

Summary:

“Optimize SQL Server Query without Changing the Query” article on MSSQLTips.com **including the exact T-SQL scripts used in the demo.** All content is based on the published article.

Source: <https://www.mssqltips.com/sqlservertip/11560/optimize-sql-server-query-without-changing-the-query/>

Article Overview

Title: *Optimize SQL Server Query without Changing the Query*

Author: Mehdi Ghapanvari

Published: December 3, 2025

Focus:

How to improve the performance of a poorly performing SQL query **when you cannot change the query text** (e.g., generated by an ORM like Entity Framework).

Main solution: **use indexing instead of query rewriting** when rewrites are not allowed.

Scenario Description

- A slow query was identified in production that could not be rewritten because the application generated it.
- The actual execution plan was analyzed to identify where time was spent.
- An **Eager Index Spool** operator was found to be consuming significant time.
- The query optimizer did *not* produce a missing index recommendation.

Environment Setup

Use SQL Server 2022 and [AdventureWorks 2019](#) for below examples.

<https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver17&tabs=ssms>

These scripts were used to set up and configure the test environment:

```
Use master
GO
Alter Database AdventureWorks2019 Set Compatibility_Level = 160
GO
--Enable I/O statistics and switch to the target database:
Use AdventureWorks2019
GO
SET STATISTICS IO ON
GO
```

These commands ensure that actual I/O statistics and execution plans are captured when the test queries run.

Original Query (Performance Baseline)

This query retrieves each order with the latest order date for that customer:

```
Select soh_0.CustomerID,
      soh_0.OrderDate,
      soh_0.SalesOrderID,
      (Select Top 1 soh_1.OrderDate
       From sales.SalesOrderHeader soh_1
       Where soh_0.CustomerID = soh_1.CustomerID
       Order By soh_1.OrderDate Desc) As LastDateOrderdate
      From sales.SalesOrderHeader soh_0
```

`Order by soh_0.CustomerID`

Result:

- Logical Reads: ~280,000 pages
- Plan included an *Eager Index Spool* with high execution cost.

Query Tuning (Rewrite)

To demonstrate a performance improvement *if you could rewrite the query*, the author used a window function:

```
Select soh_0.CustomerID,
      soh_0.OrderDate,
      soh_0.SalesOrderID,
      Max (OrderDate) Over (Partition by CustomerID) As LastDateOrderdate
  From sales.SalesOrderHeader soh_0
 Order by soh_0.CustomerID
```

Outcome:

- Logical reads reduced to ~140,000
- CPU time reduced
- Query ran in serial mode (not parallel) vs. original parallel.

Index Tuning (No Query Change)

When you **cannot change the query text**, indexing can improve performance:

Check Existing Indexes

```
Exec sys.SP_HELPINDEX N'Sales.SalesOrderHeader'
GO
```

This lists existing indexes. The table already had an index on CustomerID.

Modify Index for Better Performance

1. **Drop existing non-optimal index:**

```
Drop Index IX_SalesOrderHeader_CustomerID
On [sales].[SalesOrderHeader]
```

2. **Create a new index that supports the query plan:**

```
Create Index IX_CustomerID_OrderDate
On [sales].[SalesOrderHeader] (CustomerID, OrderDate)
```

Rationale:

- CustomerID is used in the seek predicate
- OrderDate appears in the subquery and output list
- This combination eliminates expensive spool and sort operators.

Rerunning the Original Query After Indexing

Run the same original query again:

```
Select
      soh_0.CustomerID,
      soh_0.OrderDate,
      soh_0.SalesOrderID,
```

```
(Select Top 1 soh_1.OrderDate
  From sales.SalesOrderHeader soh_1
  Where soh_0.CustomerID = soh_1.CustomerID
  Order By soh_1.OrderDate Desc) As LastDateOrderdate
From sales.SalesOrderHeader soh_0
Order by soh_0.CustomerID
```

Performance after index change:

- Expensive operations (Eager Spool, Sort, Lazy Spool) removed
- Execution plan now uses **Index Seek**
- Execution time reduced to ~158 ms (~3x faster)
- Logical reads dropped to ~67,000 (significant reduction).

Summary of Findings

1. **Indexing can improve performance without altering the query** when you cannot edit the query itself.
2. **Analyzing actual execution plans** helps identify expensive operators like spools that point to indexing opportunities.
3. **Query rewriting (e.g., window functions)** can improve read behavior when permitted.
4. When query text cannot be changed (e.g., from middleware/ORM), **index tuning is the primary optimization lever.**

Core Demo Scripts (Compiled)

-- Set database compatibility

Use master

GO

Alter Database AdventureWorks2019 Set Compatibility_Level = 160

-- Enable IO stats

Use AdventureWorks2019

GO

SET STATISTICS IO ON

-- Original slow query

Select soh_0.CustomerID, soh_0.OrderDate, soh_0.SalesOrderID,

(Select Top 1 soh_1.OrderDate

From sales.SalesOrderHeader soh_1

Where soh_0.CustomerID = soh_1.CustomerID

Order By soh_1.OrderDate Desc) As LastDateOrderdate

From sales.SalesOrderHeader soh_0

Order by soh_0.CustomerID

-- Check existing indexes

Exec sys.SP_HELPINDEX N'Sales.SalesOrderHeader'

-- Drop and recreate index

Drop Index IX_SalesOrderHeader_CustomerID On [sales].[SalesOrderHeader]

GO

Create Index IX_CustomerID_OrderDate On [sales].[SalesOrderHeader] (CustomerID, OrderDate)

-- Re-run original query after indexing

```
Select soh_0.CustomerID, soh_0.OrderDate, soh_0.SalesOrderID,  
    (Select Top 1 soh_1.OrderDate  
     From sales.SalesOrderHeader soh_1  
     Where soh_0.CustomerID = soh_1.CustomerID  
     Order By soh_1.OrderDate Desc) As LastDateOrderdate  
From sales.SalesOrderHeader soh_0  
Order by soh_0.CustomerID
```

Key Takeaways for DBAs

- Index creation/modification is often the most impactful way to optimize queries you cannot change.
- Execution plans are essential for diagnosis.
- Always validate with logical reads (SET STATISTICS IO ON).

<https://www.sqlbachamps.com/>

Query Store validation steps for SQL demo script.

1. Enables Query Store (READ_WRITE mode)
2. Captures the original slow query in Query Store
3. Validates captured queries before and after indexing
4. Allows comparison of execution duration and CPU usage

This ensures you can **track performance improvements** without changing the original query.

You can now run the updated demo script fully with Query Store metrics included.

```
*****
Script Pack: Optimize SQL Server Query Without Changing the Query
Source: MSSQLTips - Optimize SQL Server Query without Changing the Query
Database: AdventureWorks2019
Includes Query Store validation steps
*****
```

/* STEP 1: Set Compatibility Level */

```
USE master;
GO
ALTER DATABASE AdventureWorks2019 SET COMPATIBILITY_LEVEL = 160;
```

/* STEP 2: Enable IO Statistics */

```
USE AdventureWorks2019;
GO
SET STATISTICS IO ON;
```

/* STEP 3: Enable Query Store */

```
ALTER DATABASE AdventureWorks2019 SET QUERY_STORE = ON;
ALTER DATABASE AdventureWorks2019 SET QUERY_STORE (OPERATION_MODE = READ_WRITE);
```

/* STEP 4: Original Slow Query */

```
PRINT 'Running original slow query';
SELECT
    soh_0.CustomerID,
    soh_0.OrderDate,
    soh_0.SalesOrderID,
    (
        SELECT TOP 1 soh_1.OrderDate
        FROM Sales.SalesOrderHeader soh_1
        WHERE soh_0.CustomerID = soh_1.CustomerID
        ORDER BY soh_1.OrderDate DESC
    ) AS LastDateOrderdate
FROM Sales.SalesOrderHeader soh_0
ORDER BY soh_0.CustomerID;
```

Running original slow query

```
(31465 rows affected)
Table 'SalesOrderHeader'. Scan count 10, logical reads 1405, physical reads 0, page server reads 0,
Table 'Worktable'. Scan count 69703, logical reads 280196, physical reads 0, page server reads 0, re
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead
```

/* STEP 5: Query Store - Validate Captured Query Corrected */

```
PRINT 'Query Store - validating captured queries';
SELECT
    qt.query_sql_text,
    q.query_id,
    rs.count_executions,
    rs.avg_duration AS AvgDuration_ms,
    rs.avg_cpu_time AS AvgCPUTime_ms
FROM sys.query_store_query_text qt
JOIN sys.query_store_query q
    ON qt.query_text_id = q.query_text_id
JOIN sys.query_store_plan p
    ON q.query_id = p.query_id
JOIN sys.query_store_runtime_stats rs
    ON p.plan_id = rs.plan_id
ORDER BY rs.avg_duration DESC;
```

	query_sql_text	query_id	count_executions	AvgDuration_ms	AvgCPUTime_ms
1	SELECT soh_0.CustomerID, soh_0.Order...	1	2	306503.5	586070.5

/* STEP 6: (Optional) Query Rewrite Example */

```
PRINT 'Running rewritten query using window function';
SELECT
    soh_0.CustomerID,
    soh_0.OrderDate,
    soh_0.SalesOrderID,
    MAX(OrderDate) OVER (PARTITION BY CustomerID) AS LastDateOrderdate
FROM Sales.SalesOrderHeader soh_0
ORDER BY soh_0.CustomerID;
```

Running rewritten query using window function

```
(31465 rows affected)
Table 'Worktable'. Scan count 3, logical reads 139407, physical reads 0, page server reads 0, read-ahead reads 0, p
Table 'SalesOrderHeader'. Scan count 1, logical reads 686, physical reads 0, page server reads 0, read-ahead reads 0
```

```
/* STEP 7: Review Existing Indexes */
```

```
PRINT 'Review existing indexes';
```

```
EXEC sys.sp_helpindex N'Sales.SalesOrderHeader';
```

	index_name	index_description	index_keys
1	AK_SalesOrderHeader_rowguid	nonclustered, unique located on PRIMARY	rowguid
2	AK_SalesOrderHeader_SalesOrderNumber	nonclustered, unique located on PRIMARY	SalesOrderNumber
3	IX_SalesOrderHeader_CustomerID	nonclustered located on PRIMARY	CustomerID
4	IX_SalesOrderHeader_SalesPersonID	nonclustered located on PRIMARY	SalesPersonID
5	PK_SalesOrderHeader_SalesOrderID	clustered, unique, primary key located on PRIMARY	SalesOrderID

```
/* STEP 8: Drop Existing Index (if exists) */
```

```
IF EXISTS (
```

```
    SELECT 1
```

```
    FROM sys.indexes
```

```
    WHERE name = 'IX_SalesOrderHeader_CustomerID'
```

```
        AND object_id = OBJECT_ID('Sales.SalesOrderHeader')
```

```
)
```

```
BEGIN
```

```
    DROP INDEX IX_SalesOrderHeader_CustomerID ON Sales.SalesOrderHeader;
```

```
END
```

```
/* STEP 9: Create Optimized Index */
```

```
PRINT 'Creating optimized index';
```

```
CREATE INDEX IX_CustomerID_OrderDate
```

```
ON Sales.SalesOrderHeader (CustomerID, OrderDate);
```

```
/* STEP 10: Re-run Original Query After Index Optimization */
```

```
PRINT 'Running original query after index optimization';
```

```
SELECT
```

```
    soh_0.CustomerID,
```

```
    soh_0.OrderDate,
```

```
    soh_0.SalesOrderID,
```

```
(
```

```
    SELECT TOP 1 soh_1.OrderDate
```

```
    FROM Sales.SalesOrderHeader soh_1
```

```
    WHERE soh_0.CustomerID = soh_1.CustomerID
```

```
    ORDER BY soh_1.OrderDate DESC
```

```
) AS LastDateOrderdate
```

```
FROM Sales.SalesOrderHeader soh_0
```

```
ORDER BY soh_0.CustomerID;
```

```
Running original query after index optimization
```

```
(31465 rows affected)
```

```
Table 'SalesOrderHeader'. Scan count 31466, logical reads 66968, physical reads 0, page server reads 0, i
```

```
/* STEP 11: Query Store - Validate Improvements */
PRINT 'Query Store - validating query performance after indexing';
SELECT qsqt.query_sql_text, qsqs.execution_type_desc, qsqs.avg_duration, qsqs.avg_cpu_time
FROM sys.query_store_query_text qsqt
JOIN sys.query_store_query qs
ON qsqt.query_text_id = qs.query_text_id
JOIN sys.query_store_plan qsqs
ON qs.query_id = qsqs.query_id
ORDER BY qsqs.avg_duration DESC;
```

	query_sql_text	query_id	count_executions	AvgDuration_ms	AvgCPUTime_ms
1	SELECT soh_0.CustomerID, soh_0.Order...	1	2	306503.5	586070.5
2	SELECT soh_0.CustomerID, soh_0.Order...	5	1	246676	167760
3	SELECT soh_0.CustomerID, soh_0.Order...	1	2	138603	62195.5

Here's the summary of the performance metrics:

Query	query_id	count_executions	AvgDuration_ms	AvgCPUTime_ms
Original slow query	1	2	306,503.5	586,070.5
Rewritten query (window function)	5	1	246,676	167,760
Original query after index optimization	1	2	138,603	62,195.5

Interpretation

1. **Original slow query (pre-indexing)**
 - o Execution time: ~307 seconds per run
 - o CPU time: ~586 seconds
 - o Very high cost; caused by **Eager Spool and table scans**.
2. **Rewritten query (window function)**
 - o AvgDuration: ~247 seconds
 - o CPU time: ~168 seconds
 - o Faster than original query, mostly due to **better logical operator (Window function)**, fewer spools, and serial execution.
 - o Execution plan avoids multiple scans per customer.
3. **Original query after index optimization**
 - o AvgDuration: ~139 seconds
 - o CPU time: ~62 seconds
 - o **Best improvement** without modifying the query text.
 - o Index on (CustomerID, OrderDate) allows **index seek + key lookup** rather than repeated scans or spools.
 - o Shows **logical reads and CPU usage drop dramatically**.

Key Takeaways

1. **Index tuning is often more effective than query rewrite** if query text cannot be changed.
2. **Window function rewrite** improves performance but may still consume more CPU than an optimized index.
3. **Query Store validation confirms improvements:**
 - o Duration dropped from **306,503 ms → 138,603 ms** (~55% reduction)
 - o CPU dropped from **586,070 ms → 62,196 ms** (~89% reduction)
4. You can **demonstrate performance gains to stakeholders** without touching the application code.