTS packages running under SQL 2000 will continue to be supported until the next release, presumably slated for 2011, if Microsoft sticks to its stated policy of three-year intervals. In other words, we have until 2011 to migrate our DTS 2000 packages to SSIS.

The migration of DTS packages caused a bit of a stir during the Yukon Beta back in 2005 when Microsoft let us know that 100% of our DTS code may not be translated into the "new DTS." You see, DTS was rewritten from the ground up with a new architecture to provide more scalability and, eventually, was renamed to SQL Server Integration Services (SSIS). This name change was a good idea since it really is a very different animal from DTS, but the change was so late in the Beta program that developers had no time to rename the executables so you will see some DTS remnants "under the hood." For instance, we can execute an SSIS package using DTEXEC. Understandably, users who had spent 5 years or more maintaining their complex DTS packages were a little aggrieved and let Microsoft know that they would not upgrade if it couldn't guarantee a smooth migration.

So the Microsoft developers did a clever thing. They decided to allow the use of DTS within SQL Server 2005. Within SSMS there is a Legacy folder (under Management) that allows you to import an existing DTS package. And with a free download, you can install the good old DTS Designer and maintain and test your DTS packages within SSMS. This is not emulation software, it is the real thing, so anything that ran in DTS in 2000 will run in 2005. And now 2008. Complete backward compatibility. This means that we can decide if and when to migrate a particular package to SSIS on a case-by-case basis. We can even have a new SSIS package execute an existing DTS 2000 package if we wish. Co-existence - a good strategy. When we want to benefit from the new scalability of SSIS, we should use the SSIS Migration Wizard to migrate the package, but we have to remember to schedule time for code changes and re-testing of the package. Now we have a full Debugging facility within Visual Studio that makes things easier. In many cases, it may be necessary to rewrite the package from scratch, based on the new features of SSIS. The good news is that the Migration Wizard leaves the existing DTS package untouched while you work on the new SSIS package. The new extension is .DTSX (X for XML) as opposed to the old .DTS - another remnant.

The DTS backward-compatibility solution calmed everyone down, and SQL Server 2005 was released as planned. So co-existence is a fine strategy - until 2011, that is.

8. Data Compression Improves Performance, Too

Data Compression is available at last in SQL Server 2008. Even Backup Compression is available. This will help us save space and time. We have two types of compression available in SQL Server 2008: Data Compression and Backup Compression.

Data Compression can be performed on tables, indexes, and partitions. It can be defined at the row level or at the data page level. Both levels of data compression work especially well with variable length columns.

Page-level compression will also provide savings based on column values that start with similar data. You can estimate and enable data compression from T-SQL, but the easiest way is through the Data Compression Wizard, which can be started from SSMS. For instance, you can right-click a table and select Storage, then Manage Compression... from the context sensitive menu. Testing this out with the FactInternetSales table in the AdventureWorksDW sample database, I could see that row level compression would reduce the space requirements of that table down to around 12MB from 16MB. With page-level compression, that estimate dropped to 8MB. The trade-off is the extra processing to de-compress the data, but this is rated as minimal overhead of less than 10%. In fact, based on the compressed data taking up less data pages, less time to process, and less memory usage, many users are seeing an actual net gain in performance too.

Backup compression can be configured at the Server level. It is switched off by default but can be turned on as the default using sp_configure or SSMS. The default can be overridden on a backup by backup basis, if necessary. In my tests, I backed up the AdventureWorks2008 database, and the .bak file took up 187MB uncompressed. When using backup compression, it came down to 45MB. Pretty impressive. That translates to around 75% compression. And of course, the backup ran faster, too.

All in all, these new compression features in SQL Server 2008 are very positive with little or no overhead. Plenty of upside and not much downside.

9. Change Data Capture Simplifies The Incremental Load

One of the interesting new features in SQL Server 2008 is Change Data Capture (CDC). If you need to know the column data involved in an insert, update, or delete statement, then CDC is for you. For instance, you can use CDC in your incremental load package that copies changes to your Data Warehouse every night.

First of all, the CDC feature is only available in the Enterprise Edition (and the Developer Edition, which is the same bits with a different EULA; you just agree not to put anything into production). Microsoft rates this as an enterprise feature and wants you to pay for it. You will need to also buy off on the extra disk space that it will consume, because SQL Server will set up a CDC table for each table that you wish to capture changes for, and will save rows for each insert, update, and delete. Depending on how many updates occur against the enabled tables, this may be significant. There's also the extra processing to write the extra rows. Assuming this is all acceptable, CDC can be a big plus.

To enable CDC on a database, you use the system-stored procedure sp_cdc_enable_db. To identify a table that will capture changes, you use sp_cdc_enable_table. One of the options here is @supports_net_changes, which, when set to 1, will allow you to view net results when multiple updates are made to the same row. A special system table is created for each table enabled, within the special CDC schema in the enabled database.

In my tests, I enabled the FactInternetSales table in the AdventureWorksDW2008 sample database. SQL Server generated a table in the CDC schema named dbo_FactInternetSales_CT to capture changes. The CDC table contained all the columns from the source table plus 5 extra metadata columns to identify such things as the Log Sequence Number (LSN) and the operation code (1 for delete, 2 for insert, 3 for before an update, 4 for after the update). For each insert and delete on the source table, a row was generated in the CDC table. For each

update, 2 rows were generated; one for before the update and one for after, so you can clearly see what data has changed. SQL Server also generates special system functions that enable you to "get all changes" or "get net changes."

The new MERGE T-SQL statement in SQL Server 2008 goes hand-in-hand with CDC, as it allows data to be inserted, updated, or deleted from a single statement without complex conditional logic. The MERGE statement would use the CDC data as the source and a Data Warehouse table as the target.

As you can imagine, the CDC tables can get quite large, quite quickly, but since storage is relatively cheap these days, this is an elegant solution to capture data changes without the need for triggers or timestamp columns. You might be happy to hear that CDC is disabled by default, but if you need it, just turn it on.

10. Books Online Is Still The Best

When I am teaching a class, I always emphasize the importance of SQL Server Books Online. This Help system is, in my opinion, the best in the business. The proof is that even the experts use it on a regular basis. It is not just a beginner's tool. It is full of valuable information, especially working examples.

When Sybase originally developed SQL Server, it embarked on the Books Online strategy for a Help system. At the time, most products would ship with a ton of manuals in hardcopy that would sit on the shelf and would rarely be used. Or if you did need the manual, it was always in the "other" office or had been mislaid. The more expensive the product, the more manuals you received. It was product by the pound. But Sybase, to its credit, saw the green side of things and started thinking about electronic documentation – Books Online (BOL) was born.

To give Microsoft its credit, it has committed the resources to maintaining BOL over the years, and it shows. The BOL content is updated continually, and the code examples are accurate and useful. The downloadable version of Books Online is updated every couple of months, even between service packs. The 2005 and later versions allow you to access the "Online Help" on MSDN to get the latest and greatest assuming an internet connection is available. You can also switch this feature off if you do not have an internet connection, or if you just don't want to wait for the extra time for the downloads. If you do switch off Online Help, you are using the "local only" copy of BOL. Either way, use BOL as much as you can. It will definitely help.

Conclusion

Microsoft SQL Server is a sophisticated enterprise-level database management system. As a Database Administrator, the 10 items listed here are essential for you to understand. But there's much more to learn beyond the scope of this white paper.

Learn More

Learn more about how you can improve productivity, enhance efficiency, and sharpen your competitive edge. Check out the following Global Knowledge courses:

SQL Server 2008 for Database Administrators (M6231, M6232)

Developing and Implementing a SQL Server 2008 Database (M6232)

Implementing and Maintaining Microsoft SQL Server 2008 Integration Services (M6235)

Implementing and Maintaining SQL Server 2008 Analysis Services and Business Intelligence Solutions (M6234)

Maintaining, Troubleshooting, & Developing Solutions with Microsoft SQL Server 2008 Reporting Services (M6236)

For more information or to register, visit **www.globalknowledge.com** or call **1-800-COURSES** to speak with a sales representative.

Our courses and enhanced, hands-on labs offer practical skills and tips that you can immediately put to use. Our expert instructors draw upon their experiences to help you understand key concepts and how to apply them to your specific work situation. Choose from our more than 1,200 courses, delivered through Classrooms, e-Learning, and On-site sessions, to meet your IT and management training needs.

About the Author

Brian D. Egler is a Global Knowledge Instructor and Course Director specializing in Microsoft SQL Server technologies. He currently resides in Holly Springs, North Carolina.