SQL Server Optimization-Best Practices for Maximizing Performance

AzraJabeen Mohamed Ali

Independent researcher
California, USA
Azra.jbn@gmail.com

Abstract

This paper explores the best practices for SQL Server optimization, offering a comprehensive guide to enhance the performance of database systems.In the data-driven world of today, sustaining high efficiency and responsiveness requires that SQL Server databases operate at their best.By addressing key aspects such as query tuning, indexing strategies, and resource management, it presents effective techniques to minimize latency and improve execution speed. It also highlights the importance of proper configuration, efficient use of memory, and effective database maintenance practices. Through these best practices, database administrators and developers can ensure that SQL Server operates at peak performance, supporting faster queries, reduced downtime, and seamless scalability.This paper serves as an invaluable resource for anyone seeking to optimize their SQL Server environment, ensuring better performance and reliability in real-world applications.

Keywords: SQL, Relational Database, Index, Latency, Scalability, Reliability, Query, Optimization, Index

1. Introduction

Developing effective, maintainable, and high-performance database applications requires an understanding of and adherence to SQL best practices. In order to maximize query efficiency, improve readability, and guarantee scalability in relational database systems, this paper explores the fundamental SQL best practices. The correct use of indexing, query optimization strategies, preserving data integrity with suitable restrictions, and creating clear, effective SQL code that minimizes complexity and redundancy are among the topics discussed. It also covers the importance of normalization and denormalization techniques, efficient transaction management, and the most efficient methods for handling errors. Developers and database administrators can minimize potential hazards and performance bottlenecks while creating reliable, quick, and readily maintainable database systems by following these SQL best practices. Optimizing SQL Server performance is crucial for ensuring efficient query execution and resource management. Below are key SQL Server optimization techniques that can help improve performance:

A. Query Optimization:

A.1: Implementing Execution plan:

A SQL Query Execution Plan is a roadmap used by the SQL Server query processor to determine how a query will be executed. It outlines the steps SQL Server will take to retrieve, process, and return the requested data. These plans are essential for understanding query performance and optimizing database

interactions.To learn how SQL Server is carrying out the query, it is necessary to examine the query execution plan always. Instead of seeks, missing indexes, or expensive operators, look for scans. Actual Execution Plan is the real plan generated after the query is executed. It shows the steps SQL Server took during execution, including the actual number of rows processed and time spent on each operation.

A.2: Fetch necessary columns and Avoid Select *:

Instead of using SELECT *, specify only the required columns.WhenSELECT * is used, SQL Server retrieves all columns from the table, even ifit is not needed. This can increase the amount of data transferred over the network, especially when working with large tables. By specifying only the necessary columns, SQL Server can optimize query execution, making better use of indexes and avoiding unnecessary table scans.When listing the columnsexplicitly, it's clear that what data is being retrieved, improving code readability and maintainability.If the table structure changes (e.g., columns are added or removed), using SELECT * can lead to errors or unintended behavior, as your query might retrieve extra or incorrect data.Using SELECT * can inadvertently pull columns with large data types (e.g., TEXT, BLOB, or VARCHAR(MAX)) that are unnecessary for your query, leading to higher memory and CPU usage.Fig-1 shows which query to be avoided and which one to be considered.

```
--Avoid
SELECT * FROM PRODUCTS
--Implement
SELECT PRODUCT_ID,PRODUCT_NAME,  PRODUCT_SERIAL_ID FROM PRODUCTS
```

Fig-1

A.3: Efficient JOIN optimization:

In SQL Server, joins are used to combine data from two or more tables based on a related column between them. Understanding and using the correct type of join is essential for retrieving data efficiently and ensuring the accuracy of results. INNER JOIN is to be usedwhen there is a need to return rows where there is a match in both tables.LEFT JOIN is to be used when there is a need to include all rows from the left table, regardless of whether there is a match on the right table.RIGHT JOIN is to be used when there is a need to include all rows from the right table, even if there's no corresponding match in the left table.FULL JOIN is to be used to retrieve all rows from both tables, including those without matching rows in the other table.

A.3.1: Be careful with FULL JOIN and CROSS JOIN:FULL JOIN and CROSS JOIN can result in large, potentially unmanageable result sets, especially when working with large tables.So, it is necessary to be careful while handling these joins.

A.3.2: Always Specify the Join Condition: It is to ensure to have ON condition when performing any type of join. Failing to do so may result in unexpected results or a Cartesian product. Fig-2 specifies the query to be avoided and preferred using ON.

```
--Avoid
SELECT * FROM PRODUCTS, PRODUCTS_SALE WHERE PRODUCTS.PRODUCT_ID = PRODUCTS_SALE.PRODUCT_ID
--Implement
SELECT A.PRODUCT_ID,A.PRODUCT_NAME,  A.PRODUCT_SERIAL_ID FROM PRODUCTS A JOIN PRODUCTS_SALE B
ON A.PRODUCT_ID = B.PRODUCT_ID
```

Fig-2

A.3.3: Use INNER JOIN for Default Behavior: Whenever it is certain that it is expected to retrieve the rows with matches in both tables, it is better to use INNER JOIN. It's more efficient than other join types in most cases.

A.4: Avoid Cursors:

Cursors are often used to iterate over rows in a result set one at a time.Although cursors might be helpful in some circumstances, they are usually discouraged due to the severe performance degradation they can cause, particularly when working with huge data sets. Cursors are slower than set-based operations since they process a row at a time.SQL Server is optimized for set-based processing (handling multiple rows at once), and using cursors goes against this optimization.Cursors consume additional resources like memory and CPU because of the context-switching between SQL Server and the cursor's internal processing.By replacing cursors with set-based operations, joins, MERGE statements, or other alternatives, you can drastically improve the performance and readability of your queries. Always aim to leverage SQL Server's strengths in handling data in bulk, as opposed to row-by-row processing.

A.5: Use Efficient Filters: Avoid complex expressions in WHERE clauses. Instead, filter on indexed columns whenever possible.Using efficient filters in SQL Server is essential for optimizing query performance and ensuring that queries run quickly, especially as the database grows. Efficient filtering reduces the amount of data that SQL Server needs to process, which in turn helps to minimize query execution time

A.5.1: Prefer Indexed Columns in Filters: If the columns used in the WHERE clause are indexed, SQL Server may greatly speed up queries.To prevent extra table lookups for large tables, think about using covering indexes, which are indexes that contain all columns needed by a query.

A.5.2: Avoid SELECT DISTINCT with Filters: It can occasionally be inefficient to use SELECT DISTINCT with filters, particularly when paired with intricate joins or subqueries. It is preferable to use appropriate filtering to make sure that your query is already producing unique values.

A.5.3: Avoid Functions on Indexed Columns: It is better to avoid utilizing functions (such as UPPER(), LOWER(), CONVERT(), etc.) on indexed columns when filtering, as this hinders SQL Server's ability to use the index effectively.

A.5.4: Limit the Rows with TOP or Pagination (for Large Data Sets): It is better to use TOP, LIMIT or pagination techniques to restrict the number of rows your query returns if you just require a subset of the data (for example, for a report or for pagination). This improves query performance and helps prevent needless data processing.

A.5.5: Use Specific Conditions Rather Than BETWEEN When Possible: Because the BETWEEN operator is inclusive, it can encompass a larger range of values than is required. Performance may occasionally be enhanced by dividing the condition into two distinct AND conditions, particularly when indexes are used. Fig-3 represents the preferred and avoided method.

```
--Avoid
SELECT PRODUCT_ID,PRODUCT_NAME, PRODUCT_SERIAL_ID FROM PRODUCTS
WHERE PRODUCT_ID BETWEEN '21' AND '91';
--Implement
SELECT PRODUCT_ID,PRODUCT_NAME, PRODUCT_SERIAL_ID FROM PRODUCTS
WHERE PRODUCT_ID >= '21' AND PRODUCT_ID <= '91';
```

Fig - 3

A.5.6: Use IN Instead of Multiple OR Conditions: Because SQL Server can assess the set more quickly, using IN is frequently more efficient than using multiple OR conditions.

A.5.7: Use EXISTS Instead of IN for Subqueries: When working with huge datasets, EXISTS is typically more efficient than IN for subqueries since it ends searching after a match is found, whereas IN might need to execute the full subquery.Fig-4 shows the avoidable and preferred query.

```
--Avoid
SELECT PRODUCT_ID,PRODUCT_NAME, PRODUCT_SERIAL_ID FROM PRODUCTS
WHERE PRODUCT_ID IN( SELECT PRODUCT_ID FROM PRODUCTS_SALE);
--Implement
SELECT PRODUCT_ID,PRODUCT_NAME, PRODUCT_SERIAL_ID FROM PRODUCTS a
WHERE EXISTS(SELECT 1 FROM PRODUCTS_SALE b WHERE  a.PRODUCT_ID = b.PRODUCT_ID );
```

Fig - 4

A5.8:Use JOIN Conditions Rather Than WHERE for Filtering Related Tables: For queries that involve joins, it is better to always place the appropriate filtering condition in the ON clause or directly in the WHERE clause to make use of indexes. Avoid filtering the primary table in the WHERE clause after a join when possible.

A.6: Optimize Aggregations:

Aggregations, such as SUM(), COUNT(), AVG(), MIN(), and MAX(), can be computationally expensive if not handled efficiently.

A.6.1: Avoid Using Aggregates in Subqueries:SQL Server may execute redundant operations when subqueries employ aggregation, especially when big datasets are involved. Try using joins or doing the aggregation in the primary query instead. Fig-5 represents the preferred and avoided method.

```
--Avoid
SELECT PRODUCT_ID,PRODUCT_NAME, PRODUCT_SERIAL_ID,
(SELECT AVG(PRODUCT_COST) FROM PRODUCTS WHERE PRODUCT_ID= x.PRODUCT_ID ) As AvgCost
FROM PRODUCTS x;
--Implement
SELECT PRODUCT_ID,PRODUCT_NAME, PRODUCT_SERIAL_ID,AVG(PRODUCT_COST) As AvgCost
FROM PRODUCTS x
GROUP BY x.PRODUCT_ID,x.PRODUCT_NAME, x.PRODUCT_SERIAL_ID;
```

Fig- 5

A.6.2: Use WITH (NOLOCK) for Read-Only Queries (With Caution): The WITH (NOLOCK) hint can be used to prevent locking in high-concurrency contexts if the aggregating query is read-only but be cautious about the risk of dirty reads (reading uncommitted data).

A.6.3: Avoid DISTINCT in Aggregation Queries: It can occasionally be wasteful to utilize DISTINCT with aggregation since it requires SQL Server to eliminate duplicates before executing the aggregation. To accomplish the same goal, eliminate the DISTINCT if at all possible or use a different strategy. Fig-6 represents the preferred and avoided method.

```
--Avoid
SELECT DISTINCT PRODUCT_ID,COUNT(PRODUCT_SERIAL_ID)
FROM PRODUCTS x GROUP BY PRODUCT_ID;
--Implement
SELECT   PRODUCT_ID,COUNT(DISTINCT PRODUCT_SERIAL_ID)
FROM PRODUCTS x GROUP BY PRODUCT_ID;
```

Fig- 6

A.6.4: Use Window Functions for Aggregations Over Partitions: Window functions (OVER(),

PARTITION BY) can be more effective than running numerous aggregations in a GROUP BY query for some types of aggregation, such as computing running totals or averages over partitions.

A.6.5: Pre-Aggregate Data Using Materialized Views or Indexed Views: Sometimes it helps to pre-aggregate the data in an indexed view, particularly for large datasets or complex aggregations. This method speeds up the retrieval of an aggregation's results by storing them.This indexed view is to be queried quickly, and SQL Server will use the precomputed aggregations when possible.

A.6.6: Avoid Unnecessary HAVING Clauses: Although it should be used rarely, the HAVING clause is frequently used to filter the results of an aggregation. Before executing the aggregate, it is better to filter rows using the WHERE clause. As a result, SQL Server has to process fewer rows.

A.6.7: Batch Processing for Large Datasets: If the data is very large and cannot be aggregated in one go, consider breaking the query into smaller chunks using batch processing with TOP or ROW_NUMBER(). Fig-7 shows the query to process data in smaller batches, reducing the load on SQL Server.

```
WITH ProductList AS (
SELECT PRODUCT_ID, PRODUCT_NAME, PRODUCT_COST
FROM PRODUCTS
WHERE PRODUCT_ID > '10'
ORDER BY PRODUCT_ID
OFFSET 0 ROWS FETCH NEXT 1000 ROWS ONLY

SELECT DISTINCT PRODUCT_ID,SUM(PRODUCT_COST) AS SUM_COST
FROM ProductList x GROUP BY PRODUCT_ID
```

Fig- 7

B. SQL Server Configuration Tuning: SQL Server Configuration Tuning is the process of adjusting various settings within SQL Server to ensure optimal performance, stability, and scalability for your specific workload. SQL Server has many configurable options that can be tuned to improve performance, such as memory management, CPU settings, disk I/O configuration, and query optimization.

B.1: Processor and CPU Configuration: SQL Server performance can be heavily impacted by how it interacts with CPU resources, especially for systems with multiple cores. Properly configuring CPU settings can enhance query performance and resource utilization.

B.1.1: Max degree of Parallelism: For large datasets, enabling parallelism can help SQL Server divide the aggregation work across multiple processors. You can control parallelism with the MAXDOP (maximum degree of parallelism) option. Fig-8 shows the implementation of MAXDOP.

```
SELECT DISTINCT PRODUCT_ID,SUM(PRODUCT_COST)
FROM PRODUCTS x GROUP BY PRODUCT_ID
OPTION (MAXDOP 4); -- Limit parallelism to 4 processors
```

Fig- 8

B.2: Memory configuration: Memory plays a crucial role in the performance of SQL Server. SQL Server dynamically adjusts its memory usage, but it's important to set certain parameters for optimal performance.

B.2.1: Max Server Memory (max server memory): This setting limits the amount of memory that SQL Server can use. It is essential to ensure SQL Server doesn't consume all available memory on the system, leaving none for the OS or other applications. Set the maximum memory (max server memory) to a value that allows SQL Server to use enough memory for caching data while leaving some memory for the

operating system and other processes. The syntax for the same is sp_configure 'max server memory', <desired memory in MB>;RECONFIGURE;

B.2.2: Min Server Memory (min server memory): This setting determines the minimum amount of memory that SQL Server should use. It helps prevent the system from releasing memory under heavy load.Set this value to avoid SQL Server from shrinking its memory under load.

B.2.3: SQL Server Memory Allocation: SQL Server tries to allocate memory dynamically for tasks like caching data and query execution. Ensure that enough memory is allocated based on your server's capacity. Below syntax is to set max memory to 8 GB.

```
sp_configure 'max server memory', 8192;
RECONFIGURE;
```

B.3: SQL Server TempDB Optimization: TempDB is used for sorting, joining, and managing intermediate data. Optimizing TempDB can prevent performance bottlenecks in SQL Server.

B.3.1: Multiple Data Files for TempDB: To avoid contention for resources, create multiple data files for TempDB, usually one for every 4-8 CPU cores.

B.3.2: File Size Management: Set TempDB files to auto-grow in large, fixed increments to avoid frequent small file growth operations that can impact performance.

B.3.3: Place TempDB on Fast Storage: TempDB is heavily used, so placing it on the fastest available storage, such as SSDs, will significantly improve performance.

B.4:Disk I/O Configuration: Disk performance is one of the most important factors affecting SQL Server performance. Proper disk configuration ensures that SQL Server can read and write data efficiently.

B.4.1: Separation of Data Files, Transaction Logs, and TempDB: SQL Server performance can be greatly enhanced by separating the data files, transaction logs, and TempDB across different physical drives. This reduces contention and ensures better disk throughput. It is recommended to place data files and log files on different physical drives, and store TempDB on fast SSD storage.

B.4.2: SQL Server TempDB Configuration: TempDB is heavily used for storing intermediate results, sorting, and managing table versions, so configuring TempDB properly can significantly affect SQL Server performance. It is recommended to increase the number of TempDB data files to match the number of CPU cores (up to 8), and ensure each TempDB file has the same size to avoid contention.

B.4.3: Instant File Initialization: Enabling this feature allows SQL Server to skip zeroing out data files when they are created or expanded, which improves database creation and growth speed.

B.4.4: Write-Caching on Disk: Enable write-caching on disks hosting SQL Server data files to reduce disk write latency. However, ensure the disks are reliable and have power-loss protection.

B.5: Autogrowth Settings: Autogrowth settings determine how SQL Server increases the size of database files when they run out of space. Poorly configured autogrowth settings can lead to fragmentation and

performance degradation. It is recommended to set reasonable autogrowth increments for both data and log files. Using fixed sizes (e.g., 10% or 1 GB) instead of percentages (e.g., 1 MB) can improve performance and prevent excessive fragmentation. Below syntax is for database auto growth.

```
ALTER DATABASE <YourDatabaseName>
MODIFY FILE (NAME = 'YourDataFile', FILEGROWTH = 1024MB);
```

B.6: Database Compatibility Level:SQL Server allows databases to run at different compatibility levels. Each compatibility level introduces optimizations specific to that version of SQL Server. Ensure your database compatibility level matches the version's features and optimizations.If the SQL Server upgraded to a new version recently, then it is recommended to update the database compatibility level to take advantage of new features and performance improvements.

C. **Database Design Improvements:**Database Design Improvements in SQL Server are essential for achieving high performance, scalability, and maintainability. A well-designed database schema ensures efficient data retrieval, updates, and scalability, while reducing the complexity of the system and making it easier to maintain.

C.1: Use Proper Normalization: Normalization is the process of organizing data to minimize redundancy and dependency by dividing a database into smaller tables and linking them using relationships. It is necessary to **avoid excessive normalization** (i.e., reaching the highest normal form possible) in OLTP systems, as it may lead to excessive joins and reduced performance. For OLAP or reporting databases, normalization is often relaxed to improve query performance.

C.2: Use Indexes Effectively: Indexes are essential for improving query performance by reducing the time it takes to search and retrieve data.

C.2.1: Clustered Indexes: It is to ensure the table has a primary key, which automatically creates a clustered index. This should be on frequently queried columns to speed up data retrieval.
C.2.2: Non-Clustered Indexes: Non-clustered indexes are to be created on columns that are frequently used in WHERE clauses, JOIN conditions, or as part of ORDER BY.
C.2.3: Covering Indexes: A covering index includes all the columns needed for a query, reducing the need to access the table's data.

C.3: Use Appropriate Data Types: Choosing the right data types for your columns can significantly impact storage and performance.

C.3.1: Smaller Data Types: It is better to use the smallest data type that can store the data. For example, use TINYINT instead of INT if the column only holds small numbers.

C.3.2: Avoid Overuse of VARCHAR(MAX): While VARCHAR(MAX) is flexible, it can also introduce performance issues when querying large text data. Consider using a more appropriate length (VARCHAR(255) or TEXT).

C.3.3: Use DATE, DATETIME, and TIME Types Correctly: Use DATE for dates without time, TIME for times without dates, and DATETIME or DATETIME2 for full date and time.

C.4: Leverage Foreign Keys and Constraints: Foreign keys ensure data integrity by enforcing relationships between tables. Constraints help enforce business rules and ensure data validity.

C.4.1: Foreign Keys: Ensure that relationships between tables are enforced by using foreign keys to maintain referential integrity.

C.4.2: Check Constraints: Use check constraints to ensure that data inserted into a column meets certain conditions (e.g., salary must be greater than 0).

C.4.3: Unique Constraints: Use unique constraints where applicable to avoid duplicate entries (e.g., email addresses).

C.5: Optimize Stored Procedures and Triggers: Designing efficient stored procedures and triggers can ensure that business logic is executed efficiently within the database.

C.5.1: Use SET NOCOUNT ON: Prevents SQL Server from sending unnecessary row count messages, which can improve performance for stored procedures.

C.5.2: Minimize Trigger Use: Use triggers sparingly as they can lead to unintended side effects and avoid complex logic in triggers.

D. Monitoring and Diagnostic Tools: Monitoring and Diagnostic Tools in SQL Server are essential for ensuring the health, performance, and reliability of your SQL Server environment. These tools help detect issues, optimize performance, and ensure proactive management of SQL Server databases.

D.1: SQL Server Profiler:SQL Server Profiler is a powerful tool that captures real-time SQL Server events, including queries, performance metrics, errors, and system activity. It is especially useful for diagnosing performance problems and identifying problematic queries.Usage of SQL Profiler helps to trace queries and identify bottlenecks.

D.2: Extended Events:SQL Server Extended Events (XEvents) provide a lightweight and flexible framework for collecting, analyzing, and troubleshooting performance data. Extended Events allow us to capture and log a wide range of server events, such as query execution times, deadlocks, login failures, and more. SQL Server's Extended Events are more efficient than Profiler and provide in-depth performance diagnostics.

D.3: Performance Monitor: Windows Performance Monitor (PerfMon) is a powerful tool that provides real-time statistics about SQL Server’s hardware and resource usage. It can be used to monitor memory, CPU, disk, and network performance.Usage of Windows Performance Monitor (PerfMon) helps to track memory, CPU, disk, and other server metrics.

D.4: Dynamic Management Views (DMVs):SQL Server provides Dynamic Management Views (DMVs) to monitor and diagnose server performance. DMVs are system views that provide real-time performance and configuration data. They allow you to query system-level metrics such as active sessions, query performance, index usage, and buffer cache statistics. Use DMVs like sys.dm_exec_query_stats, sys.dm_exec_requests, sys.dm_exec_sessions to monitor and diagnose performance issues in real time.

E. Database Maintenance: Database Maintenance in SQL Server is a critical task to ensure the performance, reliability, and longevity of your database systems. Proper maintenance activities help reduce downtime, optimize query performance, prevent data corruption, and ensure the overall health of the database.

E.1: Automated Backups: Ensure regular automated backups (full, differential, transaction log) are scheduled to protect data.

E.2: Data Integrity Checks: Use DBCC CHECKDB to ensure the integrity of your database.

E.3: Remove Unused Indexes: Periodically review and remove unused indexes using sys.dm_db_index_usage_stats to free up resources.

F. Archiving and Data Pruning: Archiving and Data Pruning in SQL Server are essential techniques for managing large volumes of data over time. As data accumulates, performance can degrade due to slower query times and increasing storage requirements. Archiving and pruning help you maintain database performance and ensure efficient storage management by removing or storing older data that is no longer needed for day-to-day operations.

F.1: Archiving and Data Pruning: For large tables, archive old or unused data periodically to avoid unnecessary overhead on frequently queried data.

F.2: Data Purging: Prune obsolete data from the database where applicable to improve overall performance.

Conclusion:

SQL Server performance optimization is a combination of efficient query design, appropriate indexing, server and database configuration, and periodic maintenance. Constant monitoring and analysis are key to maintaining optimal performance as your database grows and usage patterns evolve. Always test any changes in a development or staging environment before applying them in production.By following these best practices, it is ensured that SQL Server databases are designed for optimal performance and can scale with growing data volumes.

References

[1] Dan Tow“SQL Tuning: Generating Optimal Execution Plans”O’Reilly publications(Dec23, 2003)
[2] Alex Grinberg “XML and JSON Recipes for SQL Server: A Problem-Solution Approach” O’Reilly (Dec 19, 2017)
[3] Markus Winand “SQL Performance Explained Everything Developers Need to Know about SQL Performance “ Markus Winand Publisher (Jul 30, 2012)
[4] Sylvia Moestl Vasilik “SQL Practice Problems: 57 beginning, intermediate, and advanced challenges for you to solve using a “learn-by-doing” approach” Independently Published (Mar13, 2017)
[5] Josephine Bush “Learn SQL Database Programming: Query and manipulate databases from popular relational database servers using SQL “Psckt Publication (May, 2020)
[6] Allen G.Taylor “SQL for dummies 9th Edition” For Dummies Publication (2018)
[7] AnthonyDebarros“Practical SQL 1st edition“ No Starch press(May01, 2018)

[8] Anthony Molinaro, Robert de Graaf "SQL Cookbook: Query Solutions and Techniques for All SQL Users" O'Reilly Media (Jan24, 2006)

[9] Allen G.Taylor "SQL Dummies 9th Edition" For dummies publication (Nov 20, 2018)

[10] Walter Shields "SQL QuickStart Guide: The Simplified Beginner's Guide to Managing, Analyzing, and Manipulating Data With SQL (Coding & Programming - QuickStart Guides)" ClydeBank Media LLC Publication (Nov 18, 2019)