SQL Server DBA Best Practices

By Brad M. McGehee

SIMPLETALK PUBLISHING

In association with redgate®

www.red-gate.com/publishing

ISBN 0-9759015-4-0

24.99>

Brad M. McGehee's

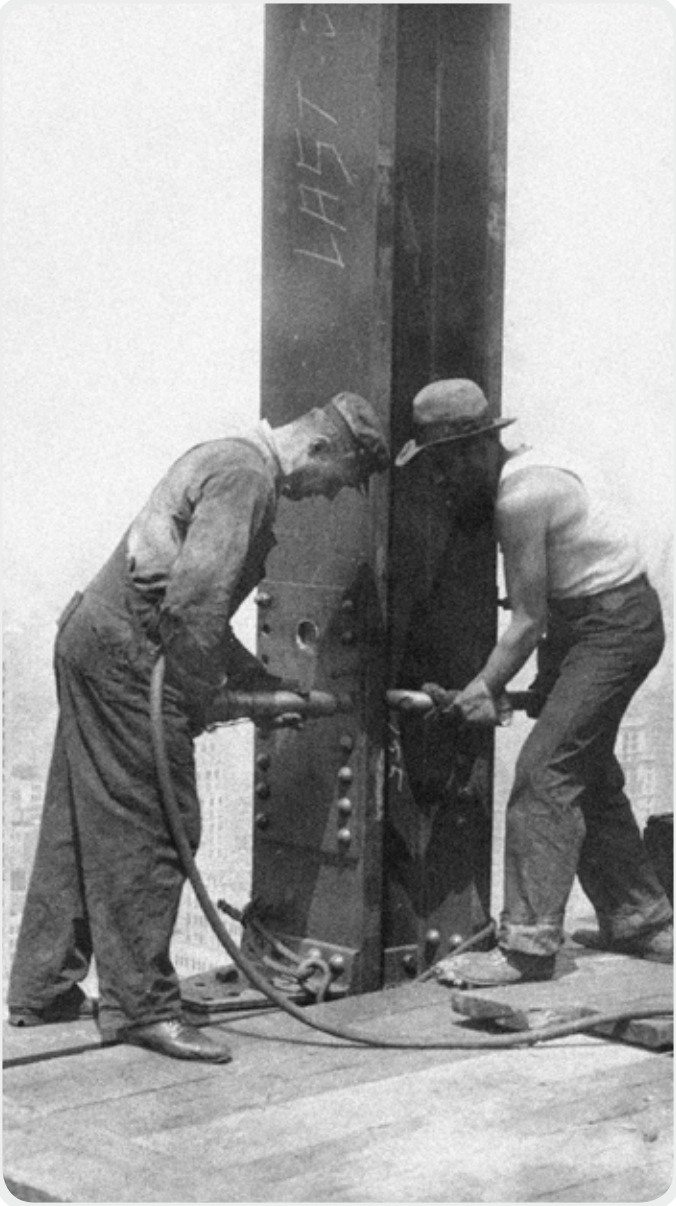
SQL Server DBA Best Practices

A bit about Brad..

Brad M. McGehee is a Microsoft SQL Server MVP with over 10 years' SQL Server experience.

He founded the popular community site SQL-Server-Performance.Com where he now acts as the technical editor and forum moderator.

Brad is also a frequent speaker at SQL PASS, SQL Connections, SQL Server user groups, and other industry seminars and he is the author or co-author of more than 12 technical books and over 100 published articles. He spends what time he has left in Missouri and Hawaii with his family.


LAST

General DBA Best Practices

Day to Day

1. Check OS Event Logs, SQL Server Logs, and Security Logs for unusual events.
2. Verify that all scheduled jobs have run successfully.
3. Confirm that backups have been made and successfully saved to a secure location.
4. Monitor disk space to ensure your SQL Servers won't run out of disk space.
5. Throughout the day, periodically monitor performance using both System Monitor and Profiler.
6. Use Enterprise Manager/Management Studio to monitor and identify blocking issues.
7. Keep a log of any changes you make to servers, including documentation of any performance issues you identify and correct.
8. Create SQL Server alerts to notify you of potential problems, and have them emailed to you. Take actions as needed.
9. Run the SQL Server Best Practices Analyzer on each of your server's instances on a periodic basis.
10. Take some time to learn something new as a DBA to further your professional development.

Installation

1. Always fully document installs so that your SQL Server instances can easily be reproduced in an emergency.
2. If possible, install and configure all of your SQL Server instances consistently, following an agreed-upon organization standard.
3. Don't install SQL Server services you don't use, such as Microsoft Full-Text Search, Notification Services, or Analysis Services.
4. For best performance of SQL Server running under Windows, turn off any operating system services that aren't needed.
5. For optimum SQL Server performance, you want to dedicate your physical servers to only running a single instance of SQL Server, no other applications.
6. For best I/O performance, locate the database files (.mdf) and log files (.ldf) on separate arrays on your server to isolate potentially conflicting reads and writes.
7. If tempdb will be used heavily, also put it on its own separate array.
8. Do not install SQL Server on a domain controller.
9. Be sure that SQL Server is installed on an NTFS partition.
10. Don't use NTFS data file encryption (EFS) and compression on SQL Server database and log files.

Upgrading

1. Run the Upgrade Advisor before upgrading. Make any necessary changes before performing the upgrade.
2. Perform a test upgrade of your test SQL Servers before you upgrade your production servers. And don't forget to test your applications with the new version also.
3. Before you upgrade, be sure you have a plan in place to fall back to in case the upgrade is problematic.
4. Don't upgrade SQL Server clusters in place. Instead, rebuild them on new hardware.
5. If you upgrade from a previous version of SQL Server, you should update all of the statistics in all your databases using either UPDATE STATISTICS or sp_updatestats. This is because statistics are not automatically updated during the upgrade process.

Security

1. Ensure the physical security of each SQL Server, preventing any unauthorized users to physically accessing your servers.
2. Only install required network libraries and network protocols on your SQL Server instances.
3. Minimize the number of sysadmins allowed to access SQL Server.
4. As a DBA, log on with sysadmin privileges only when needed. Create separate accounts for DBAs to access SQL Server when sysadmin privileges are not needed.
5. Assign the SA account a very obscure password, and never use it to log onto SQL Server. Use a Windows Authentication account to access SQL Server as a sysadmin instead.
6. Give users the least amount of permissions they need to perform their job.
7. Use stored procedures or views to allow users to access data instead of letting them directly access tables.
8. When possible, use Windows Authentication logins instead of SQL Server logins.
9. Use strong passwords for all SQL Server login accounts.
10. Don’t grant permissions to the public database role.
11. Remove user login IDs who no longer need access to SQL Server.

12. Remove the guest user account from each user database.
13. Disable cross database ownership chaining if not required.
14. Never grant permission to the xp_cmdshell to non-sysadmins.
15. Remove sample databases from all production SQL Server instances.
16. Use Windows Global Groups, or SQL Server Roles to manage groups of users that need similar permissions.
17. Avoid creating network shares on any SQL Server.
18. Turn on login auditing so you can see who has succeeded, and failed, to login.
19. Don't use the SA account, or login IDs who are members of the Sysadmin group, as accounts used to access SQL Server from applications.
20. Ensure that your SQL Servers are behind a firewall and are not exposed directly to the Internet.
21. Remove the BUILTIN/Administrators group to prevent local server administrators from being able to access SQL Server. Before you do this on a clustered SQL Server, check Books Online for more information.
22. Run each separate SQL Server service under a different Windows domain account.

23. Only give SQL Server service accounts the minimum rights and permissions needed to run the service. In most cases, local administrator rights are not required, and domain administrator rights are never needed. SQL Server setup will automatically configure service accounts with the necessary permissions for them to run correctly, you don't have to do anything.
24. When using distributed queries, use linked servers instead of remote servers.
25. Do not browse the web from a SQL Server.
26. Instead of installing virus protection on a SQL Server, perform virus scans from a remote server during a part of the day when user activity is less.
27. Add operating system and SQL Server service packs and hot fixes soon after they are released and tested, as they often include security enhancements.
28. Encrypt all SQL Server backups with a third-party backup tool, such as SQL Backup Pro.
29. Only enable C2 auditing or Common Criteria compliance if required.
30. Consider running a SQL Server security scanner against your SQL servers to identify security holes.
31. Consider adding a certificate to your SQL Server instances and enable SSL or IPSEC for connections to clients.
32. If using SQL Server 2005, enable password policy checking.

33. If using SQL Server 2005, implement database encryption to protect confidential data.
34. If using SQL Server 2005, don't use the SQL Server Surface Area Configuration tool to unlock features you don't absolutely need.
35. If using SQL Server 2005 and you create endpoints, only grant CONNECT permissions to the logins that need access to them. Explicitly deny CONNECT permissions to endpoints that are not needed by users.

Job Maintenance

1. Avoid overlapping jobs on the same SQL Server instance. Ideally, each job should run separately at different times.
2. When creating jobs, be sure to include error trapping, log job activity, and set up alerts so you know instantly when a job fails.
3. Create a special SQL Server login account whose sole purpose is to run jobs, and assign it to all jobs.
4. If your jobs include Transact-SQL code, ensure that it is optimized to run efficiently.
5. Periodically (daily, weekly, or monthly) perform a database reorganization on all the indexes on all the tables in all your database. This will rebuild the indexes so that the data is no longer logically fragmented. Fragmented data can cause SQL Server to perform unnecessary data reads, slowing down SQL Server's performance. Reindexing tables will also update column statistics.
6. Don't reindex your tables when your database is in active production, as it can lock resources and cause your users performance problems. Reindexing should be scheduled during down times, or during light use of the databases.
7. At least every two weeks, run DBCC CHECKDB on all your databases to verify database integrity.

8. Avoid running most DBCC commands during busy times of the day. These commands are often I/O intensive and can reduce performance of the SQL Server, negatively affecting users.
9. If you rarely restart the mssqlserver service, you may find that the current SQL Server log gets very large and takes a long time to load and view. You can truncate (essentially create a new log) the current server log by running DBCC ERRORLOG. Set this up as a weekly job.
10. Script all jobs and store these scripts in a secure area so they can be used if you need to rebuild the servers.

Database Settings

1. Unless you know exactly what you are doing and have already performed impartial experiments that prove that making SQL Server configuration changes helps you in your particular environment, do not change any of the SQL Server configuration settings.
2. In almost all cases, leave the “auto create statistics” and “auto update statistics” options on for all user databases.
3. In most cases, the settings for the “maximum server memory” and the “minimum server memory” should be left to their default values. This is because the default values allow SQL Server to dynamically allocate memory in the server for the best overall optimum performance. If you use AWE memory, then this recommendation is to be ignored, and maximum memory needs to be set manually.
4. Many databases need to be shrunk periodically in order to free up disk space as older data is deleted from the database. But don’t be tempted to use the “auto shrink” database option, as it can waste SQL Server resources unnecessarily. Instead, shrink databases manually.

5. Don't rely on AUTOGROWTH to automatically manage the size of your databases. Instead, proactively monitor and alter database size as circumstances dictate. Only use AUTOGROWTH to deal with unexpected growth.

Replication

1. Replication needs should be clearly defined before creating a replication topology. Successful replication can be difficult and requires much pre-planning.
2. Ideally, publishers, distributors, and subscribers should be on separate physical hardware.
3. Create, document, and test a backup and restore strategy. Restoring replicated databases can be complex and requires much planning and practice.
4. Script the replication topology as part of your disaster recovery plan so you can easily recreate your replication topology if needed.
5. Use default replication settings, unless you can ensure that a non-default setting will actually improve replication performance or other issues. Be sure that you test all changes to ensure that they are as effective as you expect.
6. Fully understand the implications of adding or dropping articles, changing publication properties, and changing schema on published databases, before making any of these changes.

7. Periodically, validate data between publishers and subscribers.
8. Regularly monitor replication processes and jobs to ensure they are working.
9. Regularly monitor replication performance, and performance tune as necessary.
10. Add alerts to all replication jobs so you are notified of any job failures.

High Availability Best Practices

General High Availability

1. Physically protect your SQL Servers from unauthorized users.
2. Physically document all of your SQL Server instances. Incorporate effective change management.
3. Always use a RAIDed array or SAN for storing your data.
4. Use SQL Server clustering, database mirroring, or log shipping to provide extra fault tolerance.
5. Replication is not an effective means to protect your data.
6. Ensure that your entire IT infrastructure is redundant. It is only as strong as its weakest link.
7. Always use server-class hardware, and standardize on the same hardware as much as possible.
8. Use hardware and software monitoring tools so you can quickly become aware of when problems first arise.
9. After testing, apply all new service packs and hot fixes to the OS and SQL Server.
10. Cross-train staff so that there are multiple people who are able to deal with virtually any problem or issue.

Disaster Recovery

1. You must create a disaster recovery plan and include every detail you will need to rebuild your servers.
2. As your SQL Servers change over time, don't forget to update your disaster recovery plan.
3. Write the disaster recovery plan so that any computer literate person will be able to read and follow it. Do not assume a DBA will be rebuilding the servers.
4. Fully test your disaster recovery plan at least once a year.
5. Re-read all the best practice just mentioned. I'm not kidding. Remember, as DBAs, we are guardians of the organization's data. This is a huge responsibility.

Backup

1. All production databases should be set to use the full recovery model. This way, you can create transaction log backups on a periodic basis.
2. Whenever possible, perform a daily full backup of all system and user databases.
3. For all production databases, perform regular transaction log backups, at least once an hour.
4. Perform full backups during periods of low user activity in order to minimize the impact of backups on users.
5. Periodically test backups to ensure that they are good and can be restored.
6. Backup first to disk, then move to tape or some other form of backup media.
7. Store backups offsite.
8. If using SQL Server 2005 encryption, be sure to backup the service master key, database master keys, and certificates.
9. If you find that backup times take longer than your backup window, or if backup file sizes are taking up too much space on your storage device, consider a third-party backup program, such as SQL Backup Pro.
10. Document, step-by-step, the process to restore system and user databases onto the same, or a different server. You don't want to be looking this information up during an emergency.

Clustering

1. Detailed planning is critical to the success of every SQL Server cluster installation. Fully plan the install before performing the actual install.
2. An expensive cluster is of little value if the supporting infrastructure is not also fault tolerant. For example, don't forget power redundancy, network redundancy, etc.
3. Run only a single instance of SQL Server per node. Whether you have two or eight nodes in your cluster, leave one node as a failover node.
4. Cluster nodes must not be domain controllers, and all nodes must belong in the same domain and should have access to two or more domain controllers.
5. All cluster hardware must be on the Microsoft Windows Clustering Hardware Compatibility List, and certified to work together as part of a cluster.
6. Since clustering is not designed to protect data (only SQL Server instances), the shared storage device used by the cluster must incorporate fault tolerant technology. Consider log shipping or mirroring to further protect your production databases.
7. When initially installing Windows and SQL Server Clustering, be sure that all drivers and software are up-to-date, including the latest service packs or hot fixes.

8. Each node of a cluster should have identical hardware, drivers, software, and configuration settings.
9. Fiber channel shared arrays are preferred over SCSI, and Fiber channel has to be used if you include more than two nodes in your cluster.
10. The Quorum drive must be on its own fault-tolerant, dedicated, logical drive.
11. Once the cluster has been installed, test it thoroughly for every possible failure scenario.
12. Do not run antivirus or antispyware on a SQL Server cluster.
13. If you need to reconfigure any Windows or SQL Server clustering configuration options, such as IP addresses or virtual names, you will need to uninstall clustering and then reinstall it.
14. Monitor active production clusters on a daily basis, looking for any potential problems. Periodically test failover on production servers to ensure all is working well.
15. Once you have a stable SQL Server Cluster running, be very leery about making any changes to it, whatsoever.

SQL Server 2005 Mirroring

1. The principal database and the mirror database should be on separate physical hardware, and ideally, in different physical locations.
2. The witness server should be on separate physical hardware, and be on a separate network (best if at a third location).
3. Initial database mirroring setup should be done during less busy times, as the setup process can negatively affect performance of the production database being mirrored.
4. Use high availability mode whenever possible, and high performance mode only when required.
5. The hardware, along with the OS and SQL Server configuration, should be identical (at least very similar) between the two servers.
6. While a fast connection is not required between mirrored servers, the faster the connection, and the better quality the connection, the better.
7. You will want to optimize the performance of the mirrored database as much as possible to reduce the overhead caused by the mirroring process itself.
8. Thoroughly test database mirroring before putting it into production.
9. Monitor database mirroring daily to ensure that it is working properly, and is meeting performance goals.

10. Develop a formal operational and recovery procedure (and document) to support mirroring. Periodically test the failover process to ensure that it works.

Log Shipping

1. If you don't currently employ clustering or database mirroring for your SQL Servers because of cost, consider employing log shipping to help boost your high availability. It provides reasonably high availability at low cost.
2. If you take advantage of SQL Server 2000 or 2005 log shipping capability, you will want to keep the log shipping monitoring service on a SQL Server of its own, not on the source or destination servers participating in log shipping. Not only is this important for fault tolerance, but because the log shipping monitoring service incurs overhead that can affect the performance of the source and destination servers.
3. Monitor log shipping daily to ensure that it is working successfully.
4. Learn what you need to know to fix shipping if synchronization is lost between the production and backup databases.
5. Document, and test your server recovery plan, so you will be ready in case of a server failure.

Performance Tuning Best Practices

Performance Monitoring

1. Regularly monitor your SQL Servers for blocked transactions.
2. Regularly monitor system performance using System Monitor. Use System Monitor for both real-time analysis and for historical/baseline analysis.
3. If running SQL Server 2005, SP2 or later, install the free SQL Server Performance Dashboard. It can be used for real-time monitoring and performance troubleshooting.
4. Regularly monitor activity using Profiler. Be sure that traces are taken during the busiest times of the day so you get a more representative trace of what is going on in each server. When running the Profiler, do not collect more data than you need to collect.
5. Perform performance monitoring from a computer that is not the SQL Server you are monitoring. Run monitoring tools on a separate desktop or server.

Hardware Performance Tuning

1. Although heavy-duty hardware can help SQL Server's performance, application and database design can play a greater part in overall performance than hardware. Keep this in mind, as throwing good money after bad on server hardware does not always fix SQL Server performance problems. Before getting faster hardware, be sure you have thoroughly tuned your applications, Transact-SQL, and database indexing.
2. In many cases, adding RAM to a server is the cheapest and fastest way to boost hardware performance of a SQL Server. But before adding more RAM to a SQL Server, ensure first that it will be used by SQL Server. Adding more RAM doesn't mean that SQL Server will always use it. If the current Buffer Hit Cache Ratio is consistently above 99% and you have well more than 10 MB of Available RAM, your server won't benefit from adding additional RAM.
3. If your SQL Server's total CPU utilization is consistently above 80% or more, you need more CPUs, faster CPUs, or you need to find a way to reduce the load on the current server.
4. If the Physical Disk Object: % Disk Time counter exceeds 55%, and the Physical Disk Object: Avg. Disk Queue Length exceeds a count of 2 for each individual disk drive in your disk storage subsystem, then you most likely experiencing a disk I/O performance

issue and need to start looking for solutions.

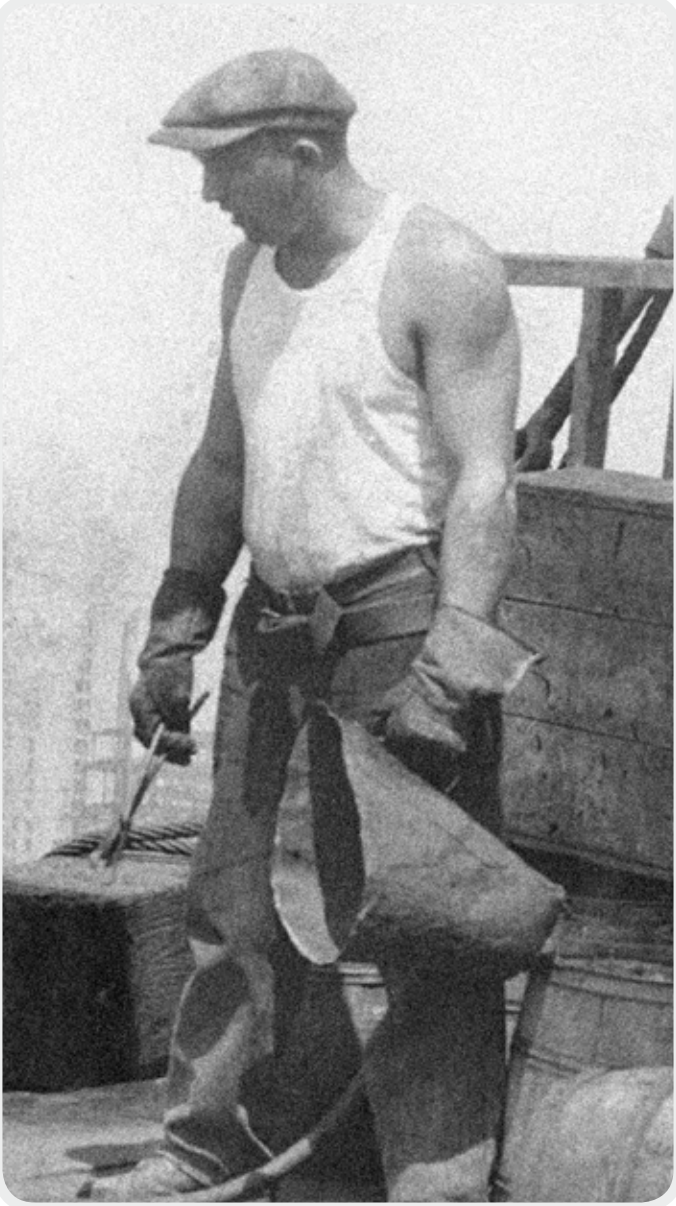
5. Don't run any applications on your server other than SQL Server, with the exception of necessary utilities.
6. NTFS-formatted partitions should not exceed 80% of their capacity. For example, if you have a 100GB logical drive, it should never be fuller than 80GB. Why? NTFS needs room to work, and when you exceed 80% capacity, NTFS become less efficient and I/O can suffer for it.
7. If your SQL Server database is mostly reads, then a RAID 5 array offers good protection and adequate performance. If your SQL Server database is mostly writes, then use a RAID 10 array for best protection and performance.
8. If your SQL Server's tempdb database is heavily used by your application(s), consider locating it on an array of its own (such as RAID 1 or RAID 10). This will allow disk I/O to be more evenly distributed, reducing disk I/O contention issues, and speeding up SQL Server's overall performance.
9. The more spindles you have in an array, the faster disk I/O will be.
10. Ensure that all hardware is running the latest, approved drivers.

Indexing

1. Periodically, run the Index Wizard or Database Engine Tuning Advisor against current Profiler traces to identify potentially missing indexes.
2. Remove indexes that are never used.
3. Don't accidentally create redundant indexes.
4. As a rule of thumb, every table should have at least a clustered index. Generally, but not always, the clustered index should be on a column that monotonically increases — such as an identity column, or some other column where the value is increasing — and is unique. In many cases, the primary key is the ideal column for a clustered index.
5. Since you can only create one clustered index per table, take extra time to carefully consider how it will be used. Consider the type of queries that will be used against the table, and make an educated guess as to which query (the most common one run against the table, perhaps) is the most critical, and if this query will benefit from having a clustered index.
6. If a column in a table is not at least 95% unique, then most likely the query optimizer will not use a non-clustered index based on that column. Because of this, you generally don't want to add non-clustered indexes to columns that aren't at least 95% unique.

7. Keep the “width” of your indexes as narrow as possible. This reduces the size of the index and reduces the number of disk I/O reads required to read the index, boosting performance.
8. If possible, avoid adding a clustered index to a GUID column (uniqueidentifier data type). GUIDs take up 16-bytes of storage, more than an Identify column, which makes the index larger, which increases I/O reads, which can hurt performance.
9. Indexes should be considered on all columns that are frequently accessed by the JOIN, WHERE, ORDER BY, GROUP BY, TOP, and DISTINCT clauses.
10. Don’t automatically add indexes on a table because it seems like the right thing to do. Only add indexes if you know that they will be used by the queries run against the table.
11. When creating indexes, try to make them unique indexes if at all possible. SQL Server can often search through a unique index faster than a non-unique index because in a unique index, each row is unique, and once the needed record is found, SQL Server doesn’t have to look any further.
12. If you perform regular joins between two or more tables in your queries, performance will be optimized if each of the joined columns have appropriate indexes.

13. Don't automatically accept the default value of 100 for the fill factor for your indexes. It may or may not best meet your needs. A high fill factor is good for seldom changed data, but highly modified data needs a lower fill factor to reduce page splitting.
14. Don't over index your OLTP tables, as every index you add increases the time it takes to perform INSERTS, UPDATES, and DELETES. There is a fine line between having the ideal number of indexes (for SELECTs) and the ideal number to minimize the overhead that occurs with indexes during data modifications.
15. If you know that your application will be performing the same query over and over on the same table, consider creating a non-clustered covering index on the table. A covering index, which is a form of a composite index, includes all of the columns referenced in SELECT, JOIN, and WHERE clauses of a query. Because of this, the index contains the data you are looking for and SQL Server doesn't have to look up the actual data in the table, reducing logical and/or physical I/O, and boosting performance.



Application Design and Coding Best Practices

Database Design

1. Bad logical database design results in bad physical database design, and generally results in poor database performance. So, if it is your responsibility to design a database from scratch, be sure you take the necessary time and effort to get the logical database design right. Once the logical design is right, then you also need to take the time to get the physical design right.
2. Normalize your data to ensure best performance.
3. Take advantage of SQL Server's built-in referential integrity. You don't need to write your own.
4. Always specify the narrowest columns you can. In addition, always choose the smallest data type you need to hold the data you need to store in a column. The narrower the column, the less amount of data SQL Server has to store, and the faster SQL Server is able to read and write data.
5. Try to avoid performing both OLTP and OLAP transactions within the same database.

Queries and Stored Procedures

1. Maintain all code in a source control system.
2. Keep transactions as short as possible. This reduces locking and increases application concurrency, which helps to boost performance.
3. Avoid using query hints unless you know exactly what you are doing, and you have verified that the hint actually boosts performance.
4. Encapsulate all transactions within stored procedures, including both the BEGIN TRANSACTION and COMMIT TRANSACTION statements in the procedure.
5. Use the least restrictive transaction isolation level possible for your user connection, instead of always using the default READ COMMITTED.
6. Whenever a client application needs to send Transact-SQL to SQL Server, send it in the form of a stored procedure instead of a script or embedded Transact-SQL. Stored procedures offer many benefits, including:

a. Reduced network traffic and latency, boosting application performance.
b. Stored procedure execution plans can be reused, staying cached in SQL Server's memory, reducing server overhead.

c. Client execution requests are more efficient. For example, if an application needs to INSERT a large binary value into an image data column not using a stored procedure, it must convert the binary value to a character string (which doubles its size), and send it to SQL Server. When SQL Server receives it, it then must convert the character value back to the binary format. This is a lot of wasted overhead. A stored procedure eliminates this issue as parameter values stay in the binary format all the way from the application to SQL Server, reducing overhead and boosting performance.
d. Stored procedures help promote code reuse. While this does not directly boost an application's performance, it can boost the productivity of developers by reducing the amount of code required, along with reducing debugging time.
e. Stored procedures can encapsulate logic. You can change stored procedure code without affecting clients (assuming you keep the parameters the same and don't remove any result sets columns). This saves developer time.
f. Stored procedures provide better security to your data

7. SET NOCOUNT ON at the beginning of each stored procedure you write. This statement should be included in every stored procedure you write.

8. Before you are done with your stored procedure code, review it for any unused code, parameters, or variables that you may have forgotten to remove while you were making changes, and remove them.
9. For best performance, all objects that are called within the same stored procedure should all be owned by the same object owner or schema, preferably dbo, and should also be referred to in the format of object_owner.object_name or schema_owner.object_name.
10. One way to help ensure that stored procedures query plans are reused from execution to execution of the same stored procedure is to ensure that any user connections information, SET options, database options, or SQL Server configuration options don't change from execution to execution of the same stored procedure. If they do change, then SQL Server may consider these same stored procedures to be different, and not be able to reuse the current query plan stored in cache.

Transact-SQL

1. Don't be afraid to make liberal use of in-line and block comments in your Transact-SQL code, they will not affect the performance of your application and they will enhance your productivity when you or others come back to the code and try to modify it.
2. If possible, avoid using SQL Server cursors. They generally use a lot of SQL Server resources and reduce the performance and scalability of your applications. If you need to perform row-by-row operations, try to find another method to perform the task.
3. When using the UNION statement, keep in mind that, by default, it performs the equivalent of a SELECT DISTINCT on the final result set. In other words, UNION takes the results of two like recordsets, combines them, and then performs a SELECT DISTINCT in order to eliminate any duplicate rows. This process occurs even if there are no duplicate records in the final recordset. If you know that there are duplicate records, and this presents a problem for your application, then by all means use the UNION statement to eliminate the duplicate rows. But if not, use UNION ALL, which is less resource intensive.

4. Carefully evaluate whether your SELECT query needs the DISTINCT clause or not. Some developers automatically add this clause to every one of their SELECT statements, even when it is not necessary.
5. In your queries, don't return column data you don't need. For example, you should not use SELECT * to return all the columns from a table if you don't need all the data from each column. In addition, using SELECT * may prevent the use of covered indexes, further potentially hurting query performance.
6. Always include a WHERE clause in your SELECT statement to narrow the number of rows returned. Only return those rows you need.
7. When you have a choice of using the IN or the BETWEEN clauses in your Transact-SQL, you will generally want to use the BETWEEN clause, as it is more efficient.
8. If you need to write a SELECT statement to retrieve data from a single table, don't SELECT the data from a view that points to multiple tables. Instead, SELECT the data from the table directly, or from a view that only contains the table you are interested in. If you SELECT the data from the multi-table view, the query will experience unnecessary overhead, and performance will be hindered.

9. If your application allows users to run queries, but you are unable in your application to easily prevent users from returning hundreds, even thousands of unnecessary rows of data, consider using the TOP operator within the SELECT statement. This way, you can limit how many rows are returned, even if the user doesn't enter any criteria to help reduce the number or rows returned to the client.
10. Try to avoid WHERE clauses that are non-sargable. If a WHERE clause is sargable, this means that it can take advantage of an index (assuming one is available) to speed completion of the query. If a WHERE clause is non-sargable, this means that the WHERE clause (or at least part of it) cannot take advantage of an index, instead performing a table/index scan, which may cause the query's performance to suffer. Non-sargable search arguments in the WHERE clause, such as "IS NULL", "<>", "!=", "!>", "!<", "NOT", "NOT EXISTS", "NOT IN", "NOT LIKE", and "LIKE '%500'" generally prevent (but not always) the query optimizer from using an index to perform a search. In addition, expressions that include a function on a column, expressions that have the same column on both sides of the operator, or comparisons against a column (not a constant), are not sargable.

SQL Server 2005 CLR

1. Don't turn on the CLR unless you will be using it.
2. Use the CLR to complement Transact-SQL code, not to replace it.
3. Standard data access, such as SELECT, INSERTs, UPDATEs, and DELETEs are best done via Transact-SQL code, not the CLR.
4. Computationally or procedurally intensive business logic can often be encapsulated as functions running in the CLR.
5. If a procedure includes both significant data access and computation and logic, consider separating the procedure code in the CLR that calls the Transact-SQL stored procedure that does most of the data access.
6. Use the CLR for error handling, as it is more robust than what Transact-SQL offers.
7. Use the CLR for string manipulation, as it is generally faster than using Transact-SQL.
8. Use the CLR when you need to take advantage of the large base class library.
9. Use the CLR when you want to access external resources, such as the file system, Event Log, a web service, or the registry.
10. Set CLR code access security to the most restrictive permissions as possible.

XML

1. XML is best used for modeling data with a flexible or rich structure.
2. Don’t use XML for conventional, structured data.
3. Store object properties as XML data in XML data types.
4. When possible, use typed XML data types over untyped XML data to boost performance.
5. Add primary and secondary indexes to XML columns to speed retrieval.

SQL Server Component Best Practices

SSIS

1. Schedule large data imports, exports, or transformation on your production servers during less busy periods of the day to reduce the impact on your users.
2. Consider running large, resource-intensive SSIS packages on a dedicated physical server.
3. Rename all default name and description properties using standard naming conventions. This will make it easier for you and others to debug or modify your packages.
4. Include annotations in your packages to make it easier for you, and others, to understand what is going on.
5. Use Sequence Containers to organize package structures into logical work units.
6. Use Namespaces for your packages.
7. Only scope variables for the containers for which they are needed.
8. If you know that data is already pre-sorted, set IsSorted=TRUE. Doing so can help prevent unnecessary sorts, which use up resources unnecessarily.

9. When selecting columns from a table to return, return only what is needed. In addition use a Transact-SQL statement in an OLE DB Source component, or the Lookup Component, instead of selecting an entire table. This way, you prevent unnecessary data from being processed.
10. When INSERTing data, use the SQL Server Destination instead of the OLE DB Destination to boost performance.

Reporting Services

1. Ideally, dedicate one or more physical servers for Reporting Services.
2. The SQL Server databases used to produce the raw data for reports should be on their own dedicated physical servers.
3. Only return the actual data you need in the report, no more or no less.
4. Optimize the performance of the Transact-SQL code you are using to return data for your report before you actually design the physical appearance of the report.
5. Manage all report code and .rdl files using a Source Control system.

Notification Services

1. Implementing Notifications Services requires careful planning and design.
2. SQL Server 2000 and 2005 Notification Services can be a heavy burden on a SQL Server, depending on how it is used. For best overall performance, it is recommended to run Notification Services to a dedicated SQL Server.
3. Notification Services makes heavy use of the tempdb database. Because of this, you should consider two things: First, consider allocating a minimum size for the tempdb database, so that when SQL Server is restarted, it will create a new tempdb database of an appropriate size. This prevents SQL Server from having to expand the tempdb database size automatically, which can lead to short bursts of poor I/O performance as the database is expanded. Second, consider installing the tempdb database on its own dedicated physical disk drive or array in order to reduce I/O contention.
4. Notification Services also makes heavy use of the transaction log. To reduce I/O contention, consider locating the transaction log file on its own dedicated physical disk drive. In addition, the Full Backup Recovery model should be used to ensure a minimum of lost data, should there be any problem. Also, because the log is heavily used, be sure that it is backed up (log truncated) often so that it does not fill up your disk space and stop your server.

5. While many indexes are created automatically by Notification Services when the Notification Services database is created, not every potentially useful index is created. Once your system is in production, consider taking a Profiler Trace during a very busy time of the server and use the Index Wizard or the Database Engine Tuning Advisor to identify potential new indexes.

Analysis Services

1. Always run OLAP applications on their own dedicated servers, never sharing a server running OLTP applications. The two types of applications are mutually exclusive when it comes to performance tuning.
2. When designing OLAP cubes, don't include measures or dimensions that your users won't use. The way to prevent this is good systems analysis and design. Unused data will increase the size of your cubes and slow performance.
3. When using the star schema design, at a minimum, you will create a non-clustered index on the primary key of each dimension table and a non-clustered index on each of the related foreign-keys. From there, you can create non-clustered indexes on additional columns that will be queried on frequently. You don't need to create composite indexes to create covering indexes because SQL Server will use index intersection to do this for you automatically.
4. When you create indexes on your data warehouses and datamarts, use a FILLFACTOR of 100 to ensure that the index pages are as full as possible. This reduces unnecessary I/O, speeding up performance.
5. Schedule cube updates on your production servers during less busy periods of the day to reduce the impact on your users.

Service Broker

1. Planning for a Service Broker implementation requires that you answer these questions:

a. Decide the role that Service Broker will play in the application.
b. Identify the required information for each step of a conversation in order to complete each task.
c. Select where the message processing logic will run.
d. Decide on the technology to be used to implement your application
e. Identify which server components will your application use the most.

2. Before rolling out a Service Broker application, test it thoroughly in a test environment that closely represents your actual production environment.
3. After a Service Broker application has been designed and written, it must be rolled out. This is generally done with a set of installation scripts that are run to create the necessary message types, contracts, queues, services, and stored procedures. Ideally, the DBA will review these scripts before running them.
4. Once the installation scripts are run, it is the job of the DBA to configure the necessary security principles, certificates, remote service bindings, and required routes.
5. Because a Service Broker installation can be complex, it must be fully documented so that it can be easily replicated and uninstalled if necessary.

Full-Text Search

1. If using SQL Server 2005, full-text catalogs can be attached and detached along with their database. This is the best way to move a database and catalog together.
2. If using SQL Server 2005, use BACKUP and RESTORE to backup and restore your full-text catalogs. Catalogs should be backed up when the database is backed up.
3. Generally speaking, limited hardware resources contribute most to poor Full-Text Search performance. Check for CPU, disk I/O, and memory bottlenecks, and if they are identified, they often must be corrected with the purchase of larger hardware.
4. Regularly defragment the index on the base tables that feed the Full-Text Search engine. In addition, regularly reorganize the full-text catalog.
5. Keep full-text key columns as narrow as possible to speed performance.

Notes