

Debug Performance Issues Caused by Implicit Data Conversion (SQL Server 2022)

It includes: **story, setup, repro, how to detect (DMVs + XE), how to read outputs, and safe fixes.**

Business Story (CSC Compliance App)

After a deployment, users report:

- “Overdue filings screen is slow”
- CPU spikes
- Reads increase
- No blocking

DBA suspects: **Implicit conversion causing index seeks to become scans**

0) What is Implicit Conversion (in DBA words)

Implicit conversion happens when SQL Server **converts one side of a predicate** to match the other datatype.

Example:

- Column is NVARCHAR(20)
- Parameter passed is INT
- SQL Server may do: CONVERT_IMPLICIT(nvarchar(20), @param) **or** convert the column
- If the **column is converted**, the index becomes unusable → SCAN.

 Key rule:

If SQL Server converts the indexed column in WHERE/JOIN, you lose SARGability.

1) Lab Setup (CSC Demo Schema)

```

Run in CSC_DBA_Demo
USE CSC_DBA_Demo;
GO

-- Safety cleanup
IF OBJECT_ID('dbo.Filings_ConversionDemo', 'U') IS NOT NULL
    DROP TABLE dbo.Filings_ConversionDemo;
GO

CREATE TABLE dbo.Filings_ConversionDemo
(
    filing_id      BIGINT IDENTITY(1,1) PRIMARY KEY,
    entity_code    NVARCHAR(20) NOT NULL,           -- stored
    as string (bad design but common in legacy apps)
    filing_status  NVARCHAR(20) NOT NULL,
    due_date       DATE NOT NULL,
    payload        CHAR(200) NOT NULL DEFAULT REPLICATE('X',
200)
);
GO

-- Index that SHOULD make lookups fast
CREATE INDEX IX_Filings_EntityCode_Status
ON dbo.Filings_ConversionDemo(entity_code, filing_status)
INCLUDE(due_date);
GO

-- Load data (large enough to show difference)
;WITH n AS
(
    SELECT TOP (3000000)
        ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) AS rn
    FROM sys.all_objects a
    CROSS JOIN sys.all_objects b
)
INSERT dbo.Filings_ConversionDemo(entity_code, filing_status,
due_date)
SELECT
    CAST((rn % 50000) + 1 AS NVARCHAR(20)) AS entity_code, --
entity_code like '1','2'...'50000'
    CASE WHEN rn % 10 = 0 THEN N'OVERDUE' ELSE N'OK' END,
    DATEADD(DAY, -(rn % 365), CAST(GETDATE() AS DATE));
GO

```

DBA learning

- This mimics CSC-style legacy columns like entity_code stored as string.
-

2) Repro the Problem (Implicit Conversion)

Case A: Good query (correct datatype = fast)

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

DECLARE @EntityCodeGood NVARCHAR(20) = N'25000';

SELECT COUNT_BIG(*)
FROM dbo.Filings_ConversionDemo
WHERE entity_code = @EntityCodeGood
    AND filing_status = N'OVERDUE';
GO

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

 Expected:

- Index Seek
- Low logical reads
- Fast

Case B: Bad query (INT parameter = implicit conversion = slow)

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

DECLARE @EntityCodeBad INT = 25000; -- wrong datatype

SELECT COUNT_BIG(*)
FROM dbo.Filings_ConversionDemo
WHERE entity_code = @EntityCodeBad -- <-- implicit conversion
    AND filing_status = N'OVERDUE';
GO

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

 Typical outcome:

- **Index Scan**
 - High reads
 - CPU spikes
-

3) Prove It in the Execution Plan (What to show in class)

In the **bad query plan**, show:

- Warning: “**Type conversion in expression may affect cardinality**”
- Plan operator will show: **CONVERT_IMPLICIT**

DBA learning

- This is the fastest “visual proof” to show devs.
-

4) Detect Implicit Conversion Systematically (Query Store / Plans DMV)

A) Find cached plans with **CONVERT_IMPLICIT**

```
SELECT TOP (30)
    DB_NAME(st.dbid) AS db_name,
    qs.total_elapsed_time / 1000 AS total_elapsed_ms,
    qs.total_worker_time / 1000 AS total_cpu_ms,
    qs.execution_count,
    SUBSTRING(st.text, (qs.statement_start_offset/2)+1,
        ((CASE qs.statement_end_offset WHEN -1 THEN
        DATALENGTH(st.text) ELSE qs.statement_end_offset END
        - qs.statement_start_offset)/2) + 1) AS stmt_text,
    qp.query_plan
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) st
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) qp
WHERE qp.query_plan LIKE '%CONVERT_IMPLICIT%'
ORDER BY qs.total_worker_time DESC;
GO
```

Output meaning

- Lists your worst CPU/time offenders **where conversion exists in plan**.

DBA learning

- In production, this query quickly surfaces “silent killers”.
-

5) Production-Safe Capture with Extended Events (Best CSC Practice)

Create XE session to capture implicit conversions

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

CREATE EVENT SESSION CSC_ImplicitConversion_XE
ON SERVER
ADD EVENT sqlserver.plan_affecting_convert
(
    ACTION
    (
        sqlserver.sql_text,
        sqlserver.database_name,
        sqlserver.client_app_name,
        sqlserver.session_id,
        sqlserver.username
    )
)
ADD TARGET package0.ring_buffer;
GO

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

DBA learning

- `plan_affecting_convert` is **the best XE signal** for this problem.
-

Read XE output (ring buffer)

```
;WITH x AS
(
    SELECT CAST(t.target_data AS XML) AS target_data
```

```

        FROM sys.dm_xe_sessions s
        JOIN sys.dm_xe_session_targets t
          ON s.address = t.event_session_address
     WHERE s.name = 'CSC_ImplicitConversion_XE'
       AND t.target_name = 'ring_buffer'
)
SELECT
    n.value('(event/@timestamp)[1]', 'datetime2') AS
event_time,
    n.value('(event/action[@name="database_name"]/value)[1]',
'sysname') AS database_name,
    n.value('(event/action[@name="client_app_name"]/value)[1]',
'nvarchar(256)') AS client_app,
    n.value('(event/action[@name="sql_text"]/value)[1]',
'nvarchar(max)') AS sql_text,
    n.value('(event/data[@name="convert_issue"]/text)[1]',
'nvarchar(4000)') AS convert_issue
FROM x
CROSS APPLY target_data.nodes('//RingBufferTarget/event') AS
q(n)
WHERE n.value('(event/@name)[1]', 'sysname') =
'plan_affecting_convert'
ORDER BY event_time DESC;
GO

```

Output meaning

- Shows which query caused a **plan-affecting conversion**
 - Provides description of the conversion issue
-

6) Fix Options (DBA-approved, real production)

Fix 1: Correct parameter datatype (Best)

If column is NVARCHAR(20) then parameter must be NVARCHAR(20).

In stored procedure:

```

CREATE OR ALTER PROCEDURE dbo.usp_GetOverdueByEntity
    @EntityCode NVARCHAR(20)
AS

```

```
BEGIN
    SELECT COUNT_BIG(*)
    FROM dbo.Filings_ConversionDemo
    WHERE entity_code = @EntityCode
        AND filing_status = N'OVERDUE';
END
GO
```

✓ Fix 2: Explicit conversion on the parameter side (Safe)

```
DECLARE @EntityCodeBad INT = 25000;

SELECT COUNT_BIG(*)
FROM dbo.Filings_ConversionDemo
WHERE entity_code = CONVERT(NVARCHAR(20), @EntityCodeBad)
    AND filing_status = N'OVERDUE';
GO
```

Why this works

- Converts the **parameter**, not the column
 - Keeps index seek usable
-

✓ Fix 3: Schema fix (Best long-term)

If entity_code is numeric, make it numeric:

- Add new column entity_code_int INT
 - Backfill
 - Add index
 - Change app gradually
-

⚠ Fix 4: Computed column approach (Migration-friendly)

```
ALTER TABLE dbo.Filings_ConversionDemo
ADD entity_code_int AS TRY_CONVERT(INT, entity_code)
PERSISTED;
GO

CREATE INDEX IX_Filings_EntityCodeInt
ON dbo.Filings_ConversionDemo(entity_code_int, filing_status);
GO
```

7) Post-Fix Validation Checklist (CSC DBA)

- Run STATISTICS IO/TIME again
 - Confirm Index Seek
 - Check Query Store duration trend improves
 - Confirm XE stops capturing plan_affecting_convert for this query
 - Monitor CPU + logical reads drop
-

8) What CSC DBAs Should Say in Incident Call (Bridge Call Script)

- “The slowdown is due to **implicit conversion** on an indexed predicate.”
 - “This made the optimizer choose an **index scan** instead of seek.”
 - “We confirmed via **CONVERT_IMPLICIT** in **plan** and XE event **plan_affecting_convert**.”
 - “Fix is to match datatypes / cast parameter, restoring seek.”
-