

SQL Server Activity Monitor

Activity Monitor is a graphical performance-diagnostic utility built into SQL Server Management Studio (SSMS). It provides a near real-time snapshot of server-wide resource consumption and query activity. Although less powerful than DMVs, Query Store, or dedicated monitoring solutions, Activity Monitor is valuable for quick triage during performance incidents.

1. What Activity Monitor Is

Activity Monitor aggregates information from various **Dynamic Management Views (DMVs)** and **performance counters** to display:

1. High-level resource utilization
2. Waits and queue bottlenecks
3. Long-running and expensive queries
4. Blocking chains
5. Sessions and active connections
6. I/O statistics per database file

It is primarily a **diagnostic dashboard**, not a long-term monitoring tool.

2. Components of Activity Monitor

A. Overview Pane

Displays four key system metrics:

- % Processor Time (from OS)
- Waiting Tasks
- Database I/O (MB/sec)
- Batch Requests/sec

Usage: A quick impression of server pressure.

If CPU or waits spike, or batch requests drop, something is wrong.

B. Processes Pane

Shows active user sessions and background SQL Server tasks, including:

- Login name
- Host/application
- Command being executed
- CPU, memory, I/O consumption
- Blocking/blocked status

Usage: Identify rogue sessions, blocking chains, or long-running commands.

C. Resource Waits Pane

Breaks down waiting tasks by wait category:

- CPU
- I/O
- Locks
- Network
- Memory/Buffer Latch
- Page IO Latch
- SOS_SCHEDULER_YIELD
- CXPACKET / CXCONSUMER

Usage: Understand what the server is “waiting” on and which subsystem is the bottleneck.

D. Data File I/O Pane

Shows read/write throughput on each data and log file.

Helps identify:

- Slow disks
- Hot data files
- TempDB contention

E. Recent Expensive Queries Pane

Displays high-cost queries by:

- CPU time
- Logical reads/writes
- Duration
- Execution count

Usage: Identify inefficient queries, missing indexes, or parameter sniffing problems.

3. When to Use Activity Monitor

Activity Monitor is best suited for **real-time investigation**, for example:

1. Sudden CPU spikes
2. Application slows down unexpectedly
3. Users complain: “the database is hanging”
4. Suspected blocking or deadlocking
5. High I/O latency observed
6. Need to quickly identify top CPU or I/O consumers

It is **not suitable** for long-term trending, forecasting, or deep analysis.

4. Detailed Real-World Troubleshooting Scenarios

Scenario 1: High CPU Usage

Symptoms: % Processor Time is high. Application is slow.

Steps in Activity Monitor:

1. Check **Overview** → % Processor Time.
2. Go to **Recent Expensive Queries** → sort by CPU Time.
3. Look for:
 - Missing indexes
 - Bad query plans
 - Parameter sniffing
4. Kill or tune the culprit query.

Follow-up actions:

Review execution plans, update statistics, or add indexes.

Scenario 2: Blocking / Long-Held Locks

Symptoms: Users report timeouts.

Steps:

1. Open **Processes** pane.
2. Filter on **Blocked By** column.
3. Identify the root blocker (the session with no “Blocked By” but many sessions below relying on it).
4. Review the SQL text causing the lock.
5. Decide whether to kill the blocking session or let it complete.

Common causes:

- Uncommitted transaction
- Long update/delete
- Bad isolation levels
- Application code not handling transactions correctly

Scenario 3: I/O Bottlenecks

Symptoms: Slow queries, time spent waiting on disk.

Steps:

1. In **Resource Waits**, check:
 - PAGEIOLATCH_* (slow reads)
 - WRITELOG (log file bottleneck)
 - IO_COMPLETION
2. In **Data File I/O**, sort by Response Time (ms).
3. Identify problematic file → check disk subsystem or TempDB.

Typical mitigations:

- Add more data files for TempDB
- Move files to faster storage (SSD/NVMe)
- Improve indexing

Scenario 4: Sudden Drop in Throughput

Symptoms: Batch Requests/sec dropping sharply.

Steps:

1. In **Overview**, confirm low Batch Requests/sec.
2. Check **Resource Waits** for abnormal spikes.
3. Check **Processes** for external blocking or a session consuming excessive resources.
4. Review expensive queries that suddenly increased execution time.

Several causes:

- Query plan regression
- Parameter sniffing
- CPU/Memory starvation
- Network slowdown

Scenario 5: TempDB Contention

Symptoms: Slowness during heavy insert/select workloads.

Steps:

1. In **Resource Waits**, look for PAGELATCH_* waits.
2. In **Data File I/O**, check TempDB file activity.
3. Confirm number of TempDB data files.

Mitigation:

- Increase TempDB files up to number of cores (with moderation)
- Check for version store pressure

Scenario 6: Excessive Parallelism

Symptoms: High CPU and inconsistent performance.

Steps:

1. In **Resource Waits**, check for CXPACKET or CXCONSUMER waits.

2. In **Recent Expensive Queries**, find parallel queries consuming most CPU.
3. Evaluate:
 - o MAXDOP setting
 - o Cost Threshold for Parallelism
 - o Execution plans

5. Limitations of Activity Monitor

While extremely useful for quick triage, Activity Monitor has constraints:

- Refresh interval introduces latency
- Can cause minor overhead on busy servers
- Cannot store historical performance data
- Less detailed than querying DMVs manually
- Limited filtering and export options
- Not suitable for enterprise-grade monitoring

For long-term or deep troubleshooting, use:

- Query Store
- Extended Events
- Performance Monitor
- SQL Trace (legacy)
- Third-party tools (e.g., SolarWinds DPA, Redgate, SentryOne)

6. Best Practices When Using Activity Monitor

1. Use it as a **first response tool**, not the only tool.
2. Use **DMVs** for detailed drill-down.
3. Correlate what you see with application behavior.
4. Avoid leaving Activity Monitor open on critical servers for long periods.
5. Always validate findings by checking execution plans/logs.

Below is a **comprehensive, structured, and practical set of materials** that a junior DBA can use immediately:

1. A step-by-step practical guide
2. DMV equivalents for Activity Monitor
3. A performance troubleshooting checklist
4. A real case study with root-cause analysis

1. Step-by-Step Practical Guide for Junior DBAs

This guide walks through how to use Activity Monitor and correlate findings with DMVs for deeper analysis.

Step 1: Establish the Baseline

Before troubleshooting, determine what “normal” looks like on your server:

- Average CPU utilization
- Typical Batch Requests/sec
- Normal wait categories
- Standard I/O latency

This prepares you to recognize anomalies during incidents.

Step 2: Start with the Activity Monitor Overview Pane

Check for obvious symptoms:

- **% Processor Time** > 80% for a sustained period
- **Waiting Tasks** spikes
- **Batch Requests/sec** falling (throughput drop)
- **Database I/O** unusually high

This tells you *where* to look next.

Step 3: Investigate High CPU

If CPU is high:

1. Open **Recent Expensive Queries**, sort by CPU.
2. Identify the top offenders (frequent and expensive queries).
3. Review their execution plan.
4. Correlate with:
 - Missing indexes
 - Implicit conversions
 - Parameter sniffing
 - Non-sargable predicates

Use DMVs:

```
SELECT TOP 20
    total_worker_time/1000 AS CPU_ms,
    execution_count,
    total_physical_reads,
    total_logical_reads,
    sql_handle,
    plan_handle
FROM sys.dm_exec_query_stats
ORDER BY total_worker_time DESC;
```

Step 4: Check for Blocking & Deadlocks

If users report "timeouts" or "everything froze":

1. In **Processes**, sort by “Blocked By”.
2. Identify the head blocker (root session).
3. Capture the SQL running in that session.
4. Determine if the query is long-running or holding open a transaction.

DMV to detect blocking:

```
SELECT
    blocking_session_id,
    session_id,
    wait_type,
    wait_time,
    wait_resource
FROM sys.dm_exec_requests
WHERE blocking_session_id <> 0;
```

Step 5: Check Resource Waits

Analyze which subsystem is under pressure:

- High **PAGEIOLATCH_*** = slow storage
- High **WRITELOG** = log bottleneck
- High **SOS_SCHEDULER_YIELD** = CPU pressure
- High **CXPACKET/CXCONSUMER** = parallelism imbalance

DMV:

```
SELECT *
FROM sys.dm_os_wait_stats
ORDER BY wait_time_ms DESC;
```

Step 6: Evaluate Data File I/O

In Activity Monitor, sort by Response Time to find slow files.

Check TempDB specifically.

DMV equivalent:

```
SELECT
    DB_NAME(database_id) AS DBName,
    file_id,
    io_stall_read_ms,
    num_of_reads,
    io_stall_write_ms,
    num_of_writes
FROM sys.dm_io_virtual_file_stats(NULL, NULL)
ORDER BY io_stall_read_ms DESC;
```

Step 7: Identify Expensive Queries

Sort by:

- CPU
- Logical Reads
- Duration
- Execution Count

DMVs for expensive queries:

```
SELECT TOP 20
    qs.total_worker_time AS CPU,
```

```

qs.total_logical_reads AS Reads,
qs.total_elapsed_time AS Duration,
SUBSTRING(qt.text, (qs.statement_start_offset/2)+1,
    ((CASE qs.statement_end_offset
        WHEN -1 THEN DATALENGTH(qt.text)
        ELSE qs.statement_end_offset END
    - qs.statement_start_offset)/2)+1) AS QueryText
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) qt
ORDER BY qs.total_worker_time DESC;

```

Step 8: Take Corrective Action

Depending on the root cause:

- Kill blocking session
- Add or adjust indexes
- Fix missing statistics
- Tune problematic SQL
- Scale up or redistribute workload
- Fix long transactions
- Improve storage subsystem

2. DMV Equivalents for Each Activity Monitor Pane

| Activity Monitor Pane | DMV / System View | Description |
|--|--------------------------------|-------------------------------|
| Overview – CPU, I/O, Waits, Batch Requests | sys.dm_os_performance_counters | CPU, Batch Requests/sec, I/O |
| | sys.dm_os_wait_stats | Wait statistics |
| | sys.dm_io_virtual_file_stats | Per-file I/O |
| Processes | sys.dm_exec_sessions | Session details |
| | sys.dm_exec_requests | Active request details |
| | sys.dm_os_tasks | Task-level details |
| | sys.dm_exec_connections | Connection info |
| Resource Waits | sys.dm_os_wait_stats | Categorized wait statistics |
| Data File I/O | sys.dm_io_virtual_file_stats | File read/write/stall metrics |
| Recent Expensive Queries | sys.dm_exec_query_stats | Cached query stats |
| | sys.dm_exec_sql_text | Query text |
| | sys.dm_exec_query_plan | Execution plans |

3. Sample Performance Troubleshooting Checklist

For use during production incidents.

A. Server-Level Checks

- [] Check CPU utilization
- [] Check memory usage / paging
- [] Check storage latency (read/write stalls)
- [] Check network packet errors

B. SQL Server Instance Checks

- [] Check top wait types
- [] Check number of blocked sessions
- [] Check TempDB contention
- [] Check VLF fragmentation in log files

C. Database-Level Checks

- [] Verify index health (fragmentation, missing indexes)
- [] Check for growing transaction log
- [] Check long-running open transactions

D. Query-Level Checks

- [] Identify top CPU consumers
- [] Identify top I/O consumers
- [] Identify long-running queries
- [] Capture execution plans
- [] Look for implicit conversions
- [] Check parameter sniffing issues

E. Corrective Actions

- [] Kill blocking session (only if necessary)
- [] Update statistics
- [] Add or adjust indexes
- [] Fix problematic queries
- [] Restart services (only as last resort)
- [] Engage storage/network teams

4. Real Case Study with Root-Cause Analysis

Case: Application slowdown during peak hours.

Background

A retail application experiences severe slowness from 10 AM to 12 PM daily.

Symptoms Observed

- CPU > 90%
- High PAGEIOLATCH_SH waits
- Activity Monitor shows several queries with very high logical reads
- Batch Requests/sec dropped from 300 to 120
- Data File I/O shows high read stall times on a specific database

Investigation Steps

Step 1: Identify High CPU Queries

Activity Monitor → Recent Expensive Queries

A nightly report query was executing repeatedly during the day.

Step 2: Check Execution Plan

Plan showed:

- Missing covering index
- Bookmark lookup on a large table
- Predicate on non-indexed column (inefficient filtering)

Step 3: Validate with DMVs

Using query stats DMV:

```
SELECT TOP 5 *
FROM sys.dm_exec_query_stats
ORDER BY total_worker_time DESC;
```

Confirmed the same query was consuming 65% of CPU.

Step 4: Check I/O

sys.dm_io_virtual_file_stats revealed high read stalls on the primary data file.

Root Cause

A heavy reporting query was performing table scans, causing:

- High CPU
- High I/O
- Long waits
- Throughput reduction

Resolution

1. Created a covering index:

```
CREATE INDEX IX_Sales_Covering
ON Sales (SaleDate, StoreID)
INCLUDE (Amount, Quantity);
```

2. Updated statistics
3. Scheduled the reporting query to run off-hours

Post-Fix Result

- CPU stabilized at 40–50%
- PAGEIOLATCH_SH waits dropped by 70%
- Application response time improved significantly

Comprehensive onboarding workbook for a junior SQL Server DBA focused on **Activity Monitor**, performance troubleshooting, DMVs, and practical exercises. We'll break it down in a structured way.

SQL Server DBA Onboarding Workbook: Activity Monitor & Performance Troubleshooting

1. Introduction to SQL Server Activity Monitor

Activity Monitor is a graphical tool in SQL Server Management Studio (SSMS) used to monitor SQL Server performance in real time. It helps DBAs identify resource bottlenecks, long-running queries, and overall server health.

1.1 Key Sections of Activity Monitor

| Section | Purpose | What to Look For |
|---------------------------------|---------------------------------------|----------------------------------|
| Overview | Quick glance at server load | CPU %, memory usage, I/O waits |
| Processes | Lists active connections and sessions | Blocking, suspended queries |
| Resource Waits | Shows wait types | Identify performance bottlenecks |
| Data File I/O | File-level read/write stats | Detect I/O contention |
| Recent Expensive Queries | Top CPU/IO consuming queries | Optimize problematic queries |

2. When and Why to Use Activity Monitor

Use Cases:

1. Identify CPU bottlenecks
2. Detect blocking or deadlocks
3. Monitor disk I/O and tempdb usage
4. Track long-running or expensive queries
5. Check overall server health during peak hours

Real-time Scenario Example:

- Users report slow report generation.
- Open Activity Monitor → Check **Recent Expensive Queries** → Identify query consuming high CPU → Drill down to execution plan → Optimize indexing or rewrite query.

3. Step-by-Step Activity Monitor Use for a Junior DBA

1. Open SSMS and connect to the SQL Server instance.
2. Right-click the server instance → **Activity Monitor**.
3. Review **Overview** to check CPU, Memory, I/O.
4. Switch to **Processes** → Sort by CPU/Duration → Identify long-running queries.
5. Check **Resource Waits** → Look for high waits like PAGEIOLATCH (I/O) or CXPACKET (parallelism).
6. Check **Data File I/O** → Identify database or files causing I/O contention.
7. Look at **Recent Expensive Queries** → Capture SQL text and execution plan for tuning.

4. Activity Monitor vs DMVs (Dynamic Management Views)

You can get **all Activity Monitor information using DMVs**, which is crucial for scripting monitoring and automating reports.

| Activity Monitor Section | DMV Equivalent | Sample Query |
|--------------------------|--|---|
| Overview | sys.dm_os_performance_counters, sys.dm_os_wait_stats, sys.dm_exec_requests | SELECT * FROM sys.dm_os_performance_counters WHERE counter_name LIKE '%CPU%' |
| Processes | sys.dm_exec_sessions, sys.dm_exec_requests | sql SELECT s.session_id, r.status, r.cpu_time, r.total_elapsed_time FROM sys.dm_exec_sessions s JOIN sys.dm_exec_requests r ON s.session_id = r.session_id |
| Resource Waits | sys.dm_os_wait_stats | SELECT wait_type, wait_time_ms FROM sys.dm_os_wait_stats ORDER BY wait_time_ms DESC |
| Data File I/O | sys.dm_io_virtual_file_stats | SELECT DB_NAME(database_id) AS DBName, file_id, num_of_reads, num_of_writes FROM sys.dm_io_virtual_file_stats(NULL,NULL) |
| Recent Expensive Queries | sys.dm_exec_query_stats, sys.dm_exec_sql_text | sql SELECT TOP 10 total_worker_time/execution_count AS AvgCPU, text FROM sys.dm_exec_query_stats CROSS APPLY sys.dm_exec_sql_text(sql_handle) ORDER BY AvgCPU DESC |

5. Sample Performance Troubleshooting Checklist

Step 1: Identify Symptoms

- High CPU, memory, or I/O
- Blocking/deadlocks
- Slow reports or jobs

Step 2: Gather Data

- Activity Monitor
- DMVs
- SQL Server Logs
- PerfMon counters (optional)

Step 3: Analyze

- Check top queries, waits, and sessions
- Identify resource bottlenecks (CPU, I/O, memory, locks)
- Review execution plans

Step 4: Take Action

- Kill or tune problematic queries
- Index optimization
- Update statistics
- Check hardware limits if needed

Step 5: Document

- Root cause
- Action taken
- Lessons learned

6. Real Case Study

Scenario: Users complain reports are slow every morning.

Step 1: Check Activity Monitor

- Recent Expensive Queries: One query consuming 90% CPU
- Resource Waits: High CXPACKET waits

Step 2: Drill Down

- Check execution plan → Query doing large table scan
- Check indexes → Missing non-clustered index

Step 3: Action

- Added a covering index
- Updated statistics

Step 4: Result

- Query runtime dropped from 120s → 15s
- CPU usage normalized

Root Cause: Missing index + parallelism causing CXPACKET waits

7. Lab Exercise for Junior DBAs

Objective: Practice using Activity Monitor and DMVs to troubleshoot performance.

Setup Test Data:

-- Create test database

```
CREATE DATABASE LabDB;
GO
USE LabDB;

-- Create test table with millions of rows
CREATE TABLE Orders (
    OrderID INT IDENTITY(1,1) PRIMARY KEY,
    CustomerID INT,
    OrderDate DATETIME,
    Amount DECIMAL(10,2)
);
-- Populate table with 1 million rows
INSERT INTO Orders (CustomerID, OrderDate, Amount)
SELECT TOP (1000000)
    ABS(CHECKSUM(NEWID()) % 1000) + 1,
    DATEADD(DAY, -ABS(CHECKSUM(NEWID()) % 1000), GETDATE()),
    CAST(RAND(CHECKSUM(NEWID())) * 1000 AS DECIMAL(10,2))
FROM sys.objects a CROSS JOIN sys.objects b;
```

-- Create another table

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName NVARCHAR(50)
);
```

Exercise Steps:

1. Open Activity Monitor → Check **Overview, Processes, Recent Expensive Queries**.
2. Run a query:
3. `SELECT CustomerID, SUM(Amount) Total`
4. `FROM Orders`
5. `GROUP BY CustomerID`
6. `ORDER BY Total DESC;`
7. Identify which query consumes most CPU using Activity Monitor.
8. Use DMVs to find top CPU queries:
9. `SELECT TOP 5 total_worker_time/execution_count AS AvgCPU, text`
10. `FROM sys.dm_exec_query_stats`
11. `CROSS APPLY sys.dm_exec_sql_text(sql_handle)`
12. `ORDER BY AvgCPU DESC;`
13. Optional: Create index on `Orders(CustomerID)` → Rerun query → Observe performance improvement.

8. Key Takeaways for Junior DBAs

- Activity Monitor is **great for real-time monitoring**, but DMVs are **better for automation and detailed analysis**.
- Always **correlate symptoms with data** before taking action.
- Document every change and result.
- Practice on **non-production environments** before touching live servers.
- Use lab exercises to simulate slow queries, blocking, and resource waits.

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