**General SQL Best Practices**

1. Do NOT write triggers to ensure referential integrity. Instead use the foreign key constraint that is already being provided by SQL Server.
2. Use fully qualified names i.e. [SchemaName].[ObjectName]
3. Use SET NOCOUNT ON, to reduce network traffic and improve performance.
4. Avoid Functions in select statement to avoid indexes being ignored
5. Use proper size for the input parameters to reduce memory usage
6. Avoid or minimize the usage of negative operators like NOT EQUAL TO (<>), NOT IN, NOT EXISTS as they cause table scan
7. Avoid Cursors wherever possible, as they force the database engine to repeatedly fetch rows, negotiate blocking, manage locks and transmit results
8. Do not use WITH RECOMPILE as it causes the procedure to recompile each time it is called.
9. Assign procedural parameters to local variables, thus allowing the database engine to generate a generically valid execution plan by avoiding parameter sniffing
10. Use a semicolon (;) to suffix all T-SQL statements
11. Instead of “SELECT \*”, use the required columns to retrieve the data
12. Use column list in INSERT statements
13. Avoid Temporary tables wherever possible. Use Table Variables for small size of data
14. Declare Character data type variable with proper size
15. Avoid Wildcard Search
16. Use UNION ALL/UNION instead of OR if a query is performing poorly
17. Use UNION ALL over UNION wherever possible when the query doesn’t return duplicate value, as DISTINCT will enforce sorting resulting in performance hit.
18. Use Derived tables in place of sub-queries
19. Use TRUNCATE over preference to DELETE where applicable
20. Use MERGE Statement where ever possible

**Stored Procedures**

1. Use Stored Procedures (SPs) wherever you can and comment your SPs
2. DO NOT start the name of a stored procedure with ‘SP\_’
3. Keep SPs small in size and call them within other to avoid large procedures
4. Return only the stored procedure completion status, but not data. If you need to return data, use OUTPUT parameters
5. Use TRY… CATCH block. Avoid capturing @@ERROR

**Views**

1. Use views appropriately to incorporate complicated joins and calculations, so that joins/calculations need not be repeated in all the queries, instead just select from the view.
2. Use indexed views only if the view has, 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.

**Index**

1. Use index hints sparingly as index statistics may change and provide a faster means of computing the query at a later date.
2. Indexing is an iterative process. It has to be monitored and amended if necessary.
3. Don’t create redundant indexes; this can adversely affect the performance
4. Ensure all tables have clustered index
5. Try to create index as unique index because SQL Server can search through a unique index faster than a non-unique index.
6. Create index with proper fillfactor
7. Use Filtered index (is a new feature added in SQL Server 2008), it is an optimized non-clustered index, especially suited to cover queries that select from a well-defined subset of data.
8. 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 –
9. Thumb Rule for Non Clustered Index.

* 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 to improve performance; especially in delete operations.
* DO NOT 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 and SQL Optimizer will not use this index.
* Keep the "width" of the indexes as narrow as possible, especially for composite (multi-column) indexes. This reduces the size of the index and reduces the number of reads required to read the index, boosting performance.
* 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. 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.
* 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.
* Avoid creating indexes on descriptive CHAR, NCHAR, VARCHAR, and NVARCHAR columns that are not accessed often. These indexes can be quite large.
* Create the index on a column in the same order as the query needs, preventing SQL Server from performing an additional sort on the data.

SQL Server Data Load Best practices

1. Minimally Logged operation - Change the database recovery model to bulk-logged or Simple.
2. Trace Flag 610 - Controls minimally logged inserts into indexed tables.
3. Drop all the indexes, if possible Clustered index to speed up the data load process.
4. For any huge data load use BCP, BULK INSERT or SELECT INTO kind of statements
5. Run SSIS on different Machine – When DTEXEC is run on a different server than SQL Server, Integration Services can deliver very high speed by offloading data conversions from the Database Engine
6. Use TRUNCATE TABLE instead of DELETE
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.
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.
9. Ensure free space in Server

SQL Server Security Best Practices

1. Service Account - Configure service wth 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.
2. Authentication - Windows authentication is the most preferred and secured authentication mode for SQL Server
3. Consider Network connectivity
4. Password Policy - Mandate a strong password policy, including an expiration and a complexity policy for your organization.
5. Manage administrator privilege wisely
6. Database Owner ship and trust
7. Use Schema to group, manage and secure objects
8. Authorization
9. Usage of Remote Data Sources
10. Data Encryption
11. Audit only the required information

For detailed explanation, refer the below document

