

Analyze Slow & Problematic Queries using Extended Events (XE) _ CSC DBA Use Case

Below is a **complete, production-grade Extended Events (XE) use case**, written exactly the way a **CSC DBA would design, run, analyze, and explain it in a live incident**.

Business Scenario (Real CSC Context)

Application: CSC Compliance Management System

Issue reported:

- Users complain: “*System is slow after deployment*”
- No blocking
- No deadlocks
- CPU spikes intermittently
- Query Store shows *many queries*, but **root cause unclear**

 DBA decision: **Use Extended Events to capture real execution behavior**

Why Extended Events (XE)?

Tool	Limitation
Query Store	Aggregated data
Execution Plans	Point-in-time
Profiler	Deprecated, heavy
Extended Events	Low overhead, real-time, surgical

 XE = Production-safe flight recorder

XE USE CASE DESIGN (What we want to capture)

We want to capture:

- Slow queries (> 2 seconds)
- High CPU queries
- Parameter sniffing cases
- Implicit conversions
- Blocking & waits

STEP 1: Create XE Session (CSC-Safe)

```
IF EXISTS (
    SELECT 1
    FROM sys.server_event_sessions
    WHERE name = 'CSC_M7_Query_Analysis_XE'
)
    DROP EVENT SESSION CSC_M7_Query_Analysis_XE ON SERVER;
GO

CREATE EVENT SESSION CSC_M7_Query_Analysis_XE
ON SERVER
ADD EVENT sqlserver.rpc_completed
(
    ACTION
    (
        sqlserver.sql_text,
        sqlserver.database_name,
        sqlserver.client_app_name,
        sqlserver.username,
        sqlserver.session_id
    )
    WHERE
    (
        duration > 2000000 -- > 2 seconds (microseconds)
        OR cpu_time > 2000000
    )
),
ADD EVENT sqlserver.sql_batch_completed
(
    ACTION
    (
        sqlserver.sql_text,
        sqlserver.database_name,
        sqlserver.client_app_name,
        sqlserver.session_id
    )
    WHERE duration > 2000000
),
ADD EVENT sqlserver.wait_info
(
    ACTION
    (
        sqlserver.sql_text,
        sqlserver.session_id
    )
    WHERE wait_time > 1000000 -- > 1 second
)
ADD TARGET package0.event_file
(
    SET filename = 'C:\XE\CSC_M7_Query_Analysis.xel',
    max_file_size = 50,
    max_rollover_files = 5
);
GO
```

🧠 DBA Learning (Design Phase)

Choice	Why
duration filter	Avoid noise
cpu_time filter	Catch CPU hogs
wait_info	Root cause (IO, CPU, locks)
event_file	Persistent, low overhead

👉 Never run XE without filters

▶ STEP 2: Start XE Session

```
ALTER EVENT SESSION CSC_M7_Query_Analysis_XE
ON SERVER
STATE = START;
GO
```

✍ STEP 3: Simulate Real CSC Problem Query

```
USE CSC_DBA_Demo;
GO

DECLARE @ClientName NVARCHAR(200) = 'BigClient';

SELECT
    e.client_name,
    cf.filing_type,
    COUNT_BIG(*) AS overdue_count
FROM dbo.ComplianceFilings cf
JOIN dbo.Entities e ON e.entity_id = cf.entity_id
WHERE
    e.client_name = @ClientName
    AND cf.filing_status = 'OVERDUE'
GROUP BY
    e.client_name,
    cf.filing_type
ORDER BY overdue_count DESC;
GO
```

👉 Assume:

- Query runs slow for big clients
 - Fast for small clients
 - **Classic parameter sniffing**
-

🕒 STEP 4: Stop XE Session (After Capture)

```
ALTER EVENT SESSION CSC_M7_Query_Analysis_XE
ON SERVER
STATE = STOP;
GO
```

🔍 STEP 5: Read XE Data (THE MOST IMPORTANT PART)

```
SELECT
    event_data.value('(event/@name)[1]', 'varchar(50)') AS event_name,
```

```

    event_data.value('(event/data[@name="duration"]/value)[1]', 'bigint') /
1000 AS duration_ms,
    event_data.value('(event/data[@name="cpu_time"]/value)[1]', 'bigint') /
1000 AS cpu_ms,
    event_data.value('(event/data[@name="wait_time"]/value)[1]', 'bigint') /
1000 AS wait_ms,
    event_data.value('(event/action[@name="session_id"]/value)[1]', 'int')
AS session_id,
    event_data.value('(event/action[@name="database_name"]/value)[1]',
'sysname') AS database_name,
    event_data.value('(event/action[@name="sql_text"]/value)[1]',
'nvarchar(max)') AS sql_text
FROM
(
    SELECT CAST(event_data AS XML) AS event_data
    FROM sys.fn_xe_file_target_read_file
    (
        'C:\XE\CSC_M7_Query_Analysis*.xel',
        NULL,
        NULL,
        NULL
    )
) AS x
ORDER BY duration_ms DESC;
GO

```

OUTPUT INTERPRETATION (Teach This Clearly)

Column	Meaning
event_name	What happened
duration_ms	End-to-end execution time
cpu_ms	CPU consumed
wait_ms	Time waiting (not CPU)
sql_text	Actual query

CSC DBA ANALYSIS (What XE Reveals)

Observation 1

- Same query text
- Different durations
- CPU very high for one execution

Parameter sniffing confirmed

Observation 2

- `wait_info` shows:
 - `PAGEIOLATCH_SH`
 - `SOS_SCHEDULER_YIELD`

👉 Mixed IO + CPU pressure

🛠 STEP 6: Fix the Issue (DBA Actions)

Option 1: RECOMPILE (Quick Fix)

```
ALTER PROCEDURE dbo.usp_ClientOverdue
    @ClientName NVARCHAR(200)
AS
BEGIN
    SELECT ...
    OPTION (RECOMPILE);
END
```

Option 2: Query Store Forced Plan (Safer)

```
EXEC sys.sp_query_store_force_plan
    @query_id = 123,
    @plan_id = 456;
```

Option 3: Index for Skew

```
CREATE INDEX IX_Entities_ClientName
ON dbo.Entities(client_name)
INCLUDE (entity_id);
```

⌚ STEP 7: Validate Fix using XE (Re-run)

1. Start XE
2. Run query again
3. Stop XE
4. Compare duration & CPU

Expected:

- CPU drops
- Duration stable

- No long waits
-

CSC DBA CHECKLIST – WHEN TO USE XE

Scenario	XE Useful?
Intermittent slowness	✓
Parameter sniffing	✓
CPU spikes	✓
Blocking	✓
Plan regression proof	✓
Long-running query	✓

INTERVIEW-READY ONE-LINERS

- “XE is my last-mile diagnostic tool when Query Store is not enough.”
 - “Query Store tells me what changed, XE tells me why.”
 - “Never troubleshoot performance blind — capture first.”
-

WHAT CSC DBAs LEARN FROM THIS LAB

- ✓ Design safe XE sessions
 - ✓ Capture only what matters
 - ✓ Analyze real production behavior
 - ✓ Correlate CPU, waits, and SQL
 - ✓ Fix issues with confidence
-