Module_4_UseCase_Wait Stats, Blocking & Concurrency (4 Hours)

Concepts

- Understanding waits vs bottlenecks
- Key waits:
  - CXPACKET
  - PAGEIOLATCH_*
  - LCK_M_*
  - SOS_SCHEDULER_YIELD
- Blocking vs deadlocks
- Deadlock graphs & root cause analysis
- Isolation levels
- Row versioning (RCSI, Snapshot)

Hands-on Labs

- Capture wait stats baselines
- Identify top waits
- Simulate blocking & deadlocks
- Analyze deadlock graphs
- Fix blocking using isolation tuning

Tools Used

- sys.dm_os_wait_stats
- sys.dm_exec_requests
- Extended Events

Create CSC Global Use Cases for DBA professionals. Also explain each line of code.

Below is a **CSC Global–grade, instructor-ready Module 4 pack** for

Wait Stats, Blocking & Concurrency (4 Hours) — written exactly for **senior DBA professionals**.

You'll get:

- Real **enterprise use cases**
- **Hands-on labs**
- **Every line of code explained**
- Clear **"what to say in class" DBA takeaways**

DAY 3 – Wait Stats, Blocking & Concurrency

Module 4: Wait Stats, Blocking & Concurrency (4 Hours)

USE CASE 1 – "System Is Slow… But CPU Is Idle"

(Understanding Waits vs Bottlenecks)

Business Context (CSC Global)

Application team says:

"SQL Server is slow, but CPU usage is only 20%."

DBA must prove:

- Is it CPU?
- IO?
- Locks?
- Parallelism?

Lab 1A – Capture baseline wait stats

```
SELECT
    wait_type,
    waiting_tasks_count,
    wait_time_ms / 1000.0 AS wait_time_sec,
    signal_wait_time_ms / 1000.0 AS signal_wait_sec
FROM sys.dm_os_wait_stats
ORDER BY wait_time_ms DESC;
```

Line-by-line explanation

- sys.dm_os_wait_stats

  Cumulative waits since last restart.

- wait_type

  What SQL Server was waiting for.

- waiting_tasks_count

  How many times this wait occurred.

- wait_time_ms

  Total wait time.

- signal_wait_time_ms

  Time waiting **after** getting resource (CPU scheduling delay).

Instructor talking point

"Waits tell us *why* SQL Server was idle."

Key DBA Learning

🔑 **High waits ≠ problem by default**

You must interpret *which* waits matter.

Filter out benign waits (very important)

```
SELECT
    wait_type,
    wait_time_ms / 1000.0 AS wait_time_sec
FROM sys.dm_os_wait_stats
WHERE wait_type NOT IN
(
 'CLR_SEMAPHORE','LAZYWRITER_SLEEP','RESOURCE_QUEUE',
 'SLEEP_TASK','SLEEP_SYSTEMTASK','WAITFOR',
 'HADR_FILESTREAM_IOMGR_IOCOMPLETION','CHECKPOINT_QUEUE',
 'XE_TIMER_EVENT','XE_DISPATCHER_WAIT'
)
ORDER BY wait_time_ms DESC;
```

Explanation

- Removes **idle/background waits**
- Focuses on **actionable waits**

USE CASE 2 – PAGEIOLATCH_* (IO Bottleneck)

Business Context

Compliance reports slow after data growth.

Lab 2A – Identify IO waits

```
SELECT
    wait_type,
    wait_time_ms / 1000.0 AS wait_time_sec,
    waiting_tasks_count
FROM sys.dm_os_wait_stats
WHERE wait_type LIKE 'PAGEIOLATCH%'
ORDER BY wait_time_ms DESC;
```

Explanation

- PAGEIOLATCH_*
  - Waiting for data pages from disk
- Indicates **storage or memory pressure**

What to say in class

"PAGEIOLATCH = SQL waiting for disk, not CPU."

Key DBA Learning

🔑 **Fix IO waits with indexing, memory, or storage — not CPU tuning**

USE CASE 3 – SOS_SCHEDULER_YIELD (CPU Pressure)

Business Context

CPU spikes during month-end processing.

Lab 3A – Identify CPU pressure

```
SELECT
    wait_type,
    wait_time_ms / 1000.0 AS wait_time_sec,
    waiting_tasks_count
FROM sys.dm_os_wait_stats
WHERE wait_type = 'SOS_SCHEDULER_YIELD';
```

Explanation

- SQL workers voluntarily yield CPU
- Indicates **CPU saturation**

Check scheduler queues

```
SELECT
    scheduler_id,
    runnable_tasks_count,
    active_workers_count
FROM sys.dm_os_schedulers
WHERE status = 'VISIBLE ONLINE'
ORDER BY runnable_tasks_count DESC;
```

Explanation

- runnable_tasks_count > 0
    - Threads waiting for CPU

Key DBA Learning

🔑 High SOS_SCHEDULER_YIELD = CPU bound workload

USE CASE 4 – CXPACKET / CXCONSUMER (Parallelism Issues)

Business Context

Queries are parallel but overall system slows down.

Lab 4A – Identify parallelism waits

```
SELECT
    wait_type,
    wait_time_ms / 1000.0 AS wait_time_sec
FROM sys.dm_os_wait_stats
WHERE wait_type IN ('CXPACKET','CXCONSUMER');
```

Explanation

- CXPACKET
  - Threads waiting on each other
- CXCONSUMER
  - Consumer threads waiting

Instructor talking point

"Parallelism is not free speed — it trades CPU for time."

Key DBA Learning

🔑 **Tune parallelism via better plans, not blindly lowering MAXDOP**

USE CASE 5 – Blocking (LCK_M_* waits)

Business Context

Users complain:

"Application freezes randomly."

Lab 5A – Identify blocking sessions

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

Explanation

- blocking_session_id
  - Session holding the lock
- LCK_M_*
  - Lock waits (Shared, Exclusive, Update)

Get blocking SQL text

```
SELECT
    r.session_id,
    t.text
FROM sys.dm_exec_requests r
CROSS APPLY sys.dm_exec_sql_text(r.sql_handle) t
WHERE r.session_id IN (SELECT blocking_session_id
                       FROM sys.dm_exec_requests
                       WHERE blocking_session_id <> 0);
```

Explanation

- CROSS APPLY
  - Fetches SQL text for each blocking request

Key DBA Learning

🔑 Blocking is not bad locking — it's long transactions

USE CASE 6 – Deadlocks (LCK_M_* with victim)

Business Context

Application throws:

"Transaction (Process ID ...) was deadlocked."

Lab 6A – Capture deadlocks using Extended Events

```
CREATE EVENT SESSION Deadlock_Monitor
ON SERVER
ADD EVENT sqlserver.xml_deadlock_report
ADD TARGET package0.ring_buffer;
GO

ALTER EVENT SESSION Deadlock_Monitor
ON SERVER STATE = START;
```

Explanation

- xml_deadlock_report
  - Captures deadlock graph
- ring_buffer
  - Stores recent events in memory

Read deadlock graph

```
SELECT
    CAST(event_data AS XML) AS deadlock_graph
FROM sys.fn_xe_file_target_read_file
(
    NULL, NULL, NULL, NULL
);
```

Instructor talking point

"Deadlocks are logic problems, not SQL Server bugs."

Key DBA Learning

🔑 **Deadlocks require query or transaction design fixes**

USE CASE 7 – Isolation Levels & Blocking Reduction

Business Context

Reports block OLTP inserts.

Lab 7A – Demonstrate blocking (Session 1)

```
BEGIN TRAN;
UPDATE dbo.Entities
SET client_name = client_name
WHERE entity_id = 1;
-- Do not commit yet
```

Session 2 – Blocked read

```
SELECT *
FROM dbo.Entities
WHERE entity_id = 1;
```

Fix using READ COMMITTED SNAPSHOT (RCSI)

```
ALTER DATABASE CURRENT
SET READ_COMMITTED_SNAPSHOT ON;
```

Explanation

- Readers use **row versioning**
- Writers no longer block readers

Key DBA Learning

🔑 **RCSI fixes read/write blocking without code changes**

USE CASE 8 – Snapshot Isolation

```
ALTER DATABASE CURRENT SET ALLOW_SNAPSHOT_ISOLATION ON;
```

Explanation

- Enables full snapshot isolation
- Readers see transactionally consistent view

FINAL CSC DBA TAKEAWAYS (Module 4)

After this module, DBAs must know:

1️⃣ Wait stats explain **why SQL Server is slow**

2️⃣ PAGEIOLATCH = IO, SOS_SCHEDULER_YIELD = CPU

3️⃣ CXPACKET ≠ always bad parallelism

4️⃣ Blocking = long transactions, not locks

5️⃣ Deadlocks are application logic issues

6️⃣ Isolation tuning solves concurrency pain elegantly

Interview-Ready One-Liners

- "Wait stats tell me *why* SQL Server waited."
- "Blocking is a symptom, not the disease."
- "Deadlocks are circular waits — not resource shortages."
- "RCSI is the safest concurrency improvement."

```
/*============================================================
=================
 CSC Global - Full Concurrency Demo Pack
 Topic: Blocking → Deadlock → RCSI Fix + Deadlock Graph
Walkthrough
 Audience: Senior DBAs

 Prereqs:
   - Run in a LAB database (NOT production)
   - Two query windows (Session A and Session B) in SSMS
   - Permissions:
       * ALTER DATABASE (for RCSI)
       * CREATE EVENT SESSION (for deadlock capture) -
typically sysadmin

 Uses your existing tables:
   dbo.Entities, dbo.ComplianceFilings

 Safety:
   - Uses small, targeted updates (entity_id 1 and 2)
```

```
================================================================
================*/

-----------------------------------------------------------------
-----------------
-- PART 0) Setup (Run ONCE by Instructor)
-----------------------------------------------------------------
-----------------
USE CSC_DBA_Demo;  -- change if needed
GO

-- Make sure XACT_ABORT is ON for safety in labs (auto-
rollback on runtime errors)
SET XACT_ABORT ON;
GO

-- (Optional) Confirm rows exist
SELECT TOP (5) entity_id, client_name, country_code FROM
dbo.Entities ORDER BY entity_id;
GO

-----------------------------------------------------------------
-----------------
-- PART 1) BLOCKING DEMO (Read blocked by Write) - Two
sessions
-- Goal: Show blocking chain and how to observe it using DMVs.
-----------------------------------------------------------------
-----------------

/*--------------------------
 SESSION A (Window 1)
--------------------------*/
-- Run this in Session A:
BEGIN TRAN;

-- Take an X lock on a single row in Entities
UPDATE dbo.Entities
SET client_name = client_name  -- no-op update still takes
locks
WHERE entity_id = 1;

-- Keep transaction open to hold locks (DO NOT COMMIT YET)
-- Instructor: leave this window as-is for 30-60 seconds.


/*--------------------------
 SESSION B (Window 2)
--------------------------*/
-- Run this in Session B (it will block under READ COMMITTED):
SET LOCK_TIMEOUT 10000; -- 10 seconds so it doesn't hang
forever in class
```

```
SELECT entity_id, client_name, country_code
FROM dbo.Entities
WHERE entity_id = 1;
-- Expected: waits / times out while Session A holds the X
lock


/*--------------------------
 OBSERVE (Window 2 or 3)
--------------------------*/
-- While Session B is blocked, run:
SELECT
    r.session_id,
    r.blocking_session_id,
    r.status,
    r.command,
    r.wait_type,
    r.wait_time,
    r.last_wait_type,
    r.cpu_time,
    r.total_elapsed_time
FROM sys.dm_exec_requests r
WHERE r.session_id IN (@@SPID)  -- in this window
   OR r.blocking_session_id <> 0
ORDER BY r.total_elapsed_time DESC;

-- Show lock details (who holds what)
SELECT
    tl.request_session_id AS session_id,
    tl.resource_type,
    tl.resource_database_id,
    tl.resource_associated_entity_id,
    tl.request_mode,
    tl.request_status
FROM sys.dm_tran_locks tl
WHERE tl.resource_database_id = DB_ID()
ORDER BY tl.request_session_id, tl.resource_type;

-- Get SQL text for blocked & blocker sessions
;WITH blockers AS
(
    SELECT DISTINCT blocking_session_id AS sid
    FROM sys.dm_exec_requests
    WHERE blocking_session_id <> 0
),
blocked AS
(
    SELECT DISTINCT session_id AS sid
    FROM sys.dm_exec_requests
    WHERE blocking_session_id <> 0
```

```
)
SELECT
    s.sid AS session_id,
    t.text
FROM (SELECT sid FROM blockers UNION ALL SELECT sid FROM
blocked) s
JOIN sys.dm_exec_requests r ON r.session_id = s.sid
CROSS APPLY sys.dm_exec_sql_text(r.sql_handle) t;


-- Cleanup for blocking demo:
-- SESSION A: COMMIT to release locks
-- (Run in Session A)
COMMIT;
GO

-----------------------------------------------------------------
-----------------
-- PART 2) DEADLOCK DEMO - Two sessions + Extended Events
capture
-- Goal: Create a real deadlock and capture the deadlock
graph.
-----------------------------------------------------------------
-----------------

/*-----------------------------------------------------------------
--------------
  2.1 Create an Extended Events session to capture deadlocks
(Run ONCE)
-----------------------------------------------------------------
------------*/
IF EXISTS (SELECT 1 FROM sys.server_event_sessions WHERE name
= 'Deadlock_Monitor_CSC')
BEGIN
    DROP EVENT SESSION Deadlock_Monitor_CSC ON SERVER;
END
GO

CREATE EVENT SESSION Deadlock_Monitor_CSC
ON SERVER
ADD EVENT sqlserver.xml_deadlock_report
ADD TARGET package0.ring_buffer
WITH (MAX_MEMORY = 4096 KB, EVENT_RETENTION_MODE =
ALLOW_SINGLE_EVENT_LOSS);
GO

ALTER EVENT SESSION Deadlock_Monitor_CSC ON SERVER STATE =
START;
GO
```

```
/*-------------------------------------------------------------
-------------
  2.2 Deadlock scenario
  Strategy:
    - Session A locks Entities(1) then tries Entities(2)
    - Session B locks Entities(2) then tries Entities(1)
  This creates a cycle: A waits for B, B waits for A.
-------------------------------------------------------------
------------*/

/*--------------------------
 SESSION A (Window 1)
--------------------------*/
-- Run in Session A:
SET DEADLOCK_PRIORITY NORMAL;
BEGIN TRAN;

UPDATE dbo.Entities
SET client_name = client_name
WHERE entity_id = 1;

-- Pause so Session B can lock entity_id = 2
WAITFOR DELAY '00:00:05';

-- Now try to lock the row Session B has
UPDATE dbo.Entities
SET client_name = client_name
WHERE entity_id = 2;

COMMIT;  -- likely won't reach if chosen as deadlock victim
GO


/*--------------------------
 SESSION B (Window 2)
--------------------------*/
-- Run in Session B quickly after Session A starts:
SET DEADLOCK_PRIORITY NORMAL;
BEGIN TRAN;

UPDATE dbo.Entities
SET client_name = client_name
WHERE entity_id = 2;

-- Pause so Session A is holding entity_id = 1
WAITFOR DELAY '00:00:05';

-- Now try to lock the row Session A has
UPDATE dbo.Entities
SET client_name = client_name
WHERE entity_id = 1;
```

```
COMMIT; -- one session will fail with deadlock error 1205
GO

/*-------------------------------------------------------------
--------------
  Expected Output:
   One session gets:
     Msg 1205, Level 13, State ...
     Transaction (Process ID ...) was deadlocked on lock
resources ...
     and has been chosen as the deadlock victim.
-------------------------------------------------------------
------------*/

-------------------------------------------------------------
-----------------
-- PART 3) READ THE DEADLOCK GRAPH (Instructor)
-- Goal: Pull XML from ring_buffer and explain it.
-------------------------------------------------------------
-----------------
;WITH x AS
(
    SELECT CAST(t.target_data AS XML) AS target_xml
    FROM sys.dm_xe_sessions s
    JOIN sys.dm_xe_session_targets t
      ON s.address = t.event_session_address
    WHERE s.name = 'Deadlock_Monitor_CSC'
      AND t.target_name = 'ring_buffer'
),
d AS
(
    SELECT
        n.value('(event/@timestamp)[1]', 'datetime2(3)') AS
[utc_timestamp],
        n.query('.') AS event_xml
    FROM x
    CROSS APPLY
x.target_xml.nodes('//RingBufferTarget/event[@name="xml_deadlo
ck_report"]') AS q(n)
)
SELECT TOP (5)
    [utc_timestamp],
    event_xml.query('(event/data/value/deadlock)[1]') AS
deadlock_graph_xml
FROM d
ORDER BY [utc_timestamp] DESC;
GO

-- Stop session after demo (optional)
```

```
ALTER EVENT SESSION Deadlock_Monitor_CSC ON SERVER STATE =
STOP;
GO
```

Deadlock graph walkthrough with annotations (how to teach it)

When you open the XML (from deadlock_graph_xml) in SSMS, you'll see this structure:

1) <deadlock>

Top-level container: "Here is the cycle."

2) <victim-list>

Look for something like:

```
<victim-list>
  <victimProcess id="process123" />
</victim-list>
```

Meaning: SQL Server chose this process to kill to break the cycle.
DBA key learning: Deadlock is solved by **victim selection**, not by "waiting longer."

3) <process-list>

You'll see 2 (or more) <process> nodes (one per session). Key attributes to point out:

- id="process123"

  Internal id referenced by victim-list.

- spid="57"

  The SQL Server session id (SPID).

- isolationlevel="read committed (2)" (or similar)

  Helps you connect deadlocks to isolation behavior.

- waitresource="KEY: ..." or OBJECT: ...

  What it is waiting on.

- <inputbuf>

  The exact statement text that was executing at deadlock time.

Teaching script:

“Process-list answers: *Who*, *what SQL*, *what isolation*, *what they waited for*.”

4) <resource-list>

This is the most important for root cause. You’ll see resources like:

- <keylock> (common) → row/key-level locks
- <pagelock> → page locks (hot pages, escalation)
- <objectlock> → table-level locks

Inside each resource:

- <owner-list> → who owns the lock
- <waiter-list> → who is waiting for it

Example idea (simplified):

```
<keylock ...>
  <owner-list>
    <owner id="processA" mode="X"/>
  </owner-list>
  <waiter-list>
    <waiter id="processB" mode="X" requestType="wait"/>
  </waiter-list>
</keylock>
```

Teaching script:

“Resource-list answers: *What was locked*, *who held it*, *who waited*, and *what mode*.”

5) Lock modes you’ll commonly see

- S = Shared (read)
- X = Exclusive (write)
- U = Update (common in some patterns)
- IS/IX = intent locks (higher-level coordination)

DBA key learning: Deadlocks happen because two sessions acquire locks in **different orders**.

RCSI Fix Demo (Read/Write blocking reduction)

Deadlocks are usually **write/write** ordering problems; **RCSI primarily fixes read/write blocking** (reports vs OLTP).

So after the deadlock demo, show a *very practical CSC scenario*:

“Morning report reads millions of rows and blocks inserts/updates.”

RCSI setup (run once, requires exclusive access)

```
-- Run when no one else is connected to the DB (training lab
is perfect)


ALTER DATABASE CURRENT SET READ_COMMITTED_SNAPSHOT ON WITH
ROLLBACK IMMEDIATE;
GO
```

Re-run the blocking demo to show improvement

Session A (Window 1)

```
BEGIN TRAN;
UPDATE dbo.Entities
SET client_name = client_name
WHERE entity_id = 1;
-- keep open
```

Session B (Window 2)

```
-- Under RCSI, this SELECT should NOT be blocked

SELECT entity_id, client_name, country_code
FROM dbo.Entities
WHERE entity_id = 1;
```

What changed?

- Under normal Read Committed: reader waits on writer lock.
- Under RCSI: reader gets a **versioned row** from tempdb version store.

DBA key learning (say this clearly):

- ✅ RCSI is a **safe concurrency upgrade** for OLTP + reporting mixed workloads.
- ✅ It reduces “app freeze” complaints caused by reader/writer blocking.
- ⚠️ It increases tempdb usage (version store), so monitor tempdb.

Production-grade takeaways for CSC DBAs

Blocking

- Usually caused by **long transactions**, not “bad locks.”
- Fix by: shorten transactions, index to reduce lock footprint, tune isolation.

Deadlocks

- Usually caused by **inconsistent object access order** across code paths.

- Fix by: consistent ordering, proper indexing, reduce lock duration, retry logic.

RCSI

- Best for reducing **read/write blocking** (reports vs OLTP).
- Not a magic deadlock cure for write/write cycles.
