SQL Server Performance and Troubleshooting in SSMS

Beginner Hands-On Lab Manual

Learn how to troubleshoot slow queries, blocking, missing indexes, statistics issues, wait symptoms, and database performance using SSMS tools and T-SQL diagnostics.

Audience	Beginners learning SQL Server troubleshooting
Estimated Time	2 to 3 hours
Tool	SQL Server Management Studio (SSMS)
Lab Database	PerfTroubleshootingDB
Goal	Identify slow queries, diagnose causes, apply safe fixes, and verify improvements

Training Safety: Use a test SQL Server instance only. Performance troubleshooting often includes changing indexes, statistics, queries, and settings. Never apply tuning changes to production without testing and change control.

Lab Objectives

- Use SSMS tools such as Include Actual Execution Plan, Activity Monitor, Performance Dashboard, and Query Store reports.
- Measure query performance with SET STATISTICS IO and SET STATISTICS TIME.
- Recognize common execution plan clues such as scans, seeks, key lookups, sort warnings, and row estimate differences.
- Identify blocking sessions and long-running requests using Dynamic Management Views.
- Create and test an index improvement safely.
- Enable Query Store and use it to review query runtime history.
- Understand basic wait categories and how they guide troubleshooting.
- Build a repeatable troubleshooting workflow: capture evidence, identify bottleneck, change one thing, retest, and document.

Quick Troubleshooting Reference

Tool / Concept	What It Helps Find	Beginner Action
Actual Execution Plan	How SQL Server ran the query	Look for scans, warnings, high-cost operators, and row estimate gaps
STATISTICS IO/TIME	Logical reads and CPU/elapsed time	Record before and after tuning numbers
Activity Monitor	Current sessions and waits	Check active expensive requests and blocking
Performance Dashboard	CPU, I/O, waits, missing indexes, expensive queries	Use as an overview starting point
Query Store	Query history and plan changes	Find regressed or expensive queries over time
DMVs	Live internal diagnostic data	Use scripts to inspect requests, waits, and indexes

Module 1: Performance Troubleshooting Mindset

Performance troubleshooting is not guessing. It is a repeatable process of measuring, identifying the bottleneck, changing one thing, and measuring again.

- Define the symptom: slow query, blocking, timeout, high CPU, high I/O, or high memory pressure.
- Capture evidence: query text, actual execution plan, IO/time output, and affected time window.
- Identify the likely cause: poor indexing, outdated statistics, blocking, bad query pattern, parameter sensitivity, or resource pressure.
- Apply one controlled change in a test environment.
- Retest and compare before/after results.
- Document what changed and why.

Core rule: Measure first. Tune second. Verify third.

Module 2: Build the Performance Lab Database

Open SSMS, connect to a test SQL Server instance, select New Query, and run the setup script below.

```
CREATE DATABASE PerfTroubleshootingDB;
GO

USE PerfTroubleshootingDB;
GO

CREATE TABLE dbo.Customers
(
    CustomerID INT IDENTITY(1,1) CONSTRAINT PK_Customers PRIMARY KEY,
    CustomerName VARCHAR(100) NOT NULL,
    Region VARCHAR(50) NOT NULL,
    IsActive BIT NOT NULL DEFAULT (1)
);

CREATE TABLE dbo.Orders
(
    OrderID INT IDENTITY(1,1) CONSTRAINT PK_Orders PRIMARY KEY,
    CustomerID INT NOT NULL,
    OrderDate DATE NOT NULL,
    OrderStatus VARCHAR(30) NOT NULL,
    OrderTotal DECIMAL(12,2) NOT NULL,
    CONSTRAINT FK_Orders_Customers FOREIGN KEY (CustomerID) REFERENCES dbo.Customers(CustomerID)
);
GO

INSERT INTO dbo.Customers (CustomerName, Region, IsActive)
VALUES
('North Branch','North',1),('South Branch','South',1),('East Branch','East',1),
('West Branch','West',1),('Central Branch','Central',1);
GO

;WITH n AS
(
    SELECT TOP (10000) ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) AS rn
    FROM sys.all_objects a CROSS JOIN sys.all_objects b
)
INSERT INTO dbo.Orders (CustomerID, OrderDate, OrderStatus, OrderTotal)
SELECT
    ((rn - 1) % 5) + 1,
    DATEADD(DAY, -1 * (rn % 730), CAST(GETDATE() AS DATE)),
    CASE WHEN rn % 10 = 0 THEN 'Cancelled'
         WHEN rn % 4 = 0 THEN 'Open'
         ELSE 'Closed' END,
    CONVERT(DECIMAL(12,2), (rn % 500) + 25.00)
FROM n;
GO
```

Module 3: Baseline a Query with STATISTICS IO and TIME

Before changing anything, capture baseline evidence. Logical reads and elapsed time help prove whether a tuning change improved the query.

```
USE PerfTroubleshootingDB;
```

```
GO

SET STATISTICS IO ON;
SET STATISTICS TIME ON;

SELECT
    c.CustomerName,
    c.Region,
    o.OrderID,
    o.OrderDate,
    o.OrderStatus,
    o.OrderTotal
FROM dbo.Customers c
INNER JOIN dbo.Orders o
    ON c.CustomerID = o.CustomerID
WHERE c.Region = 'North'
  AND o.OrderStatus = 'Open'
  AND o.OrderDate >= DATEADD(DAY, -180, CAST(GETDATE() AS DATE))
ORDER BY o.OrderDate DESC;

SET STATISTICS IO OFF;
SET STATISTICS TIME OFF;
```

Record the logical reads, CPU time, elapsed time, and execution plan shape before making changes.

Module 4: Use Actual Execution Plans in SSMS

- In SSMS, select Query > Include Actual Execution Plan or press Ctrl+M.
- Run the query from Module 3.
- Open the Execution Plan tab.
- Hover over operators to review estimated rows, actual rows, warnings, and operator details.
- Look for Table Scan, Clustered Index Scan, Key Lookup, Sort, Hash Match, and warning icons.

```
-- Helpful plan review questions:
-- 1. Are there scans where a selective index seek would help?
-- 2. Are estimated rows very different from actual rows?
-- 3. Are key lookups repeated many times?
-- 4. Are sorts, spills, or missing index suggestions present?
-- 5. Does the plan use parallelism due to large work?
```

Execution plans show how SQL Server accessed tables, joined rows, filtered data, aggregated data, and sorted results.

Module 5: Apply a Safe Index Improvement and Retest

The query filters by OrderStatus and OrderDate and joins by CustomerID. Create an index that supports that access pattern, then retest the same query.

```
CREATE INDEX IX_Orders_Status_Date_Customer
ON dbo.Orders (OrderStatus, OrderDate, CustomerID)
INCLUDE (OrderTotal);
GO

SET STATISTICS IO ON;
SET STATISTICS TIME ON;
```

```
SELECT
    c.CustomerName,
    c.Region,
    o.OrderID,
    o.OrderDate,
    o.OrderStatus,
    o.OrderTotal
FROM dbo.Customers c
INNER JOIN dbo.Orders o
    ON c.CustomerID = o.CustomerID
WHERE c.Region = 'North'
  AND o.OrderStatus = 'Open'
  AND o.OrderDate >= DATEADD(DAY, -180, CAST(GETDATE() AS DATE))
ORDER BY o.OrderDate DESC;

SET STATISTICS IO OFF;
SET STATISTICS TIME OFF;
```

Compare the before and after results. Did logical reads decrease? Did elapsed time improve? Did the plan change from a scan to a seek?

Index advice must be tested. An index can improve one query but add storage and write overhead for INSERT, UPDATE, and DELETE operations.

Module 6: Use SSMS Activity Monitor and Performance Dashboard

Lab 6.1: Activity Monitor

- In Object Explorer, right-click the SQL Server instance.
- Select Activity Monitor.
- Review Overview, Processes, Resource Waits, Data File I/O, and Recent Expensive Queries.
- Look for blocking sessions, long-running queries, high CPU, and high I/O.

Lab 6.2: Performance Dashboard

- In Object Explorer, right-click the SQL Server instance.
- Select Reports > Standard Reports > Performance Dashboard.
- Review CPU utilization, waiting requests, I/O, expensive queries, missing indexes, blocking, and resource contention reports.
- Use the dashboard to guide deeper investigation, not as the only evidence.

Performance Dashboard is a fast starting point. Always confirm recommendations and test changes before applying them broadly.

Module 7: Enable and Use Query Store

Query Store captures query text, plans, and runtime statistics over time. It is useful when troubleshooting query regressions or reviewing workload patterns.

```
ALTER DATABASE PerfTroubleshootingDB
SET QUERY_STORE = ON
(
    OPERATION_MODE = READ_WRITE,
    CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 30),
```

```
    DATA_FLUSH_INTERVAL_SECONDS = 900,
    INTERVAL_LENGTH_MINUTES = 60
);
GO

-- Run the lab queries several times, then review recent Query Store activity.
SELECT TOP (20)
    qt.query_sql_text,
    q.query_id,
    p.plan_id,
    rs.count_executions,
    rs.avg_duration,
    rs.avg_cpu_time,
    rs.avg_logical_io_reads,
    rs.last_execution_time
FROM sys.query_store_query_text qt
INNER JOIN sys.query_store_query q
    ON qt.query_text_id = q.query_text_id
INNER JOIN sys.query_store_plan p
    ON q.query_id = p.query_id
INNER JOIN sys.query_store_runtime_stats rs
    ON p.plan_id = rs.plan_id
ORDER BY rs.last_execution_time DESC;
```

- In SSMS Object Explorer, expand the database.
- Expand Query Store.
- Review reports such as Regressed Queries, Overall Resource Consumption, and Top Resource Consuming Queries.
- Use Query Store history to understand whether performance changed after a deployment, index change, or statistics update.

Module 8: Troubleshoot Live Requests and Waits with DMVs

Dynamic Management Views expose live activity. These queries are beginner-friendly starting points for current requests and waits.

```
-- Current active requests
SELECT
    r.session_id,
    r.status,
    r.command,
    r.cpu_time,
    r.logical_reads,
    r.reads,
    r.writes,
    r.wait_type,
    r.wait_time,
    r.blocking_session_id,
    DB_NAME(r.database_id) AS DatabaseName,
    SUBSTRING(t.text, 1, 4000) AS QueryText
FROM sys.dm_exec_requests r
CROSS APPLY sys.dm_exec_sql_text(r.sql_handle) t
WHERE r.session_id <> @@SPID
ORDER BY r.cpu_time DESC;

-- Top waits since SQL Server startup or last wait statistics reset
SELECT TOP (20)
```

```
    wait_type,
    waiting_tasks_count,
    wait_time_ms,
    signal_wait_time_ms
FROM sys.dm_os_wait_stats
WHERE wait_type NOT LIKE 'SLEEP%'
ORDER BY wait_time_ms DESC;
```

Wait statistics are clues, not final answers. Use waits to decide what category to investigate: CPU, I/O, memory, locking, network, or parallelism.

Module 9: Simulate and Diagnose Blocking

Blocking occurs when one session holds a lock and another session waits for that lock. Use two separate SSMS query windows for this lab.

Window 1: Create a Blocking Transaction

```
USE PerfTroubleshootingDB;
GO

BEGIN TRANSACTION;

UPDATE dbo.Orders
SET OrderTotal = OrderTotal + 1
WHERE OrderID = 1;

-- Leave this transaction open for the lab.
-- Do not commit or rollback yet.
```

Window 2: Run a Query That Gets Blocked

```
USE PerfTroubleshootingDB;
GO

SELECT *
FROM dbo.Orders
WHERE OrderID = 1;
```

Window 3 or Activity Monitor: Find the Blocker

```
SELECT
    r.session_id,
    r.blocking_session_id,
    r.wait_type,
    r.wait_time,
    DB_NAME(r.database_id) AS DatabaseName,
    t.text AS QueryText
FROM sys.dm_exec_requests r
CROSS APPLY sys.dm_exec_sql_text(r.sql_handle) t
WHERE r.blocking_session_id <> 0;
```

Window 1: Clean Up

```
ROLLBACK TRANSACTION;
GO
```

Never leave test transactions open. In production, investigate the blocker carefully before killing a session.

Module 10: Statistics, Index Health, and Maintenance Checks

Statistics help SQL Server estimate how many rows a query will return. Bad estimates can cause poor execution plans.

```
-- Review statistics objects for a table
EXEC sp_helpstats 'dbo.Orders', 'ALL';
GO

-- Update statistics on one table
UPDATE STATISTICS dbo.Orders;
GO

-- Review index usage since last SQL Server restart
SELECT
    OBJECT_NAME(i.object_id) AS TableName,
    i.name AS IndexName,
    s.user_seeks,
    s.user_scans,
    s.user_lookups,
    s.user_updates
FROM sys.indexes i
LEFT JOIN sys.dm_db_index_usage_stats s
    ON i.object_id = s.object_id
   AND i.index_id = s.index_id
   AND s.database_id = DB_ID()
WHERE OBJECT_NAME(i.object_id) IN ('Orders','Customers')
ORDER BY TableName, IndexName;
```

Do not blindly rebuild every index. Maintenance decisions should consider fragmentation, page count, workload, maintenance window, and testing.

Module 11: Beginner Troubleshooting Playbook

Step	Question	Evidence to Capture
1	What is slow or failing?	User report, query text, error message, time window
2	Is the issue happening now?	Activity Monitor, active requests DMV, blocking DMV
3	Is one query expensive?	Actual execution plan, STATISTICS IO/TIME, Query Store
4	Is there blocking?	blocking_session_id, Activity Monitor Processes
5	Is resource pressure present?	Performance Dashboard CPU, I/O, waits, memory report
6	What changed recently?	Query Store, deployment notes, indexes, stats, data growth

Step	Question	Evidence to Capture
7	What safe fix can be tested?	Query rewrite, index, stats update, batch size change
8	Did the fix help?	Before/after logical reads, CPU, duration, plan comparison

Challenge Exercises

#	Exercise	Done
1	Capture STATISTICS IO and TIME for the baseline query.	☐
2	Enable Actual Execution Plan and identify the largest-cost operator.	☐
3	Create the suggested lab index and compare before/after logical reads.	☐
4	Enable Query Store and run the lab query several times.	☐
5	Use a Query Store query to review recent executions.	☐
6	Use Activity Monitor to review current sessions.	☐
7	Use Performance Dashboard to review expensive queries and missing index suggestions.	☐
8	Simulate blocking with two query windows and identify blocking_session_id.	☐
9	Update statistics on dbo.Orders and rerun the query.	☐
10	Write a short troubleshooting summary with symptom, evidence, fix, and result.	☐

Mini Cheat Sheet

```
-- Measure query work
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
-- Run query here
SET STATISTICS IO OFF;
SET STATISTICS TIME OFF;
```

```
-- Include Actual Execution Plan in SSMS
-- Query menu > Include Actual Execution Plan, or press Ctrl+M

-- Active requests
SELECT session_id, status, command, cpu_time, logical_reads,
       wait_type, blocking_session_id
FROM sys.dm_exec_requests
WHERE session_id <> @@SPID;

-- Top waits
SELECT TOP (20) wait_type, waiting_tasks_count, wait_time_ms
FROM sys.dm_os_wait_stats
ORDER BY wait_time_ms DESC;

-- Enable Query Store
ALTER DATABASE DatabaseName
SET QUERY_STORE = ON (OPERATION_MODE = READ_WRITE);

-- Update statistics
UPDATE STATISTICS dbo.TableName;
```

Final Lab Completion Checklist

Skill	Complete
Created PerfTroubleshootingDB	☐
Captured baseline STATISTICS IO and TIME	☐
Used Actual Execution Plan	☐
Identified common plan operators	☐
Created and tested an index	☐
Compared before/after query performance	☐
Opened Activity Monitor	☐
Viewed Performance Dashboard	☐
Enabled Query Store	☐
Reviewed Query Store runtime data	☐
Used DMVs for active requests	☐
Reviewed wait statistics	☐

Skill	Complete
Simulated blocking	☐
Identified blocking_session_id	☐
Updated statistics	☐
Completed challenge exercises	☐

References

Microsoft Learn: Analyze an actual execution plan - https://learn.microsoft.com/en-us/sql/relational-databases/performance/analyze-an-actual-execution-plan

Microsoft Learn: Execution plan overview - https://learn.microsoft.com/en-us/sql/relational-databases/performance/execution-plans

Microsoft Learn: Server performance and activity monitoring - https://learn.microsoft.com/en-us/sql/relational-databases/performance/server-performance-and-activity-monitoring

Microsoft Learn: Performance Dashboard - https://learn.microsoft.com/en-us/sql/relational-databases/performance/performance-dashboard

Microsoft Learn: Monitor performance by using Query Store - https://learn.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store

Microsoft Learn: Tune performance with Query Store - https://learn.microsoft.com/en-us/sql/relational-databases/performance/tune-performance-with-the-query-store