SQL Server – Best Practices

DOCUMENT HISTORY

Version	Date	Author	Comments
0.01	25 Aug 2010	K, Madhusudanan Nair	Initial Draft
		Hemant	Clustering, Log shipping and Mirroring BP

1 INTRODUCTION

The objective of this document is to provide an exhaustive list of best practices that can be leveraged for programming with SQL Server 2000/2005/2008.

2 SCOPE

This document covers the SQL Server best practices in the following areas

1. T-SQL
2. Stored procedures
3. XML Usage
4. CLR Integration
5. Cursors
6. Views
7. Datatypes
8. General guidelines
9. Data load best practices
10. Security best practices
11. Storage best practices
12. Clustering best practices
13. Mirroring best practices
14. Log shipping best practices

3 INTENDED AUDIENCE

This document is intended for all SQL Server database developers and designers.

4 T-SQL GUIDELINES

4.1. General SQL Best Practices

4.1.1 Use Stored procedures:

The Use of stored procedures instead of ad-hoc queries results in better performance, manageability and security. Some advantages in using stored procedures are mentioned below-

- Stored procedures reduce network traffic and latency, thereby boosting application performance.

- They cause re-use of execution plans in most cases (Execution plan creation is a CPU intensive operation and can hinder performance).

- Client execution requests are more efficient when a stored procedure is used.

  For example, if an application needs to INSERT a large binary value into an image data column, without 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 overhead in terms of network traffic.

  A stored procedure eliminates this issue as parameter values remain in the binary format all the way from the application to SQL Server, thereby reducing overhead and improving performance drastically.

- Stored procedures help in promoting 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.

- Stored procedures can encapsulate logic thereby helping in reducing the overhead in releasing new builds for every application logic change, if the logic were to be present in the front-end code. The stored procedure code can be changed without affecting clients (assuming you keep the parameters the same and don’t remove any result sets columns). This saves developer time.

- Stored procedures provide better flexibility in terms of security. Execution permission on the stored procedure alone can be given to a user or group of users who are denied permission to access the underlying objects (tables, views, etc) that the stored procedure accesses.

- Stored procedure helps to modularize the code.

4.1.2 Use sargable WHERE clause:

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 on that column 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 non-sargable.

4.1.3 Use Fully Qualified Name for objects:

It is recommended to use a fully qualified name for the objects like <schema name>.<object name> rather than just <object name>. When the execution plan is prepared by the query engine, in the binding process, the query engine has to resolve the existence of the object. If the fully qualified object name is specified, the object resolution become easy for the engine and the code will also be more readable. This will avoid contention on the system objects since the engine doesn’t have to refer them to know the scope.

4.1.4 DO NOT use undocumented system objects:

Usage of undocumented system objects can break the code if Microsoft has decided to remove that system object in future release. Microsoft will not publish the removal of undocumented system objects as it does for the documented system objects.

4.1.5 DO NOT reuse variable

Do not use a common variable for multiple logical blocks in the same stored procedure/function. If we miss to reset the variable before the next logical block, the variable will have the last value assigned and this can cause logical error.

This is a general programming best-practice.

4.1.6 Remove redundant or unwanted code

It is very common that during development, as part of debugging, lots of “print” and “select” statements are added to the code to track the results step-by-step. But because of developer’s oversight, these may not be removed while going live.

Ensure that all the debugging code is removed before promoting to production. This is a general programming best-practice.

4.1.7 Use built-in key constraints

Do NOT write trigger to ensure referential integrity. Instead use the foreign key constraint that is already being provided by SQL Server.

4.1.8 Avoid using Scalar Function in SELECT statement

Scalar functions are to be avoided in queries involving large result sets. It is because scalar functions behave like cursors in such occasions causing performance degradation. Change the scalar function to inline or multi-statement table-valued functions or a view.

4.1.9 Avoid mixing-up DML and DDL statements on a temp table in stored procedure

When a temporary table (# table) is created in a stored procedure and later altered down the line, this can cause the stored procedure to get re-compiled every time it is called.

4.1.10 Use SET NOCOUNT ON

This option is useful in stored procedures having many statements. When “NOCOUNT” is set to “ON”, the message on the number of rows affected for every statement is not returned back to the caller, thus reducing network traffic and increasing performance.

4.1.11 Avoid or minimize the usage of HINTS

Proving hints in a query forces the SQL Server query engine to do as instructed. While the hints provided by the developer may be beneficial in a certain circumstance, it might not be the best pick in all scenarios when the data topography changes.

Query access path determination is best left to the SQL Server query optimizer.

Reference: http://msdn.microsoft.com/en-us/library/ms187713.aspx

4.1.12 Do not change SET OPTION In connection

Changing SET Option during connection or anywhere else will cause the stored procedure to recompile.

Refer this KB for more info http://www.microsoft.com/technet/prodtechnol/sql/2005/recomp.mspx

4.1.13 Keep the transaction as short as possible

When defining an explicit transaction (using BEGIN TRAN…END TRAN), care should be taken to make it as short as possible. This will help avoid deadlock scenario where two long running transactions wait for each other to release a resource the other one wants.

One other way to avoid deadlock is to ensure that concerned transactions access the objects in the same sequence, for example, Table1 first, Table 2 next and Table 3 lastly. This will also reduce the probability of a deadlock.

4.1.14 Avoid user input inside a transaction

Do not accept a user input inside a transaction.

4.1.15 Avoid doing Front-End work in Databases

Front-end programming languages are better in processing characters, be it sorting a list of strings or string conversion compared to databases. Hence, in such cases, it is better to fetch the data out and do the manipulation in the front-end as much as possible.

There needs to be a clear segregation between Data Layer, Data Access Layer and Business Layer.

4.1.16 Avoid Function in select statement

Any functions like CONVERT, CAST, ISNULL usage in a query's WHERE clause may ignore the indexes available on the table. Hence, they are better avoided by adopting a proper data model.

4.1.17 Dynamic SQL

Dynamic SQL is to be avoided as much as possible. In cases where they are the only option, sp_executesql system stored procedure should be used to execute them and not the "EXECUTE" statement. The reason is that "EXECUTE" statement is prone to SQL injection and is not a parameterized query which can re-use execution plan.

4.1.18 Use proper size for the input parameters

Using only the required size for the input parameters will help reduce memory usage and also to avoid SQL Injection.

4.1.19 Avoid or minimize the usage of negative operators

The usage of negative operators like NOT EQUAL TO (<>), NOT IN, NOT EXISTS should be avoided because they causes a table scan. Query engine has to ensure there is not data till the last row is read.

Wherever possible, replace this with positive conditions.

4.1.20 Avoid or minimize cursors/loops

Since RDBMS is a set-based product, set operations work very well. In most cases, looping structures like CURSOR and WHILE loop can be replaced with set-based operations like join, sub-query, derived tables, common table expression, etc.

4.1.21 Do not use WITH RECOMPILE

Usage of WITH RECOMPILE causes the procedure to recompile each time it is called. Hence, this needs to be avoided.

4.1.22 JOIN Consideration

While implementing joins, negative operators (<>, NOT IN) and LIKE operator are to be avoided.

4.1.23 Avoid GOTO

- This is a time-proven means of adding disorder to program flow.
- There are some cases where intelligent use of GOTO is preferable to dogmatically refusing to use it.
- On the other hand, unintelligent use of GOTO is a quick ticket to unreadable code.

4.1.24 Assign procedural parameters to local variables

- Assign input parameters to local variables and use local variables in the where clause.
- This allows the database engine to generate a generically valid execution plan by avoiding parameter sniffing

4.1.25 Use OUTPUT parameters instead of result set wherever possible

In cases of stored procedures that return a single row as result set, using OUTPUT clauses for each of the columns will be beneficial as these are handled faster by connection providers like ADO.

4.1.26 Use a semicolon (;) to suffix all T-SQL statements

- The use of a semicolon is ANSI-compliant and is mandatory with other database platforms.
- Although it's not mandatory in SQL Server 2000, it standardizes the code and also make the code portable ready to any other database product.

4.1.27 Retrieve only the data that is required

- A SELECT statement should retrieve only data that is necessary. Use WHERE clause to filter unwanted rows.
- Columns that are not used or already known should not be returned.
- SELECT * is not a good coding practice. The reason is very logical, more data, more memory, more network and other resource.

4.1.28 Use a column list in your INSERT statements

This helps in avoiding problems when the table structure changes (like adding or dropping a column).

4.1.29 Avoid temporary tables if possible

- Temp table involves logging which is IO intensive especially if you have Temp DB I/O contention.
- Temp table can cause the stored procedure to recompile.
- Temp table should be preceded with a IF EXISTS…DROP statement.
- In case of temp table being created in a SP which may be called in multi thread, do not use SELECT INTO to create your temp table, as it places locks on system objects (syscolumns, sysindexes, syscomments). Instead, create the table, and then use INSERT INTO to populate the table.
- Do not alter temp table after creation in the same stored procedure. This can cause the SP to recompile.

4.1.30 Alternatives for Temp table

- TABLE variable data type which can provide in-memory solutions for small tables inside stored procedures too.
- CTE can replace many temp table requirement
- If possible try correlated sub query or derived table.

4.1.31 Declare Character datatype variable with proper size

When you do not specify the length of your string objects in SQL Server, it applies its own defaults. For most things, the default length is one character.

4.1.32 Avoid Wildcard Search

- Avoid wildcards in search as much as possible.
- If not, try to avoid wildcard characters at the beginning of a word while searching using the LIKE keyword

Example:
Use this
```
SELECT LocationID FROM Locations WHERE Specialities LIKE 'A%s'
```

Instead of this
```
SELECT LocationID FROM Locations WHERE Specialities LIKE '%pples'
```

4.1.33 Use 'Derived tables' wherever possible

Example:
```
--Use this
SELECT MIN(salary)
FROM   (SELECT TOP 2 salary
       FROM   employees
       ORDER  BY salary DESC) AS a

--Instead of this
SELECT MIN(salary)
FROM   employees
WHERE  empid IN (SELECT TOP 2 empid
                 FROM   employees
                 ORDER  BY salary DESC)
```

4.1.34 Do not use column numbers in the ORDER BY clause

Consider the following example in which the first query is more readable than the second one:

Example:
```
--Correct Usage
SELECT orderid,
       orderdate
FROM   orders
ORDER  BY orderdate

--Incrrect usage
SELECT orderid,
       orderdate
FROM   orders
ORDER  BY 2
```

4.1.35 Use SCOPE_IDENTITY

Don’t do “SELECT MAX(ID) from Master Table” or @@Identity function when inserting in a Details table. This is a common mistake, and will fail when concurrent users are inserting data at the same instance.

Use one of SCOPE_IDENTITY or IDENT_CURRENT. SCOPE_IDENTITY would give you the identity value from the current context in perspective.

4.1.36 Avoid assumptions in Date Usage

- Prevent issues with the interpretation of centuries in dates.
- Do not specify years using two digits.
- Assuming dates formats is the first place to break an application. Hence avoid making this assumption.

4.1.37 Getdate() vs GetUTCDate()

- In scenarios where the business has applications that span over multiple time zones/regions, consider storing the date using getutcdate() instead of getdate().
- Avoid daylight-saving updates to the dates stored. Instead, use a separate function which will manipulate the dates stored to the user’s timezone, also incorporating the day-light saving difference
- Avoid passing dates from the front-end.
- In cases where dates are passed from the front-end to the DB, make sure that the date passed does not incorporate the daylight saving adjustment.
- SQL Server 2008 and later version supports new datatype like datetimeoffset Defines a date that is combined with a time of a day that has time zone awareness and is based on a 24-hour clock.

4.1.38 Use UNION ALL over UNION wherever possible

- Use UNION ALL over UNION when the SELECT statement does not produce duplicate rows. UNION does an additional distinct operation over the accumulated result set which can be avoided in certain cases beforehand
- A UNION (without the ALL) performs an internal sorting to eliminate duplicate rows, thus creating a worktable hindering the performance

4.1.39 Use UNION ALL/UNION instead of OR if a query is performing poorly

- The query plan may be reused for the second query
- Union of multiple select statements leads to better use of indexes in complex joins
- Union/Union All scores over OR when the OR condition is used for filtering over multiple columns.

Example:
Usage of UNION

```
SELECT resourceid,
       resourcename,
       resourcedod
FROM   dbo.resourcemaster
WHERE  projectcategory = 'LEGACY'
UNION ALL
SELECT resourceid,
       resourcename,
       resourcedod
FROM   dbo.resourcemaster
```

```
WHERE  billingcategory = 'OLD'
UNION ALL
SELECT resourceid,
       resourcename,
       resourcedod
FROM   dbo.resourcemaster
WHERE  resourcetype = 'ARCHITECT'

--Usage of OR
SELECT resourceid,
       resourcename,
       resourcedod
FROM   dbo.resourcemaster
WHERE  projectcategory = 'LEGACY'
        OR billingcategory = 'OLD'
        OR resourcetype = 'ARCHITECT'
```

4.1.40 Use SQLCLR stored procedures, functions, triggers with care

- Use SQLCLR stored procedures only for computation intensive algorithms
- Avoid calling a SQLCLR function for computations that can be performed in T-SQL
- Avoid using SQLCLR for set based computation. T-SQL is built for that and performs much faster
- Try to avoid SQLCLR User defined aggregates if possible. It uses and disposes lots of objects in the aggregation process and might hit the performance during high concurrency

4.1.41 SET vs. SELECT

SET is the ANSI standard way of assigning values to variables and SELECT is not.
SET is a more structured way of assigning values. Use it unless necessary otherwise.

4.1.42 Common Table Expressions vs CROSS APPLY in hierarchy generation

- It is best to go for a Common Table Expressions if the joining columns of the self join are indexed.
- If index cannot be created to the joining columns then its better to use CROSS APPLY. CROSS APPLY does not use indexes. So it’s best to avoid this operator when indexes can be utilized in the query. CROSS APPLY is best suited if the number of rows returned is less and iteration depth <5

4.1.43 Minimize the number of tables in a Join

Minimize the number of tables in a join where possible and qualify joins as much as possible, this may eliminate rows in certain table before the join takes place

4.1.44 Ensure columns with similar data type are used when using JOINS

Join columns that have identical data types whenever possible. Performance suffers when non-identical columns are compared; SQL Server needs an implicit conversion between the two types. Performance is better when the tables being joined are defined with identical types. This prevents implicit data conversions by the query engine.

4.1.45 Use in-line table valued function instead of using redundant queries

In-line table valued function (parameterized query) helps in code reuse and maintainability and better abstraction of common functionality. Does not affect the query execution plan of the calling procedure (only inline table valued functions).The table valued function can be used as a table in joins.

4.1.46 Use TRUNCATE TABLE instead of DELETE

Just like DROP TABLE, TRUNCATE TABLE is a metadata-only operation. Be aware that certain limitations apply to TRUNCATE TABLE. These are:

- You cannot truncate a table that is referenced by foreign key constraints.
- You can truncate the entire table only, not a single partition (but see "Deleting All Rows from a Partition or Table").

4.1.47 Table Types and Table-Valued Parameters

SQL Server 2008 introduces table types and table-valued parameters that help abbreviate your code and improve its performance. Table types allow easy reuse of table definition by table variables, and table-valued parameters enable you to pass a parameter of a table type to stored procedures and functions.

4.1.48 Table-Valued Parameters

You can now use table types as the types for input parameters of stored procedures and functions. Currently, table-valued parameters are read only, and you must define them as such by using the READONLY keyword.

A common scenario where table-valued parameters are very useful is passing an "array" of keys to a stored procedure. Before SQL Server 2008, common ways to meet this need were based on dynamic SQL, a split function, XML, and other techniques. The approach using dynamic SQL involved the risk of SQL Injection and did not provide efficient reuse of execution plans. Using a split function was complicated, and using XML was complicated and nonrelational. (For details about this scenario, see "Arrays and Lists in SQL Server" by SQL Server MVP Erland Sommarskog at http://www.sommarskog.se/arrays-in-sql.html.)

4.1.49 Table Value Constructor Support through the VALUES Clause

SQL Server 2008 introduces support for table value constructors through the VALUES clause. You can now use a single VALUES clause to construct a set of rows. One use of this feature is to insert multiple rows based on values in a single INSERT statement

Eg.

```
 INSERT INTO dbo.Customers(custid, companyname, phone, address)
  VALUES
  (1, 'cust 1', '(111) 111-1111', 'address 1'),
  (2, 'cust 2', '(222) 222-2222', 'address 2'),
  (3, 'cust 3', '(333) 333-3333', 'address 3'),
  (4, 'cust 4', '(444) 444-4444', 'address 4'),
  (5, 'cust 5', '(555) 555-5555', 'address 5');
```

4.1.50 If possible Use MERGE Statement

The new MERGE statement is a standard statement that combines INSERT, UPDATE, and DELETE actions as a single atomic operation based on conditional logic. Besides being performed as an atomic operation, the MERGE statement is more efficient than applying those actions individually.

4.1.51 If application needs to store images try to use FILESTREAM feature

In SQL Server, BLOBs can be standard varbinary(max) data that stores the data in tables, or FILESTREAM varbinary(max) objects that store the data in the file system. The size and use of the data determines whether you should use database storage or file system storage. If the following conditions are true, you should consider using FILESTREAM:

- Objects that are being stored are, on average, larger than 1 MB.

- Fast read access is important.

- You are developing applications that use a middle tier for application logic.

For smaller objects, storing varbinary(max) BLOBs in the database often provides better streaming performance.

.

4.2. Stored Procedures

4.2.1 Use Stored Procedures (SPs) wherever you can and comment your SPs

- Execution plans may be cached for improved performance
- Separate the business logic from presentation logic
- Limit multiple network calls by encapsulating complex business logic
- SP also helps to modularize the code, increase the manageability and security

4.2.2 DO NOT start the name of a stored procedure with 'SP_'

This is because all the system related stored procedures follow this convention. SP_ name make the optimizer to check the existence of SP in Master db first hence the context switch. This can cause a performance issue in high end application. There is also a chance to clash names if Microsoft publishes a system stored procedure with the same name in future release.

4.2.3 Keep SPs small in size and scope

Two users invoking the same stored procedure simultaneously will cause the procedure to create two query plans in cache. It is much more efficient to have a stored procedure call other ones then to have one large procedure.

4.2.4 Return only the stored procedure completion status

Standardize the return values of stored procedures for success and failures. The RETURN statement is meant for returning the execution status only, but not data. If you need to return data, use OUTPUT parameters.

4.2.5 Use TRY… CATCH block. Avoid capturing @@ERROR

- TRY CATCH block is a better and a more readable way of error management
- To catch all the errors which are of severity greater than 10 and which do not close the database connection
- Enclose all Data Manipulation queries in TRY… CATCH BLOCK
- Enclose all transactions within a TRY… CATCH BLOCK. This will also help in trapping deadlocks and handling it at the server rather than throwing to the client.
- Enclose batches calling external stored procedures or CLR routines in TRY… CATCH block

4.3. Views

Views are generally used to show specific data to specific users based on their interest. Views are also used to restrict access to the base tables by granting permission on only the views. Yet another significant usage of views is that they simplify your queries. Incorporate your frequently required complicated joins and calculations into a view, so that you don't have to repeat those joins/calculations in all your queries, instead just select from the view.
Do not use indexed views unless you have one of the following scenarios.

- Joins and aggregations of large tables
- Repeated patterns of queries
- Repeated aggregations on the same or overlapping sets of columns
- Repeated joins of the same tables on the same keys
- Combinations of the above
- The query takes more than a few seconds to run. The index in the view is ignored unless SQL Server anticipates a real performance hit

4.4. Cursors

4.4.1 Use Cursors wisely

Cursors force the database engine to repeatedly fetch rows, negotiate blocking, manage locks and transmit results. They consume network bandwidth as the results are transmitted back to the client, where they consume RAM, disk space, and screen real estate. Consider the resources consumed by each cursor you build and multiply this demand by the number of simultaneous user

4.4.2 Open late and release soon

Open your cursor as late as possible and close and deallocate your cursor as soon as possible.

4.4.3 Use the right cursor type

- Don't use a scrollable cursor if you don't plan to let the user scroll through the rows.
- Use read-only cursors unless you need to update tables. In most cases, you should only update through use of stored procedures.
- The Forward-only, read-only, CacheSize (RowsetSize) "firehose" cursor is the fastest way to get data from the server—because these settings don't create a cursor at all.

4.4.4 Use @@FETCH_STATUS

Use @@FETCH_STATUS to determine the results of a cursor fetch

4.4.5 Avoid Cursors in a Transaction

Avoid using cursors inside transactions. The records retrieved in cursor are locked. Since cursors are more time consuming, lock time of data might increase resulting into a performance hit.

4.5. Data Types

Following are some key aspects to be kept in mind when choosing the data types. Use may SQL Server 'User Defined Data types' provided, if a particular column repeats in a lot of your tables, so that the datatype of that column is consistent across all your tables. Also try to leverage the new datatypes supported by the product's current version. For eg. If the current version is SQL Server 2008 , there are new DATE datatypes which can optimize the storage and coding

1. Try not to use text, ntext data types for storing large textual data. In SQL Server 2005 and later versions have VARCHAR(MAX), VARBINARY(MAX) kind of datatypes which can be used in place of these datatypes.

   Text datatype has some inherent problems associated with it. You can not directly write, update text data using INSERT, UPDATE statements (You have to use special statements like READTEXT, WRITETEXT and UPDATETEXT). There are a lot of bugs associated with replicating tables containing text columns. So, if you don't have to store more than 8 KB of text, use char(8000) or varchar(8000) data types.

2. Implicit conversions are invisible to the user and they have some performance overhead.
   a. SQL Server automatically converts the data from one data type to another. For example, if a smallint is compared to an int, the smallint is implicitly converted to int before the comparison proceeds.
   b. Explicit conversions use the CAST or CONVERT functions

3. Use char data type for a column, only when the column is non-nullable. If a char column is nullable, it is treated as a fixed length column in SQL Server 7.0+. So, a char(100), when NULL, will eat up 100 bytes, resulting in space wastage. So, use varchar(100) in this situation. Of course, variable length columns do have a very little processing overhead over fixed length columns. Carefully choose between char and varchar depending up on the length of the data you are going to store.

4. Minimize the usage of NULLs, as they often confuse the front-end applications, unless the applications are coded intelligently to eliminate NULLs or convert the NULLs into some other form. Any expression that deals with NULL results in a NULL output. ISNULL and COALESCE functions are helpful in dealing with NULL values.

5. Use Unicode data types like nchar, nvarchar, ntext, if your database is going to store not just plain English characters, but a variety of characters used all over the world. Use these data types, only when they are absolutely needed as they need twice as much space as non-unicode data types.

6. Always store 4 digit years in dates (especially, when using char or int datatype columns), instead of 2 digit years to avoid any confusion and problems. This is not a problem with datetime columns, as the century is stored even if you specify a 2 digit year. But it's always a good practice to specify 4 digit years even with datetime datatype columns.

7. If you do a proper analysis of the data and then select the datatype, then you can control the row, page, table size and hence increase the overall performance of the database. Following points you may consider when you design a table :-

   a. If your database is to support web-based application better to go for UNICODE for the scalability of the application. (Unicode (nchar, nvarchar) takes 2 bytes per char where as ASCII (char,varchar) datatypes takes 1 bytes per char)
   b. If your application is multi-lingual go for UNICODE.

   c. If you are planning to include CLRDatatype (SQL Server 2005 or later version) in the database go for UNICODE Datatypes , because, if CLRDatatype is going to consume the data then it must be in UNICODE.

   d. For numeric column, find the range that column is going to have and then choose the datatype. For eg. You have a table called Department and DepartmentID is the Primary key Column. You know that the maximum number of rows in this table is 20-30. In such cases it is recommended to choose TinyINT datatype for this column. Generally keeping all numeric columns type of INT without analyzing the range that column going to support is not at all recommended from storage perspective.

e. Description /Comments /Remarks sort of columns may or may not have data for all the rows. So it is better to go for Variable datatypes like Varchar, Nvarchar.

f. If you know the column is not nullable and it may contain more or less the same size of the data then for sure go for Fixed datatype like CHAR or NCHAR. Having said that it is important to know that, if you select fixed datatypes and if the column is nullable then, if you do not have any data (null) then also the column will consume the space.

g. If the size of the column is less than 20 char , use fixed width datatypes like NCHAR or CHAR.

h. I have seen in many applications use Decimal to store currency kind of data though the application needs the precision which can be supported by money. So, my point here is, use Money datatype if you need only 4 precision.

i. Use UniqueIdentitifier column as PK and Clustered Index or so only when it is unavoidable because UniqueIdentitifier takes 16 Bytes of the space.

8. Do not use the decimal (or numeric) data type when scale is 0. If you do not want to store fractional numbers, then you should use a different data type, like bigint, int, smallint, or tinyint.
9. Do not use deprecated datatypes even though it meets the current requirement.

4.6. Error Handling

Use the new structured exception handling feature of SQL Server 2005 and later version as much as possible.

Be aware that inside a TRY…CATCH construct, transactions can enter a state in which the transaction remains open but cannot be committed. The transaction cannot perform any action that would generate a write to the transaction log, such as modifying data or attempting to roll back to a savepoint.

In this state, however, the locks acquired by the transaction are maintained, and the connection is also kept open. The transaction's work is not reversed until a ROLLBACK statement is issued. The code in a CATCH block should test for the state of a transaction by using the XACT_STATE function. XACT_STATE returns a -1 if the session has an uncommittable transaction. The CATCH block must not perform any actions that would generate writes to the log if XACT_STATE returns a -1.

4.7. Indexes

4.7.1 Know the impact of using indexes

Indexes are vital to efficient data access; however, there is a cost associated with creating and maintaining an index structure. For every insert, update and delete, each index must be updated. In a data warehouse, this is acceptable, but in a transactional database, you should weigh the cost of maintaining an index on tables that incur heavy changes. The bottom line is to use effective indexes judiciously. On analytical databases, use as many indexes as necessary to read the data quickly and efficiently.

4.7.2 Use Index Hints sparingly

Use index hints sparingly as index statistics may change and provide a faster means of computing the query at a later date. In reality, they should never be used (ideally).

Table scans are done because of two main reasons –

1. Requesting so much data that a non-clustered index would incur more i/o than a single table scan
2. An appropriate index does not exist (not always bad, see previous comment).

At any rate, one of the keys to effective data access is to code to the structure. In other words, your WHERE clause should contain the necessary columns to make an index usable. This means that the most significant column of an index must be reference able. A good guideline is to make your request for data very specific and as unique as possible.

4.7.3 Consistently monitor

Indexing is an iterative process. It has to be monitored and amended if necessary. Periodically, run the Index Wizard or Database Engine Tuning Advisor against current Profiler traces to identify potentially missing indexes.

4.7.4 Don’t create redundant indexes

This can adversely affect the performance. Before creating any index ensure there is no index existing which have the same set of column(s).

4.7.5 Ensure all tables have clustered index

As a rule of thumb, every table should have at least the clustered index. Generally, but not always, the clustered index should be on a column that sequentially 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.

4.7.6 Try to create index as unique index

While creating indexes, try to make them unique, if possible. SQL Server can 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.

4.7.7 Create index with proper fillfactor

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.

4.7.8 Try to create Filtered Index (SQL Server 2008)

Filtered index is a new feature added in SQL Server 2008. A filtered index is an optimized non-clustered index, especially suited to cover queries that select from a well-defined subset of data.

Filtered index have following advantages -

- Improved query performance and plan quality
- Reduced index maintenance costs
- Reduced index storage costs

Read more about filtered index here: http://msdn.microsoft.com/en-us/library/cc280372.aspx

4.7.9 General thumb rules while creating clustered indexes

The primary key you select for your table need not always be a clustered index. If you create the primary key and don't specify otherwise, then SQL Server automatically makes the primary key a clustered index.

Clustered indexes are ideal for queries that select by a range of values or where you need sorted results. This is because the data is already presorted in the index for you. Consider clustered index in following situations –

- The query that use BETWEEN and LIKE
- The query which have ORDER BY and GROUP BY
- Clustered index on Identity column or ever increasing column can eliminate INSERT page split
- Clustered indexes are good for queries that look up a record with a unique value (such as an employee number) and when you need to retrieve most or all of the data in the record.

To avoid -

- Avoid using Clustered index on volatile column
- Avoid using clustered index on column which is not unique.

4.7.10 General rules of thumb while creating non-clustered indexes

- Non-clustered indexes are best for queries that return few rows (including just one row) and where the index has good selectivity (above 95%).

- Critical query which needs to be covered so that all the data available for the query is available in the index itself. Try to use INCLUDE clause and add the columns to leaf level

- Indexing Foreign key column - Try to create index on foreign keys. Generally indexing foreign key will improve performance; especially it can improve delete performance.

- If a column in a table is not at least 95% unique, then most likely the SQL Server Query Optimizer will not use a non-clustered index based on that column. Because of this, don't add non-clustered indexes to columns that aren't at least 95% unique. For example, a column with "yes" or "no" as the data won't be at least 95% unique.

- Keep the "width" of your indexes as narrow as possible, especially when creating composite (multi-column) indexes. This reduces the size of the index and reduces the number of reads required to read the index, boosting performance.

- If possible, try to create indexes on columns that have integer values instead of characters. Integer values have less overhead than character values.

- If you know that your application will be performing the same query over and over on the same table, consider creating a covering index on the table.

A covering index includes all of the columns referenced in the 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.

On the other hand, if the index gets too big (too many columns), this can increase I/O and degrade performance.

- An index is only useful to a query if the WHERE clause of the query matches the column(s) that are leftmost in the index.

  So if you create a composite index, such as "City, State", then a query such as "WHERE City = 'Houston'" will use the index, but the query "WHERE STATE = 'TX'" will not use the index.

- Keep index and column statistics up-to-date. Avoid excessive indexes on tables that have a high proportion of writes vs. reads.

- Do not create indexes that contain the same column. For example, instead of creating two indexes on LastName, FirstName and LastName, eliminate the second index on LastName.

- Avoid creating indexes on descriptive CHAR, NCHAR, VARCHAR, and NVARCHAR columns that are not accessed often. These indexes can be quite large.

- If a column requires consistent sorting (ascending or descending order) in a query, for example:

  ```
  SELECT LastName, FirstName
  FROM Customers
  WHERE LastName LIKE ‘N%’
  ORDER BY LastName DESC
  ```

  Consider creating the index on that column in the same order, for example:

  ```
  CREATE CLUSTERED INDEX lastname_ndx ON customers(LastName, FirstName) DESC
  ```

  This prevents SQL Server from performing an additional sort on the data.

4.8. Transaction Handling

- Keep transaction boundary short
- Avoid nested transactions as much as possible
- Don't use transactions inside stored procedures unless the stored procedure directly or indirectly executes more than one insert, update, or delete statement
- While using transactions inside stored procedures, name your transactions
- Before exiting the stored procedure you must commit or rollback any transactions you began
- Don't use implicit transactions ("SET IMPLICIT_TRANSACTIONS ON")

4.9. CLR Integration guidelines

- The CLR stored procedures are efficient when the vector operations or complex mathematical operations are to be performed. For set based operations T-SQL is the best
- CLR would be better option for string processing than T-SQL
- CLR would be better in case application need access to an external resource
- Avoid data access logic inside CLR procedures
- When there is a need to use extended stored procedures, it is better to go for CLR procedures as an alternative
- Set CLR code access security to the most restrictive as possible
- For Error handling CLR may be a better option since it offers more flexibility

4.10. XML Usage Guidelines

4.10.1 Usage of OPENXML

Stored procedures have the ability to accept XML and parse it into individual insert/update statements. This XML can contain many rows across many tables to affect in one document. This provides a significant performance increase because you only have the overhead of one network round trip for the entire commit rather than one round trip per modification to the database. However it is not the right choice for all kinds of applications. Some guidelines on the usage of OPENXML are given below –

1. OPENXML is not recommended for online transactions with large number of concurrent users. This is typically recommended for batch processes where you may need to process and send large number of records for update into the database

2. It is recommended to keep the size of the XML small and not process a huge XML document as this puts the SQL Server memory under pressure due to the recursive nature of processing the XML. Typical tests show that it is better to keep it under 300K, but this number can vary based on the hardware and usage scenarios

3. Bulk DML operations using OPENXML is not recommended if the requirement is to track individual rows for failure or success.

4. Attribute centric XML structure is preferred since the size is small

4.10.2 XML or Relational Storage

4.10.2.1 Reasons for Storing XML Data in SQL Server instead of storing in file system:

Here are some reasons for using native XML features in SQL Server 2005 and later versions as opposed to managing your XML data in the file system:

- You want to use administrative functionality of the database server for managing your XML data (for example, backup, recovery and replication).
- You want to share, query, and modify your XML data in an efficient and transacted way. Fine-grained data access is important to your application. For example, you may want to extract some of the sections within an XML document, or you may want to insert a new section without replacing your whole document.
- You have relational data and SQL applications, and you want interoperability between relational and XML data within your application. You need language support for query and data-modification for cross-domain applications.
- You want the server to guarantee well-formed data, and optionally validate your data according to XML schemas.

- You want indexing of XML data for efficient query processing and good scalability, and the use a first-rate query optimizer.
- You want SOAP, ADO.NET and OLE DB accesses to the XML data.

4.10.2.2 Follow these guidelines when deciding the choice of storage (XML or relational)

If your data is highly structured with known schema, the relational model is likely to work best for data storage.

XML is a good choice if you want a platform-independent model in order to ensure portability of the data by using structural and semantic markup. Additionally, it is an appropriate option if some of the following properties are satisfied:

- Your data is sparse or you do not know the structure of the data, or the structure of
- Your data may change significantly in the future.
- Your data represents containment hierarchy, instead of references among entities, and may be recursive.
- Order is inherent in your data.
- You want to query into the data or update parts of it, based on its structure.

4.11. Recommended naming standards in SQL Server

4.11.1 Naming standard for Stored Procedures

- It is strongly recommended that you do not create any stored procedures using sp_ as a prefix
- Do Not use Special character or space in object name or Column names
- Name the stored procedure by the function it performs. The following list contains the list of functions:

  - csp_Table_Find<Field> for SELECT <field> procedures
  - csp_Table_Get for SELECT <columns> procedures
  - csp_Table_Ins for INSERT INTO procedures
  - csp_Table_Del for DELETE procedures
  - csp_Table_Upd for UPDATE procedures

Please note that in “csp”, “c” can be replaced with any custom notation denoting either company name, application name etc..

4.11.2 Naming standard for Indexes

Each index should begin with idx_<table>_<columns>,

Where, <table> is the name of the table the index is applied on. If the name for the index exceeds the system limit, find a more appropriate name for the <columns> field. Keep the idx_<table> prefix for consistency and easy recognition of the index.

4.11.3 Variables and Parameters

- All variables should be explicitly declared at the top of each stored procedure. Place each declaration on its own line. Explain what each parameter is for:

```
DECLARE @lCaseStart integer -- Starting case number
DECLARE @dStartDate datetime -- First posted date of cases
DECLARE @dEndDate datetime -- Last posted date of cases
DECLARE @lCaseCount integer -- Running total of cases
```

- Do not make a variable name that begins with '@@'. Although it is a legitimate thing to do, the '@@' is typically reserved for SQL Server global functions. This makes readers confused in thinking a system variable or function when it is not.

```
DECLARE @sGoodName varchar(50) -- Good name
DECLARE @@sBadName varchar(50) -- Bad name
```

4.11.4 User-Defined functions

A user-defined function may be prefixed by "udf_" and then a description that logically follows the function process. The description should be proper case words with no spaces.

Examples:
udf_GetNextID
udf_SumOrderLines

4.11.5 Others

All the database objects like tables, stored procedures, views, functions etc should be referenced in their fully qualified name like [database].[owner].[ObjectName].

For example, Sales.dbo.SearchCustomer, Sales.dbo.Customer.

If database name is not fixed, use [owner].[ObjectName].

4.12. General Guide lines

4.12.1 Keep a Baseline performance report

The first thing in any performance tuning or migration/ upgradation project is to get the baseline report. We should know what was the performance matrix before starting tuning or migration.

4.12.2 Write ANSI standard Code

- You must write standard code which will scale your application, that is, migration to next version will not be an issue.
- Do not use Deprecated features. For eg. There are DBCC Command to rebuild index but in SQL Server 2005 and later versions in order to standardize things, you have ALTER INDEX command which does the same thing.

4.12.3 Do not give Error message which expose your architecture to the frontend

I have seen folks giving very detailed error message which tells you " blah blah table do not have this rows in blah blah database" kind which can be a invitation to the hacker.

4.12.4 Use proper Isolation level required for the application

This is very significant. Before going for any isolation level, you must know the implication. All application cannot afford READUNCOMMITTED Isolation level since it can cause data inconsistency issues like Dirty Read, Phantom read, Lost Update etc. WITH NOLOCK Hint is nothing but READ UNCOMMITTED isolation level.

4.12.5 Keep the Database Object Scripts in Source Control

We all are aware of this point but generally ignore. This is important for fault tolerance and management when multiple developers are working on same project.

4.12.6 Install latest tools provided by Microsoft

If possible install all the tools provided by Microsoft. For example, if running SQL Server 2005 or later versions, SP2 or later, install the free SQL Server Performance Dashboard. It can be used for real-time monitoring and performance troubleshooting.

4.12.7 Data compression for IO Intensive tables

This new feature provides the ability to easily enable or disable data compression as an online command as well as offer more efficient data storage above and beyond traditional data compression

4.13. SQL Server Data Load Best practices

Any data load operation can be optimized if we use minimally logged operators provided by sql server in conjunction with features like Partition. Also indexes on the target table play a huge role here.

Here are few best practices which can be followed in Data load operation. For more info refer http://msdn.microsoft.com/en-us/library/dd425070.aspx

4.13.1 Minimally Logged operation

If possible change your database recovery model to bulk-logged or Simple. Changing recovery model will have an impact on backup strategy.

4.13.2 Trace Flag 610

SQL Server 2008 introduces trace flag 610, which controls minimally logged inserts into indexed tables. Additionally, a bulk load operation will be minimally logged even without the trace flag when both of the following conditions are true:

- A WITH (TABLOCK) hint is specified on the target table
- The target table is empty

4.13.3 Drop all the indexes, if possible Clustered index too

If the data load is huge and there are lot of indexes on the table, dropping index can speed up the data load process. An INSERT statement that takes its rows from a SELECT operation and inserts them into a heap is minimally logged when a WITH (TABLOCK) hint is used on the destination table

The following syntax should be used to achieve minimal logging into heaps.

```
INSERT INTO <DestinationTable> (<Columns>) WITH (TABLOCK)
SELECT <Columns> FROM <SomeStatement>
```

4.13.4 For any huge data load use BCP, BULK INSERT or SELECT INTO kind of statements

To provide fast data insert operations these are few data load methods SQL Server offer. BCP can be used for both EXPORT and IMPORT of data, whereas BULK INSERT can only be used for loading data into a SQL Server table. Otherwise both will give you same performance.

4.13.5 Run SSIS on different Machine

If you run DTEXEC on a different server than SQL Server, Integration Services can deliver very high speed by offloading data conversions from the Database Engine.

Use SSIS Stream feature if possible : Note that INSERT … SELECT does not allow concurrent inserts into a single table. If used to populate a single table, Integration Services will often be a faster option because you can run multiple streams in parallel.

4.13.6 Use TRUNCATE TABLE instead of DELETE

Just like DROP TABLE, TRUNCATE TABLE is a metadata-only operation. Be aware that certain limitations apply to TRUNCATE TABLE. These are:

- You cannot truncate a table that is referenced by foreign key constraints.
- You can truncate the entire table only, not a single partition (but see “Deleting All Rows from a Partition or Table”).

4.13.7 Use Partition

SQL Server 2005 introduced partitions and partition switching. Partition switching does not physically move the data; it is a metadata-only operation. It allows you to load many similar data streams in parallel into multiple tables. After this action is complete, issue the SWITCH command to create one, large table.

4.13.8 Number of Flat file and number of CPUs

Use the same number of flat files with CPU (core) for exporting and importing data if you use flat file data load.

4.13.9 Ensure free space in Server

Make sure the system has sufficient disk space for staging tables, indexes, logging, and tempdb when you use SELECT INTO or INSERT SELECT migration. Additional disk space is required for flat files when you use flat file migration.

4.14. SQL Server Security Best Practices

4.14.1 Surface Area Reduction : Install only the required components.

- Install only those components that you will immediately use. Additional components can always be installed as needed.
- Enable only the optional features that you will immediately use.
- Review optional feature usage before doing an in-place upgrade and disable unneeded features either before or after the upgrade.
- Develop a policy with respect to permitted network connectivity choices. Use SQL Server Surface Area Configuration to standardize this policy.
- Develop a policy for the usage of optional features. Use SQL Server Surface Area Configuration to standardize optional feature enabling. Document any exceptions to the policy on a per-instance basis.
- Turn off unneeded services by setting the service to either Manual startup or Disabled.

4.14.2 Service Account

Configure service with proper service account (domain account if possible) with minimum required privilege. Each service have its own significance and different security needs. Do not share same service account for al the services.

- Use a specific user account or domain account rather than a shared account for SQL Server services.
- Use a separate account for each service.
- Do not give any special privileges to the SQL Server service account; they will be assigned by group membership.
- Manage privileges through the SQL Server supplied group account rather than through individual service user accounts.
- Always use SQL Server Configuration Manager to change service accounts.
- Change the service account password at regular intervals.
- Use CREDENTIALs to execute job steps that require specific privileges rather than adjusting the privilege to the SQL Server Agent service account.

If an agent user needs to execute a job that requires different Windows credentials, assign them a proxy account that has just enough permissions to get the task done.

4.14.3 Authentication

Windows authentication is the most preferred and secured authentication mode for SQL Server

- Always use Windows Authentication mode if possible.
- Use Mixed Mode Authentication only for legacy applications and non-Windows users.
- Use the standard login DDL statements instead of the compatibility system procedures.
- Change the **sa** account password to a known value if you might ever need to use it. Always use a strong password for the **sa** account and change the **sa** account password periodically.
- Do not manage SQL Server by using the **sa** login account; assign **sysadmin** privilege to a knows user or group.
- Rename the **sa** account to a different account name to prevent attacks on the **sa** account by name.

4.14.4 Network connectivity

- Limit the network protocols supported.
- Do not enable network protocols unless they are needed.
- Do not expose a server that is running SQL Server to the public Internet.
- Configure named instances of SQL Server to use specific port assignments for TCP/IP rather than dynamic ports.
- If you must support SQL logins, install an SSL certificate from a trusted certificate authority rather than using SQL Server 2005 (or later versions) self-signed certificates.
- Use "allow only encrypted connections" only if needed for end-to-end encryption of sensitive sessions.
- Grant CONNECT permission only on endpoints to logins that need to use them. Explicitly deny CONNECT permission to endpoints that are not needed by users or groups.

4.14.5 System stored procedures or extended stored procedure

- Disable **xp_cmdshell** unless it is absolutely needed.
- Disable COM components once all COM components have been converted to SQLCLR.
- Disable both mail procedures (Database Mail and SQL Mail) unless you need to send mail from SQL Server. Prefer Database Mail as soon as you can convert to it.
- Use SQL Server Surface Area Configuration to enforce a standard policy for extended procedure usage.
- Document each exception to the standard policy.
- Do not remove the system stored procedures by dropping them.
- Do not DENY all users/administrators access to the extended procedures.

4.14.6 Password Policy

- Mandate a strong password policy, including an expiration and a complexity policy for your organization.
- If you must use SQL logins, ensure that SQL Server 2005 (or later versions) runs on the Windows Server 2003 operating system and use password policies.
- Outfit your applications with a mechanism to change SQL login passwords.
- Set MUST_CHANGE for new logins.

4.14.7 Manage *administrator* privilege wisely

- Use administrator privileges only when needed.
- Minimize the number of administrators.
- Provision admin principals explicitly.
- Have multiple distinct administrators if more than one is needed.
- Avoid dependency on the builtin\administrators Windows group.

4.14.8 Database Owner ship and trust

Be aware of Ownership chaining in SQL Server. In later version this is by default off.

- Have distinct owners for databases; not all databases should be owned by **sa**.
- Minimize the number of owners for each database.
- Confer trust selectively.
- Leave the Cross-Database Ownership Chaining setting off unless multiple databases are deployed at a single unit.
- Migrate usage to selective trust instead of using the TRUSTWORTHY property.

4.14.9 Use Schema to group, manage and secure objects

- Group like objects together into the same schema.
- Manage database object security by using ownership and permissions at the schema level.
- Have distinct owners for schemas.
- Not all schemas should be owned by **dbo**.
- Minimize the number of owners for each schema.

4.14.10 Authorization

Authorization is the process of granting permissions on securables to user. In SQL Server, securables are database objects. SQL Server principals include both instance-level principals, such as Windows logins, Windows group logins, SQL Server logins, and server roles and database-level principals, such as users, database roles, and application roles.

- Encapsulate access within modules.
- Manage permissions via database roles or Windows groups.
- Use permission granularity to implement the principle of least privilege.
- Do not enable **guest** access.
- Use users without logins instead of application roles

4.14.11 Usage of Remote Data Sources

- Phase out any remote server definitions.
- Replace remote servers with linked servers.
- Leave ad hoc queries through linked servers disabled unless they are absolutely needed.
- Use constrained delegation if pass-through authentication to a linked server is necessary.

4.14.12 Data Encryption

- Encrypt high-value and sensitive data.
- Use symmetric keys to encrypt data, and asymmetric keys or certificates to protect the symmetric keys.
- Password-protect keys and remove master key encryption for the most secure configuration.
- Always back up the service master key, database master keys, and certificates by using the key-specific DDL statements.
- Always back up your database to back up your symmetric and asymmetric keys.

4.14.13 Audit only the required information

- Auditing is scenario-specific. Balance the need for auditing with the overhead of generating addition data.
- Audit successful logins in addition to unsuccessful logins if you store highly sensitive data.
- Audit DDL and specific server events by using trace events or event notifications.
- DML must be audited by using trace events.
- Use WMI to be alerted of emergency events.
- Enable C2 auditing or Common Criteria compliance only if required

4.15. Storage Best practices

Proper configuration of IO subsystems is critical to the optimal performance and operation of SQL Server systems. Below are some of the most common best practices that the SQL Server team recommends with respect to storage configuration for SQL Server.

1. Understand the IO characteristics of SQL Server and the specific IO requirements / characteristics of your application.
2. More / faster spindles are better for performance
3. Validate configurations prior to deployment
4. Always place log files on RAID 1+0 (or RAID 1) disks. This provides:
5. Isolate log from data at the physical disk level
6. Consider configuration of TEMPDB database
7. Lining up the number of data files with CPU's has scalability advantages for allocation intensive workloads

For more info: http://technet.microsoft.com/hi-in/library/cc966534(en-us).aspx

4.16. Clustering Best practices

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.

4.17. SQL Server Mirroring best practices

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.

4.18. Log Shipping best practices

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.

5 APPENDIX (REFERENCES &COURTESY)

- SQL Server Best Practices : http://msdn.microsoft.com/en-us/sqlserver/bb671432.aspx
- Microsoft® SQL Server® 2008 R2 Best Practices Analyzer
- SQL Server 2005 Best Practices Analyzer (August 2008)