simple-talk
DATA, DEV, ADMIN, CHOP SUEY

Brad's Sure DBA Checklist

Contents

General DBA Best Practices

Best Practices for Becoming an Exceptional SQL Server DBA

1. Join (or start) a local SQL Server users group (chapters.sqlpass.org)
2. Join SQL PASS (www.sqlpass.org)
3. Attend at least one professional conference each year.
4. Attend at least one training session each year.
5. Read at least four books on SQL Server each year.
6. Read the free e-book, How to Become an Exceptional DBA
7. Learn everything you can about your job, especially those areas that nobody else likes or wants to master.
8. Volunteer at your work for new and challenging tasks that will make you known throughout your organization.
9. Install SQL Server on a laptop or desktop computer at home and practice learning new aspects of SQL Server, especially SQL Server 2008.
10. Participate in SQL Server forums (asking and answering questions)
    1. www.SQLServerCentral.com
    2. http://ask.sqlservercentral.com/com
    3. www.microsoft.com/communities/newsgroups/en-us/default.aspx
11. Get Certified as a SQL Server DBA:
    1. a. Microsoft Certified Technology Specialist (http://www.microsoft.com/learning/en/us/certification/cert-sql-server.aspx#tab2)
    2. b. Microsoft Certified IT Professional (http://www.microsoft.com/learning/en/us/certification/cert-sql-server.aspx#tab3)

Day-to-Day

1. Check OS Event Logs and SQL Server 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. For best performance, all disks should have 15% or more of free space.
5. Throughout the day, periodically monitor performance using both System Monitor and Profiler/SQL Trace.
6. Regularly 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 e-mailed to you. Take action as needed.
9. Regularly restore backups to a test server in order to verify that you can restore them. You don't need to restore all backups every day, but do so often to ensure that you are confident you have good backups.
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 on instances that don't use them, such as Microsoft Full-Text Indexing, Reporting 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, along with no other applications.
6.
7. For best I/O performance, locate the database files (.mdf) and log files (.ldf) on separate spindles to isolate disk access patterns.
8. If tempdb will be used heavily, put it on its own separate spindles.
9. Do not install SQL Server on a domain controller.
10. Be sure that SQL Server is installed on an NTFS partition.
11. Don't use NTFS data file encryption (EFS) and compression on SQL Server database and log files**.**

Upgrading

1. Run the SQL Server 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. 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 from physically access 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. Instead, use a Windows Authentication account to access SQL Server as a sysadmin.
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. Disable the guest user account from each user database by using REVOKE CONNECT FROM GUEST.
13. Don't use 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. Configure 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. In SQL Server 2005 and earlier, remove the BUILTIN/Administrators group to prevent local server administrators from being able to access SQL Server. In SQL Server 2008, the BUILTIN/Administrators group does not exist by default.
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.

24. When using distributed queries, use linked servers instead of remote servers. Remote servers only exist for backward compatibility.
25. Do not browse the web from a production SQL Server instance.
26. Instead of installing virus/antispyware protection on a SQL Server, perform 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 Red Gate SQL Backup Pro.
29. Only enable C2 auditing or Common Criteria compliance if required, as they add significant performance overhead.
30. Consider running a SQL Server security scanner against your SQL servers to identify security holes.
31. Consider enabling SSL or IPSEC for connections between SQL Server and its clients.
32. If using SQL Server 2005/2008, enable password policy checking.
33. If using SQL Server 2008 in a high security environment, consider implementing Transparent Data 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 2005/2008 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), rebuild or reorganize the indexes of your databases to remove logical fragmentation and wasted space.
6. Periodically, as part of your scheduled database maintenance tasks, run DBCC CHECKDB on all your databases to verify database integrity.
7. 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.
8. If you rarely restart the SQL Server service, you may find that the current SQL Server log gets very large and takes a long time to load and view. You can close the current error log and create a new one by running sp_cycle_errorlog or DBCC ERRORLOG, so that the log doesn't get overly large. Set this up as a weekly job.
9. Script all jobs and store these scripts in a secure area so they can be used if you need to rebuild the servers.

SQL Server Configuration Settings

1. SQL Server configuration settings should remain at their default settings. Any changes to these settings should only be made by an experienced DBA who understands the pros and cons of making changes.
2. If you are using the 32-bit version of SQL Server, and if you are using 4 GB or more of RAM, ensure that you have all the AWE settings correctly set.
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. Exceptions to this rule of thumb can come into play if you use 32-bit AWE memory, or 64-bit memory, where you may have to change the "maximum server memory" to an amount that is less than the physical amount of RAM in your server.

Database Settings

1. Leave the "auto create statistics" and "auto update statistics" options on for all user databases. Only in very rare cases should these be turned off, and if they are turned off, then you must manually update the statistics yourself.

2. Don't be tempted to use the "auto shrink" database option, as it can waste SQL Server resources unnecessarily and contribute to index fragmentation. Instead, if you need to shrink a database, do so manually.
3. 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 RAID-enabled direct attached storage 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 from a high availability standpoint.
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 thorough 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. Review all the best practices you know and implement them as appropriate. Remember, as DBAs, we are guardians of the organization's data. This is a huge responsibility.

Backup

1. All OLTP 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 OLTP 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 perform test backup restores to ensure that your backups are good and can be restored.
6. Encrypt your backups in case they should become "lost."
7. Store backups offsite and in a secure location.
8. If using SQL Server encryption, be sure to back up the appropriate service master keys, 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 Red Gate 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 at least 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, database mirroring, or SAN replication to further protect your production databases.
7. Before installing Windows Clustering 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 when building a cluster, and Fiber channel has to be used if you include more than two nodes in your cluster.
10. In conventional clusters, the quorum drive must be on its own fault-tolerant, dedicated, disk of 500MB or larger
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. Before you begin to install Cluster Services or SQL Server clustering, determine your virtual names and gather your IP addresses ahead of time. This makes it easier when it is time to enter this information when installing and configuring your cluster.
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 2005/2008 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. For ease of administration, 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 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 is working successfully.
4. Learn what you need to know to fix log 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 system performance using System Monitor or similar tool.
2. Regularly monitor system performance using Performance Monitor. Use Performance 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. If you are using SQL Server 2008, consider using the Performance Data Collector.
4. Regularly monitor SQL Server activity using SQL Trace or Profiler. Be sure that traces are taken during times of the day that are representative of the typical activity that is going on in each server. When running the SQL Trace or Profiler, do not collect more data than you need to collect, in order to reduce the overhead of performing the trace.
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 100 MB of Available RAM, your server may not 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 Logical Disk: % Idle Time counter is less than 50%; and the Logical Disk: Avg. Disk Sec/Read counter or the Logical Disk: Avg. Disk Sec/Write counter is more than 15ms, then you most likely are 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 85% of their capacity. For example, if you have a 100GB drive, it should never be fuller than 85GB. Why? NTFS needs room to work, and when you exceed 85% capacity, NTFS becomes less efficient and I/O can suffer for it. In addition, disk file fragmentation utilities need at least 15% of free disk space to work properly.
7. If your SQL Server database experiences 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, consider locating it on an array of its own spindles (such as RAID 1 or RAID 10). This will allow disk I/O to be more evenly distributed for the SQL Server instance, 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. Be sure the disk controllers can handle the extra I/O provided by adding more spindles.
10. Ensure that all hardware is running the latest, approved drivers.

Indexing

1. Periodically, run the 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 most from having a clustered index.
6. When creating a composite index, and when all other considerations are equal, make the most selective column the first column of the index.
7. Generally speaking, 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. 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 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 I/O, and boosting performance.

SQL Server 2008 Compression Best Practices

1. SQL Server 2008, Enterprise Edtition, offers both row and column compression, which can be used to reduce the amount of space data takes up on disk and in the buffer pool, potentially boosting performance.
2. Compression requires the use of additional CPU cycles to implement. Because of this, if your server currently has a CPU bottleneck, then compression should probably be avoided.
3. Compression works best for tables that experience mostly reads. Tables that experience a large number of DML activities should probably not be compressed.
4. Don't use compression on tables subject to heavy inserts, such as bulk inserts.

SQL Server 2008 Data Collector Best Practices

1. The SQL Server 2008 Data Collector can be used to gather, store, and analyze many different types of SQL Server performance activities.
2. Using the SQL Server 2008 Data Collector imposes some performance overhead, and because of this, you may want to minimize the amount of data that is collected by it.
3. While the Management Data Warehouse can exist on each instance of SQL Server, it is recommended that a dedicated SQL Server be used to store the Management Data Warehouse used to store the collected data.

Best Practices for Resource Governor

1. The SQL Server 2008 Resource Governor can be used to restrict how much CPU and memory resources are allocated to a particular user connection, allowing the DBA to give resource preference to some connections over other connections.
2. If you use the Resource Governor, ensure that the DAC is turned on, as it is the only way to troubleshoot and fix a bad Resource Governor configuration.
3. Classifier user-defined function should be well tested and optimized. If written incorrectly, a classifier function can cause connections to slow down, time out, and even create an infinite loop that locks up SQL Server.
4. Monitor Resource Governor after implementation to confirm appropriate use. If you are not careful, you can introduce query performance problems that didn't exist before using Resource Governor. For example, if memory is limited too much, it can cause a suboptimal execution plan to be created and executed, hurting performance.
5. Test your Resource Governor implementation on test system before implementing it on production

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. When designing a new database, implement consistent naming standards for all of your objects.
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. When designing a new database, fully document every table, column, default, check constraint, and relationship, so others can easily find out what they mean, or how they are to be used.

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 concurrently, 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. Of course, don't do anything that could compromise the integrity of your data.
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. This is also mostly true for Transact-SQL code sent to SQL Server outside of a stored procedure.
    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 for your data.

7. SET NOCOUNT ON at the beginning of each stored procedure you write.
8. Before rolling a stored procedure out into production, review it for any unused code, parameters, or variables that you may have forgotten to remove while you were creating it, and remove them.

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.
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 every column. In addition, using SELECT * may prevent the use of covering 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. 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.
8. 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 prevents (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.
9. If you are using the SQL Server 2008 debugger, only run it against a test database, never a production database.

SQL Server CLR

1. Use the CLR to complement Transact-SQL code, not to replace it.
2. Standard data access, such as SELECT, INSERTs, UPDATEs, and DELETEs are best done via Transact-SQL code, not the CLR.
3. Computationally or procedurally intensive business logic can often be encapsulated as functions running in the CLR.
4. Use the CLR for error handling, as it is more robust than what Transact-SQL offers.
5. Use the CLR for string manipulation, as it is generally faster than using Transact-SQL.
6. Use the CLR when you need to take advantage of the large base class library.
7. Use the CLR when you want to access external resources, such as the file system, Event Log, a web service, or the registry.
8. 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 Transact-SQL statements 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.

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.
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 ran 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 principals, 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 reinstalled if necessary.