#FABCONSQLCON2026

FABCON
Microsoft Fabric
COMMUNITY CONFERENCE

SQLCON
Microsoft SQL
COMMUNITY CONFERENCE

ATLANTA MARCH 16 - 20, 2026

Bulletproof Your Dataflows:

Error Handling and Data Quality in Fabric

Cristian Angyal

About me

- **Electrical Engineering background**
- **Former Project Manager** *(PMP®, PMI-ACP®)*
- **Microsoft Certified Professional**

  (DP-600, DP-700, PL-300, MCT, MCSA, MCSE, MOS Master)
- **Romania PBI and Modern Excel UG Founder**

  https://www.meetup.com/*romaniapug*
- **Microsoft MVP (*Excel & Power BI*)**
- **Husband** *(25y+)* and **proud father**
- Love to ski, travel and learn

Cristian Angyal

in/cristian-angyal
https://linktr.ee/cristiangyal

Session Plan

1. Dataflows Gen2 - Short Overview
2. Error Types in Power Query M
3. Handling Errors: try/otherwise, try/catch, structured patterns
4. Cell-Level Errors - Detect, Isolate, Route
5. DEMO - Lakehouse Error Handling in Fabric
6. Centralized Error Log Table
7. Best Practices & Wrap-up

What are Dataflows Gen2?

Data Transformation & preparation engine in Microsoft Fabric

Extract

Connect to 150+ data sources:
Lakehouse, Warehouse, SQL, APIs, Files, SharePoint, and more

Transform

Power Query M: full ETL capabilities
UI-driven or advanced M code editing

Load

Write directly to Fabric Lakehouse,
Data Warehouse, or other destinations

Refresh

Scheduled, on-demand, or triggered
via Fabric Pipeline orchestration

Key differences vs. Power BI Dataflows (Gen1)

- ✅ Native Lakehouse & Warehouse output destinations
- ✅ Staging layer automatic or configurable
- ✅ Fabric Pipeline integration for orchestration and error monitoring

How Power Query M Works

Every M expression must produce exactly one of two outcomes:

✅ A Value

- Scalar: text, number, date, logical
- Structured: list, record, table
- Binary, function, type

❌ An Error

- Expression could NOT produce a value
- Always has 3 components: Reason, Message, Detail
- Can propagate or be caught and handled

🎯 **Today's focus: understanding, catching, routing, and logging errors**

Anatomy of a Power Query Error

Every error in M is a record with 3 fields:

Reason

The error category. Defaults to "Expression.Error" if not specified.
Examples: DataFormat.Error, Expression.Error, DataSource.Error

Message

Human-readable explanation of what went wrong.
Can be a plain string or structured with Message.Format + parameters.

Detail

Additional context. Can be any M value — a record, a list, a table.
This is where you put your custom diagnostic data.

```
// Raising a custom error
error [
    Reason = "DataFormat.Error",
    Message = "Invalid date format in source file",
    Detail = [SourceFile = "sales_2024.csv", ColumnName = "OrderDate", Value = "31/02/2024"]
]
```

Two Types of Errors in Power Query

⛔ STEP-LEVEL Errors

- **Prevents the entire query from loading**
- Data Source Errors - file/DB not found
- Step Formula Errors - syntax, type mismatch
- Precedent Query Errors - upstream failure
- Formula Firewall Errors - privacy level conflicts

In Fabric: these block the Dataflow run and surface as Pipeline activity failures

⚠️ CELL-LEVEL Errors (Value Errors)

- **Does NOT stop the query from loading**
- Data Type Conversion Errors - most common
- Operation Errors - e.g. divide by zero, null ops
- Custom validation errors you raise explicitly

PQ silently replaces errors with null
→ silent data quality issues!
***This is the focus of today's demo**.*

Dealing with Errors in M

Five strategies — choose based on your data quality requirements:

⚠ **Ignore**	*Let PQ silently replace errors with null.* *⚠ Only acceptable if null is a valid value.*
🗑 **Remove**	*Use Table.RemoveRowsWithErrors to delete rows with errors.* *Good when error rows have no salvageable data.*
🔄 **Replace**	*Use Table.ReplaceErrorValues with a default/fallback value.* *Good for non-critical fields with known defaults.*
🎯 **Keep**	*Use Table.SelectRowsWithErrors to isolate error rows.* *Great for building a dedicated error inspection table.*
🛡 **Leverage**	*Use try/otherwise or try/catch to handle errors per-cell and extract structured error data. Most powerful pattern.*

`try … otherwise` - Basic Error Catching

Evaluates an expression; returns the fallback value if it raises an error:

```
// Basic try/otherwise
let
    Result = try Number.FromText("abc") otherwise -1,
    // Returns -1 because "abc" cannot be converted to a number

// In a custom column (very common use case)
    SafeAmount = try Number.FromText([RawAmount]) otherwise null,

// With a meaningful fallback
    SafeDate = try Date.FromText([RawDate]) otherwise #date(1900,1,1)
in
    Result
```

✅ When to use try/otherwise

- You want a fallback value on error
- Errors are expected and acceptable
- Quick, simple, readable code

⚠️ Limitations

- Error information is lost — you only get the fallback
- Cannot log what went wrong or where
- Not suitable for audit/monitoring patterns

`try … catch` - Full Error Record Access

try/catch gives you access to the full structured error record:

```
// try/catch – access the error record
let  SafeConvert = (val as text) =>
        try Number.FromText(val)
        catch (e) => [
                HasError = true,  ErrorReason = e[Reason],
                ErrorMsg = e[Message],  ErrorDetail = e[Detail]?,
                OriginalVal = val ]
in  SafeConvert("abc")
// Returns: [HasError=true, ErrorReason="Expression.Error", OriginalVal="abc"]
```

e[Reason]	e[Message]	e[Detail]?
Error category/type. E.g. "DataFormat.Error"	Human-readable description. Use in error log table.	Structured context data. ? avoids error if null.

`try` - The Error Record Pattern

try alone returns a record with HasError + Value or Error:

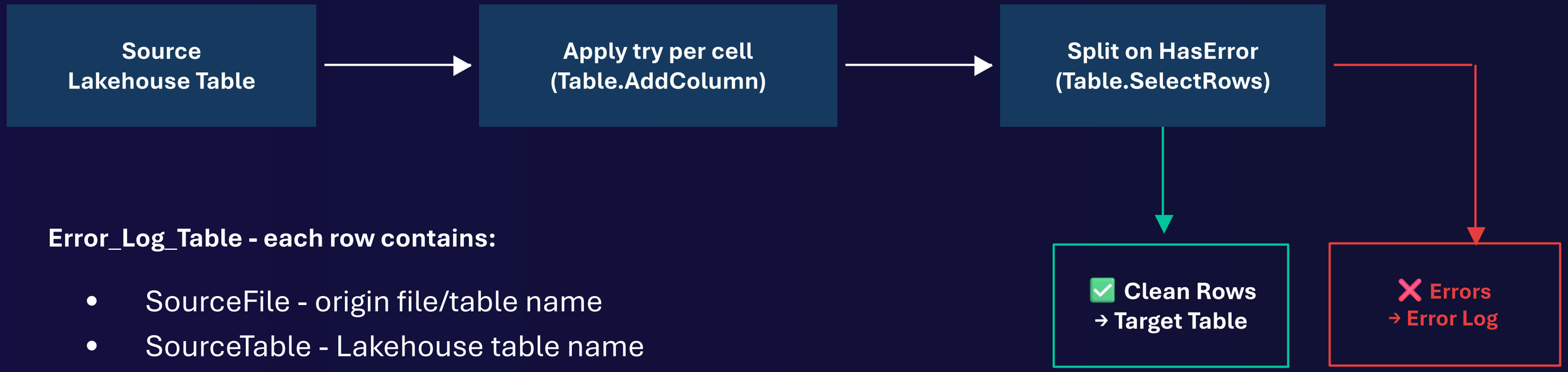
```
// try alone — returns a typed record
let
    Result1 = try Number.FromText("42")
    // Result: [HasError = false, Value = 42, Error = null]

    Result2 = try Number.FromText("abc")
    // Result: [HasError = true, Value = null, Error = [Reason="...", Message="...", Detail=...]]

// Accessing the error fields
    ErrReason  = Result2[Error][Reason],
    ErrMessage = Result2[Error][Message],
    ErrDetail  = Result2[Error][Detail]?
in
    ErrReason
```

💡 **Practical pattern: apply to each cell in a column using Table.AddColumn**

```
// Apply per cell in a column
Table.AddColumn(SourceTable, "TryQty", each try [Quantity])
// Creates a new column with [HasError, Value, Error] record for each row
```

Partial Failure Pattern

Load clean rows to target table AND route errors to audit log — in one Dataflow:



Error_Log_Table - each row contains:

- SourceFile - origin file/table name
- SourceTable - Lakehouse table name
- ColumnName - which column had the error
- ErrorReason - the M error reason
- ErrorMessage - the M error message
- OriginalValue - the raw value that caused the error

Demo

Live Demo

Fabric Dataflows Gen2 — Error Handling & Error Log

⚙️ **Setup**	Fabric Workspace with Lakehouse containing multiple tables with cell-level errors
🔍 **Inspect**	Examine the raw tables - identify which columns have errors and why
🛠️ **Handle**	Build **Dataflow Gen2** with try/catch per-column error catching
✅ **Route**	Split clean rows to target table, errors to Error Log table
📊 **Monitor**	Browse the Error Log table in the Lakehouse with full error context

Monitoring in Fabric: Pipeline + Error Log

Two layers of monitoring for Dataflows Gen2 in production:

Layer 1: Pipeline Activity Status

- Dataflow succeeds or fails at the run level
- Cell-level errors do NOT cause a run failure
- Step-level errors cause run failure → Pipeline can branch on failure
- Use On Failure activity path in Pipeline to trigger alerts or fallback logic

Layer 2: Error_Log_Table (our build)

- Captures cell-level quality errors the Pipeline cannot see
- Queryable from Lakehouse SQL endpoint or Power BI
- Enables Power BI Data Quality dashboard: error trends by source, table, column
- Can trigger alerts via Fabric Activator when error count exceeds threshold

Best Practices for Dataflows Gen2

...learned from the community and production experience

🎯 **PLAN for errors,** don't wait for them to happen	Design error handling into your query architecture from day one. Retrofitting is much harder.
🔒 **Disable Auto-Detect Data Types**	Turn off automatic type detection in Dataflow settings. Apply types explicitly as the LAST step before loading.
🛡 **Use try/catch over try/otherwise** for any monitored column	try/otherwise discards the error. try/catch preserves it for your audit log.
📈 **Set Error Log destination to Append mode**	Never overwrite the error history. Accumulate runs to enable trend analysis.
⏱ **Add LoadTimestamp** to both clean and error tables	Enables partitioning, incremental queries, and time-based analysis in Power BI.
⚙ **Use parameters** for source paths and table names	Parameterize source connections to avoid hard-coded paths that break on environment changes.
⚠ **Don't rename or remove columns** early in the query	Reordering/renaming early causes query fragility. Keep source structure until you need to transform.

Additional Resources

- **Microsoft Docs — M Error Handling** *https://learn.microsoft.com/en-us/power-query/dealing-with-errors*
- **Microsoft Docs — Power Query Error Record** *https://learn.microsoft.com/en-us/powerquery-m/error-record*
- **Fabric Dataflows Gen2 documentation** *https://learn.microsoft.com/en-us/fabric/data-factory/dataflows-gen2-overview*
- **Chris Webb — Power Query error handling deep dive** *https://blog.crossjoin.co.uk/*
- **Ben Gribaudo — M Primer Part 15: Error Handling** *https://bengribaudo.com/blog/2020/01/15/4883*
- **DataChant — Power Query tips** *https://datachant.com/*

Session Summary

1	**Dataflows Gen2 Overview**	ETL engine in Fabric: connect, transform, load to Lakehouse/Warehouse
2	**Error Types**	Step-Level (blocks run) vs Cell-Level (silent data quality risk)
3	**`try/otherwise` vs `try/catch`**	try/catch preserves the full error record - essential for audit logging
4	**Partial Failure Pattern**	Clean rows → destination table Error rows → Error Log table
5	**Error Log Table**	Centralized, append-mode audit table with full context per error
6	**Monitoring**	Pipeline status (step errors) + Error Log (cell errors) = complete coverage

Questions?

I'm happy to dive deeper on any of the patterns we covered

- Can I use this pattern with non-Lakehouse sources (SQL Server, SharePoint)?
- How do I handle errors in nested JSON or complex types in M?
- What happens to error rows when I use incremental refresh?
- Can I build a reusable error-handling function across multiple queries?

Thank You!

Bulletproof Your Dataflows: Error Handling and Data Quality in Fabric

"The goal is not zero errors.

The goal is knowing about every error."

Cristian Angyal

Microsoft MVP

Romania Power BI & Modern Excel UG Founder

meetup.com/romaniapug

Sound off.
The mic is all yours.
Influence the product roadmap.

Join the Fabric User Panel

Share your feedback directly with our Fabric product group and researchers.

https://aka.ms/JoinFabricUserPanel

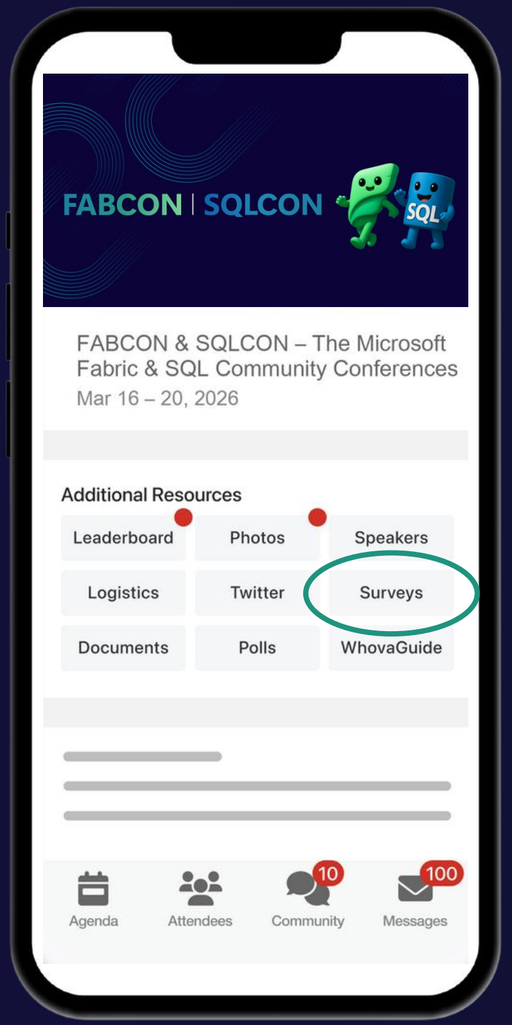
Join the SQL User Panel

Influence our SQL roadmap and ensure it meets your real-life needs

https://aka.ms/JoinSQLUserPanel

How was the session?

Complete Session Surveys in Whova for your chance to WIN PRIZES!



Two Fabric Certifications, One FREE Exam Included

Attendees can take the Fabric Analytics Engineer or Fabric Data Engineer exam for free. Be part of the 2 fastest growing role-based certifications in Microsoft history.

Request your voucher by March 31, 2026.

https://aka.ms/GetDataCertified