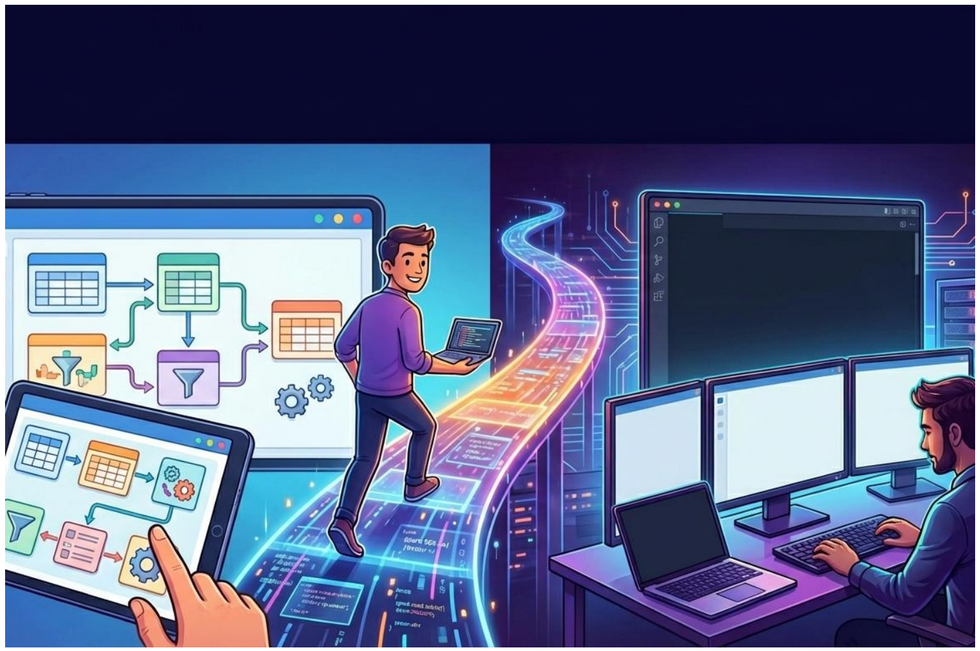
#FABCONSQLCON2026

FABCON
Microsoft Fabric
COMMUNITY CONFERENCE

SQLCON
Microsoft SQL
COMMUNITY CONFERENCE

ATLANTA MARCH 16 - 20, 2026



Transition from Data Analyst to Data Engineer

Thomas Leblanc, Fabric Architect
Nikola Ilic, Data Mozart, Microsoft Data Platform MVP

Your Transition Assistants

- **About Me…**
  - Data Platform MVP (8 years)
  - Database Normalization “Nut”
    - Retired Developer
    - Crazy about Dimensions
    - Data Vault 2.0 Certified
  - Bluesky - @PowerBIDude
  - volunteer – Local User Group(s)
    - Data, Dev and Analytics UGs

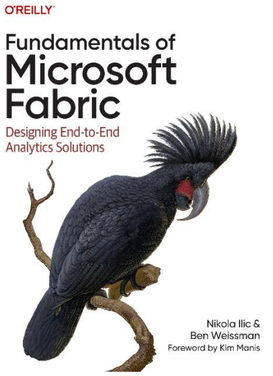
Thomas LeBlanc

Microsoft® Most Valuable Professional

Microsoft® Most Valuable Professional

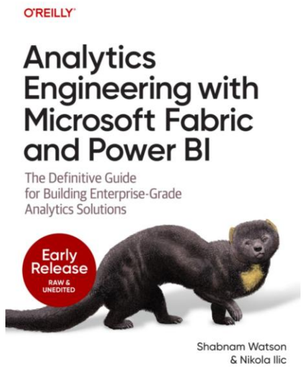
Nikola Ilic

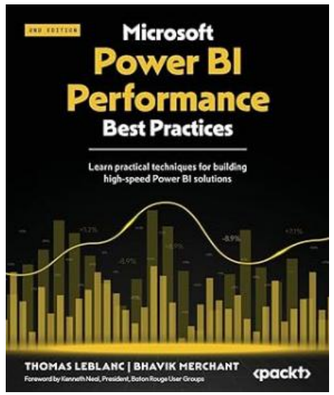
Consultant & Trainer



- *I'm making music from the data*!
- Power BI and Fabric addict, blogger, speaker…
- Father of 2, Barca & Leo Messi fan…

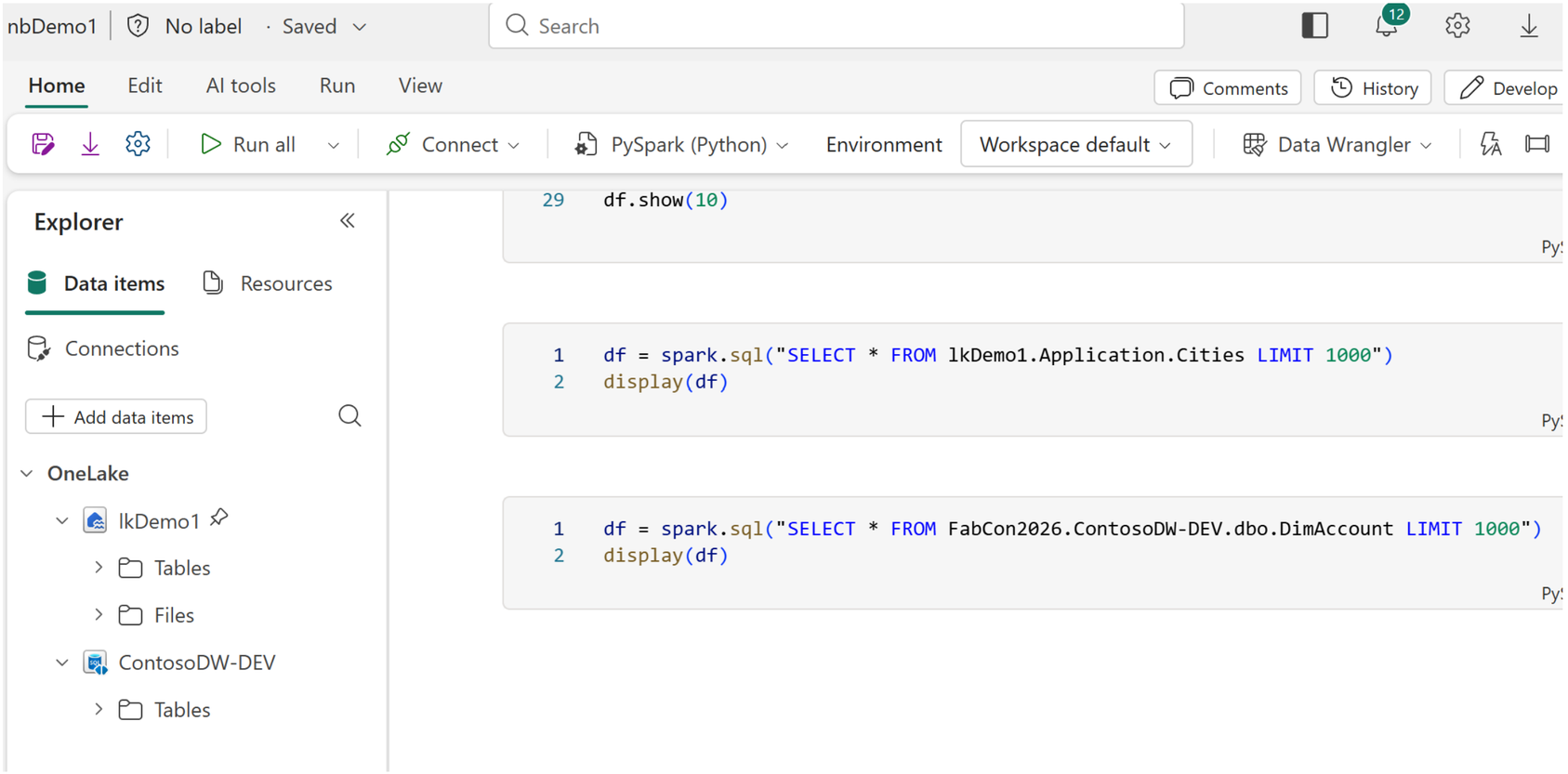
data-mozart.com





Agenda

- “The bridge.
  - Why notebooks? Mental model shift
- Connections
  - Azure SQL DB and Mirroring
  - Spark engine
- Common transformations
  - Data cleansing operations
  - Incremental loading
- Destinations
- Optimization strategies
- Takeaways and Q&A



The Bridge: When to Reach for Notebooks

Dataflow Gen2 — What it does & where it stops

✓ Low-code ingestion & Power Query transforms

✓ Incremental refresh (built-in, UI-configured)

✓ Reuse existing M / Power Query skills

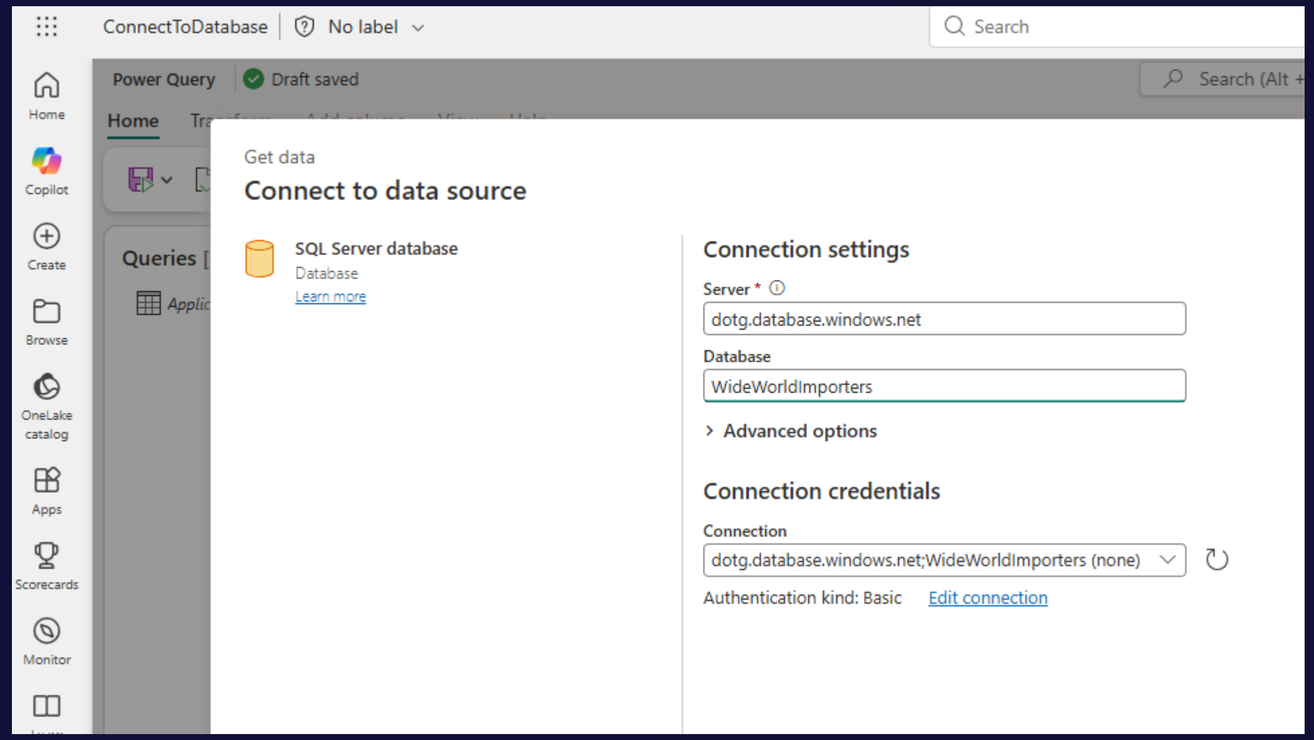
✗ IR = bucket replace, not row-level upsert
✗ Source must fully support query folding
✗ OPTIMIZE / VACUUM blocked on IR tables
✗ Max 50 buckets/query, 150/dataflow
✗ No custom error handling or retry logic
✗ No visibility into the execution plan

Fabric Notebooks — What you unlock

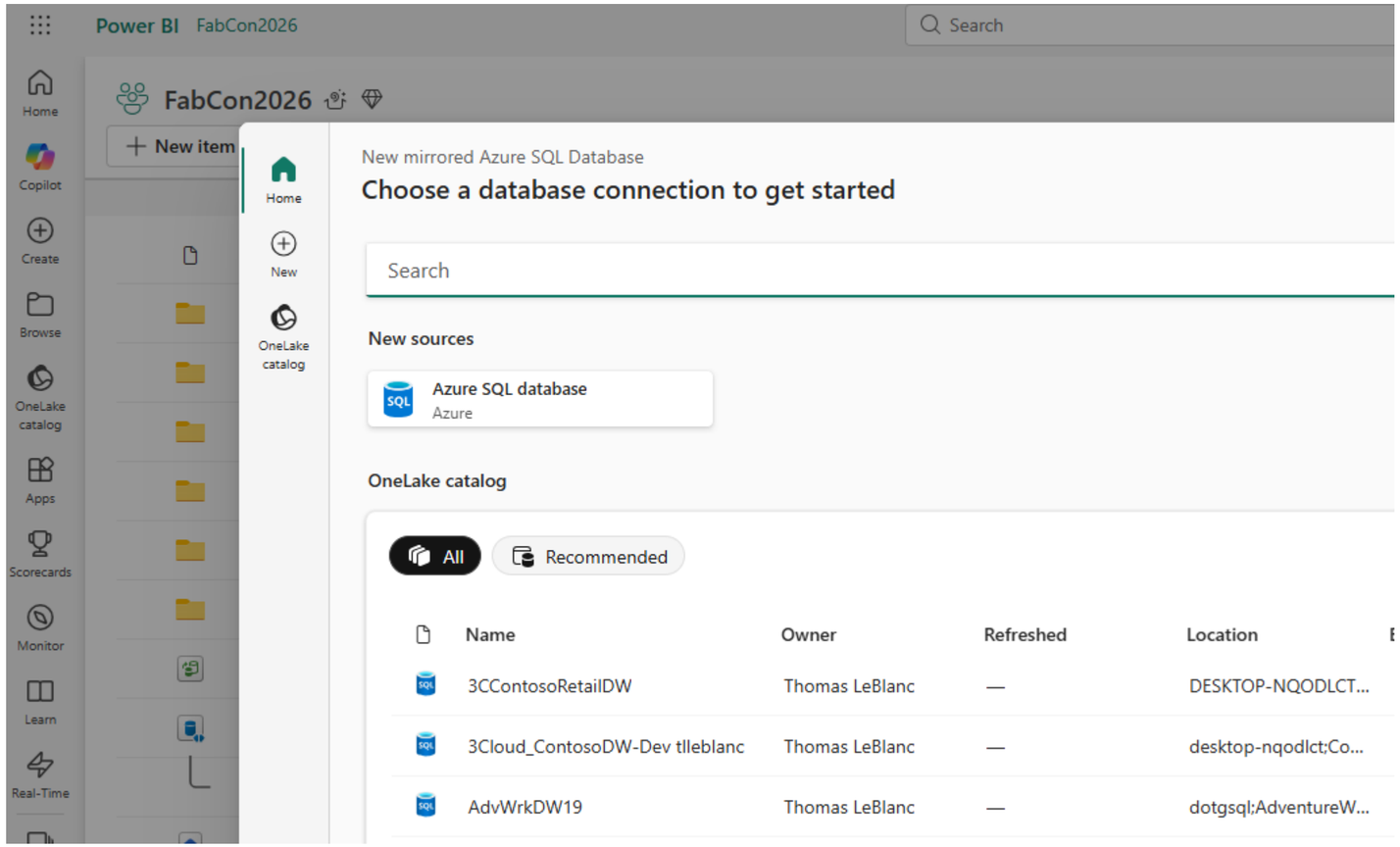
✓ True row-level MERGE / upsert via Delta Lake
✓ No folding requirement - any data source
✓ Full OPTIMIZE + V-Order after every load
✓ Unlimited scale: partitions, cache, broadcast
✓ Custom retry logic, logging, error branches
✓ Full Spark plan visibility (.explain())
✓ Version-controlled, testable helper functions
✓ Orchestrate with pipelines or job scheduler

Connections Available

- Mirroring – Azure SQL DB
  - Creates a mirrored table into a Delta Table in Lakehouse
  - Dataflow - 'Enable Staging'
- Copy Job
  - GUI interface
  - More flexible
  - Dataflow - 'Enable Staging'
- Other options
  - Pipeline with Copy activity
  - Real-time streaming
  - Notebook code to connect to data source
- Why? More flexible than Dataflows
  - Monitoring

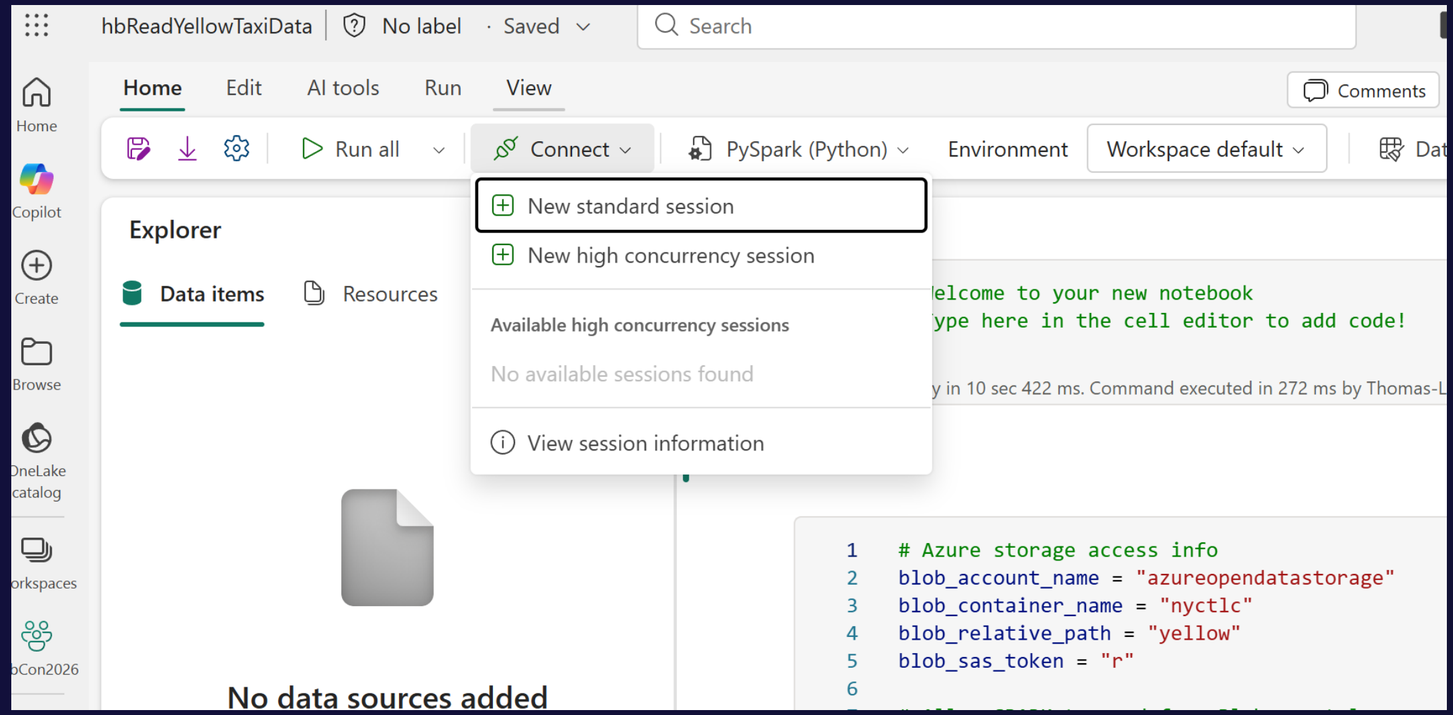


Demo - Connecting

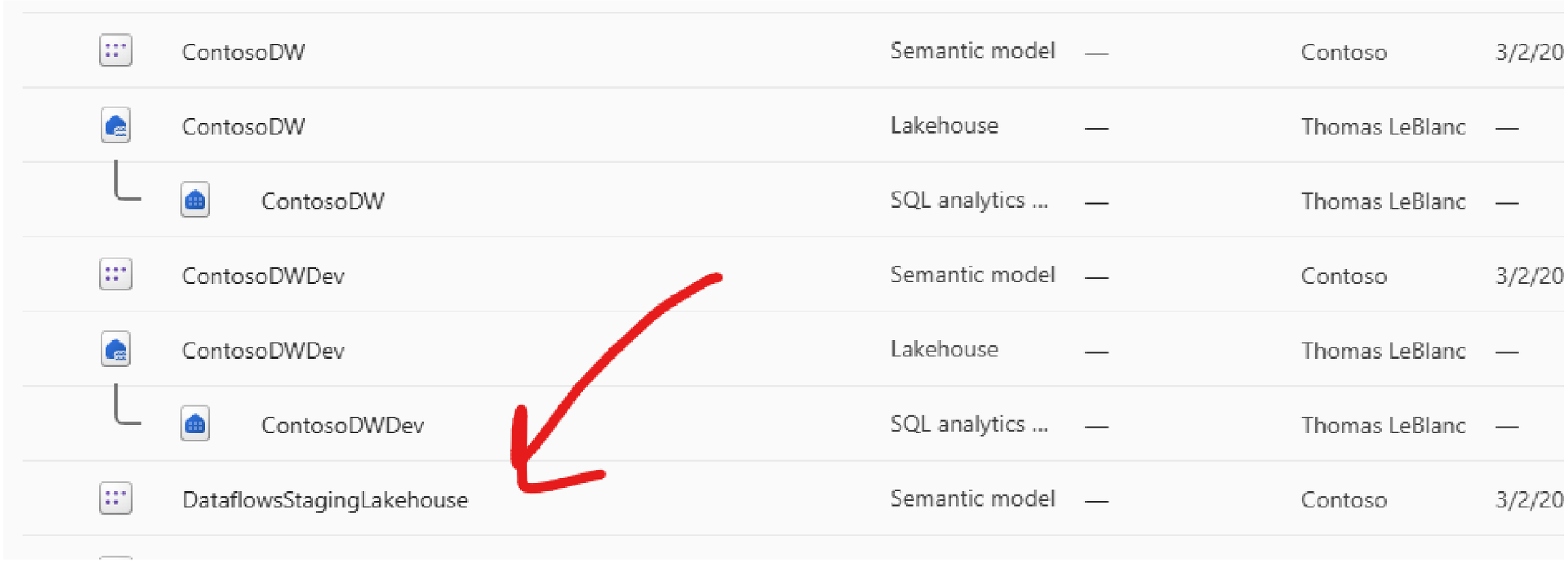


Spark Engine

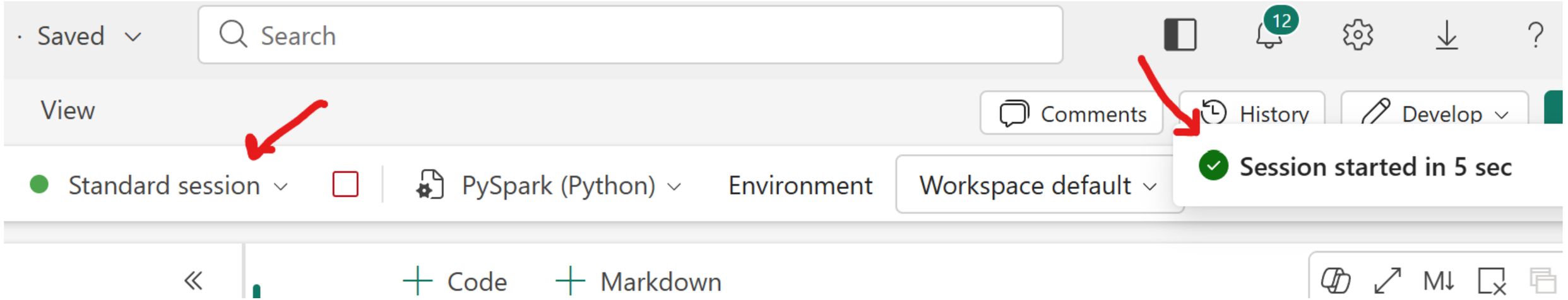
- Spark Pool –default
  - Usually, seconds to startup
  - A Default pool
- Environment changes
  - Minutes to start
  - Workspace Managed Identity (even if not used)
  - New environment (libraries, change size)
- Fabric Capacity Metrics App
  - Monitor usage
  - Less CUs than Dataflow
    - Capacity Units



Demo - Stage Lakehouse

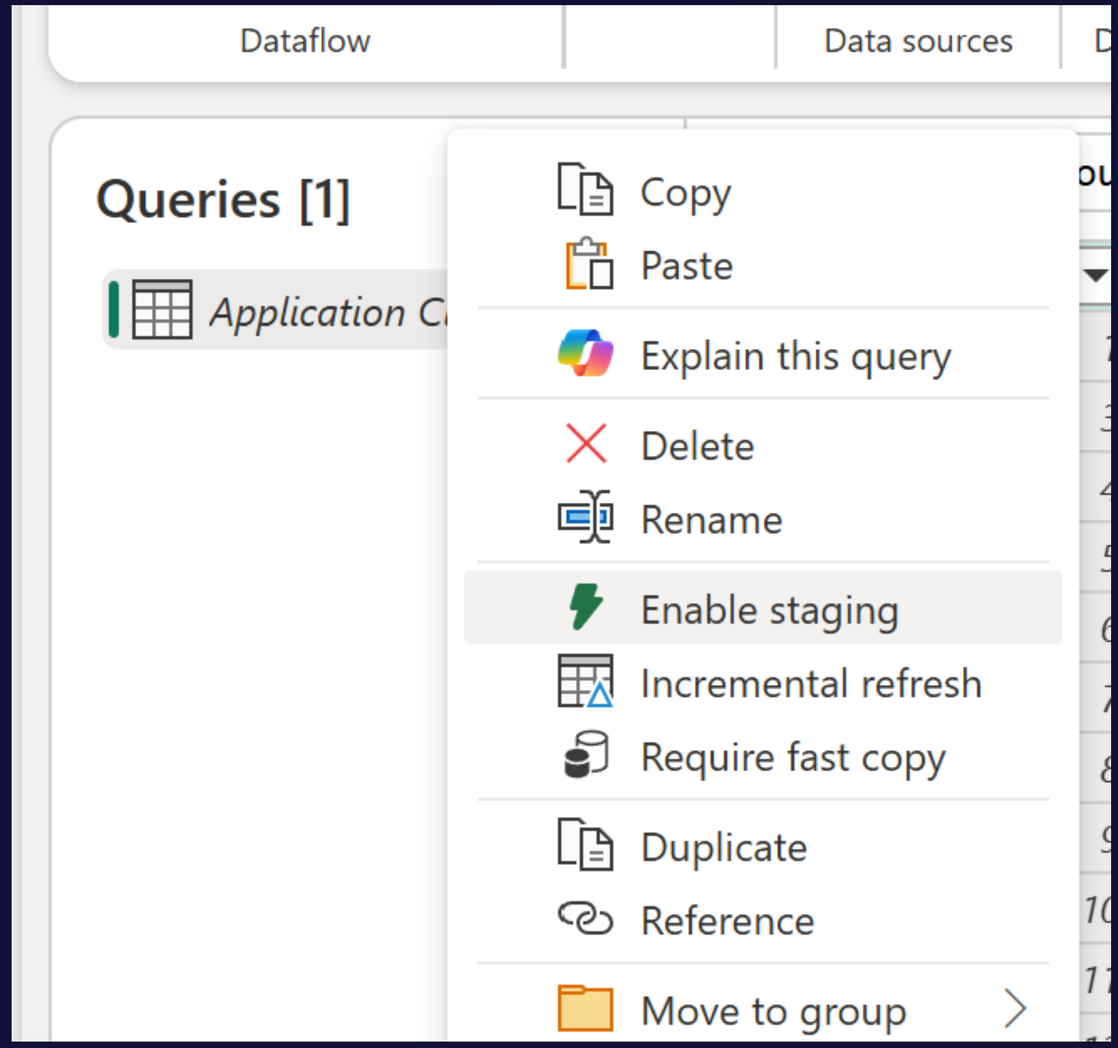


Demo - Start engine



Medallion Architecture

- Bronze
  - Like Staging
- Silver
  - Transformations
  - ODS – Operational Data Store
  - Lakehouse
- Gold
  - Warehouse
  - Dimensions and Facts
    - Bridge, plus other tables



Common Data Transformations

- Split columns
- Extract before/after delimiter
- Replace values
- Merge/Append queries
- Pivot and unpivot

Splitting Columns – M vs. PySpark

M (Power Query)

```
// Power Query - Split Column
= Table.SplitColumn(
    Source,
    "product_info",
    Splitter.SplitTextByDelimiter(
        "|", QuoteStyle.Csv),
    {"category",
     "sub_category",
     "product_name"}
)

// One column at a time
// No reuse across queries
// GUI step - not version-controlled
```

PySpark (Fabric Notebook)

```
# PySpark - split() + getItem()
from pyspark.sql.functions import split, col

parts = split(col("product_info"), r"\|")

df = (df
  .withColumn("category",     parts.getItem(0))
  .withColumn("sub_category", parts.getItem(1))
  .withColumn("product_name", parts.getItem(2))
  .drop("product_info")
)

# Reusable helper function:
def split_col(df, src, delim, names):
    p = split(col(src), delim)
    for k, name in enumerate(names):
        df = df.withColumn(name, p.getItem(k))
    return df.drop(src)
```

Splitting Columns – M vs. PySpark

M (Power Query)

```
// Power Query - Replace Value
= Table.ReplaceValue(
    Source,
    "Done", "Completed",
    Replacer.ReplaceText,
    {"status"}
)

// One value per step
// Case-sensitive by default
// Multiple steps for multiple values
// No regex support in basic Replace
```

PySpark (Fabric Notebook)

```
# Approach A: regexp_replace (case-insensitive)
from pyspark.sql.functions import regexp_replace
df = df.withColumn("status",
    regexp_replace("status",
        r"(?i)done|complete", "Completed")
)

# Approach B: when/otherwise (analyst-friendly)
from pyspark.sql.functions import when
df = df.withColumn("status",
    when(col("status").isin(
              "Done","complete","COMPLETED"),
         "Completed")
    .otherwise(col("status"))
)

# Approach C: mapping dictionary
mapping = {"Done":"Completed","complete":"Completed"}
df = df.replace(mapping, subset=["status"])
```

Engine Optimizations

- V-Order
- OPTIMIZE
- Caching
- Broadcast joins

Understanding V-Order

Write-time optimisation for Parquet files, built into Fabric's Spark runtime (disabled by default). It applies VertiPaq-compatible sorting, encoding, and compression so that Power BI, SQL endpoint and Spark all read your Delta tables faster.

Power BI Direct Lake

40–60%
faster cold-cache queries

SQL Analytics Endpoint

~10%
read speed improvement

Spark

No direct read gain but OPTIMIZE still consolidates small files

V-Order in Fabric Notebooks

```
# Option 1: Enable V-Order for the whole session
spark.conf.set("spark.sql.parquet.vorder.enabled", "true")

# Option 2: Set permanently on a table
spark.sql("""
    ALTER TABLE sales_orders
    SET TBLPROPERTIES ('delta.parquet.vorder.enabled' = 'true')
""")

# Option 3: OPTIMIZE + V-Order after every significant load
%%sql
OPTIMIZE sales_orders VORDER

-- V-Order and Z-Order are compatible — use both if needed:
-- OPTIMIZE sales_orders ZORDER BY (region) VORDER
```

Your Optimization Toolkit

V-Order + OPTIMIZE

- Write-time Parquet optimization
- Enabled by default in Fabric
- Always run after significant loads

```
%%sql OPTIMIZE sales_orders VORDER

# Enable at session level:
spark.conf.set(
  "spark.sql.parquet.vorder.enabled",
  "true")
```

OPTIMIZE (bin-compaction)

- Merges small Parquet files into larger ones (~128 MB–1 GB target)
- Reduces file-count overhead on reads

```
%%sql
-- Schedule VACUUM too:
VACUUM sales_orders RETAIN 168 HOURS

-- Reclaims unreferenced old files
```

DataFrame Caching

- Pins a DataFrame in Spark memory
- Ideal for dimension/reference tables
- Read many times in one notebook run

```
dim = (
  spark.read.table("dim_product")
       .cache()
)
dim.count()  # materialise cache
```

Broadcast Joins

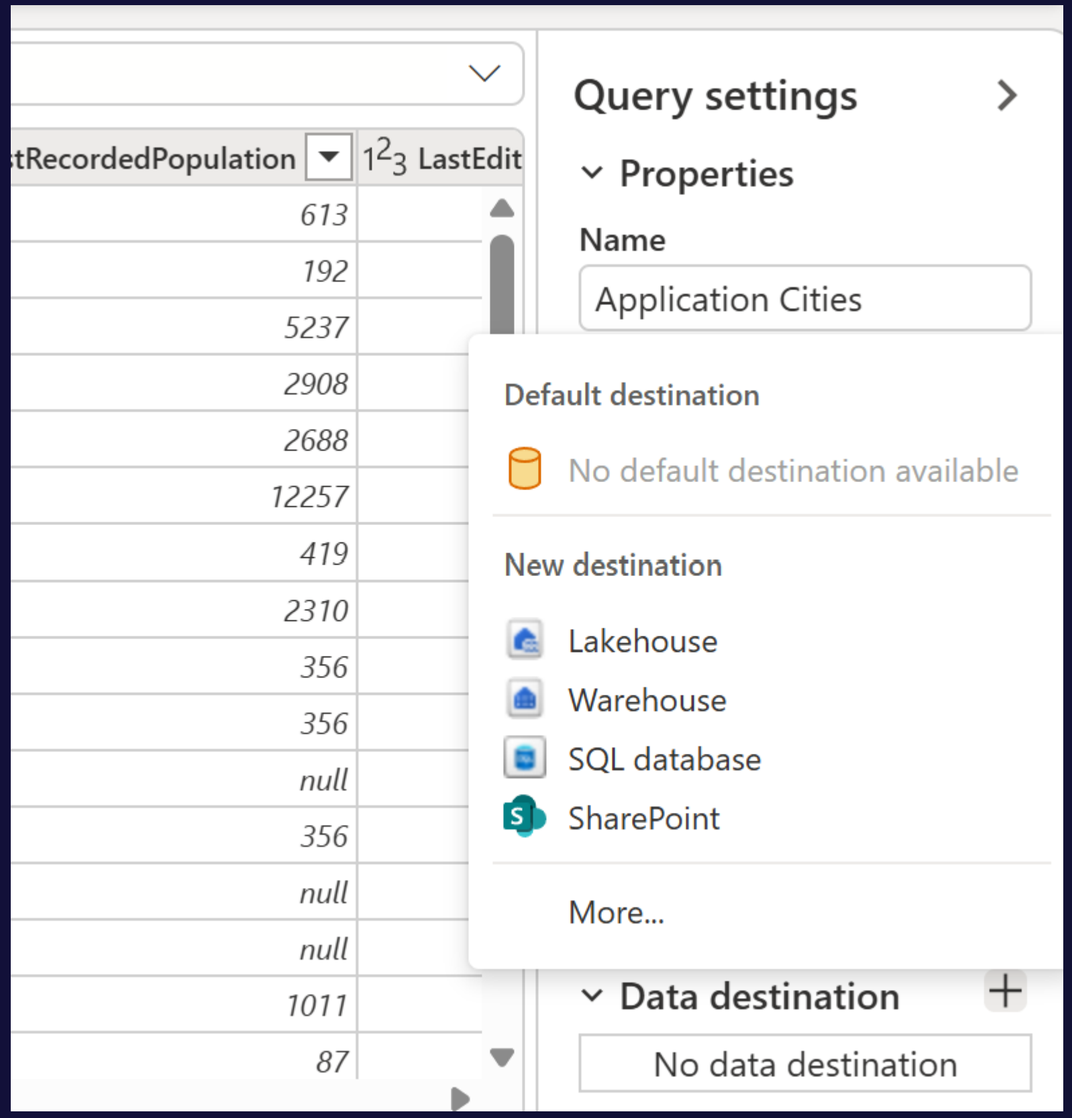
- Sends a small table to every executor, eliminating the shuffle step
- Use for tables < a few hundred MB

```
from pyspark.sql.functions import broadcast

result = large_df.join(
    broadcast(small_df),
    "product_id"
)
```

Destinations

- Dataflow Gen 1
  - None, in memory
- Dataflow Gen2
  - 4 – Lakehouse, Warehouse, SQLDB, Sharepoint
- Notebooks
  - Mainly
    - Lakehouse
    - Files (csv or parquet – and others)
    - Connect to database (drivers)
    - Almost anything Python can do



Demo - Destinations

Fact - Sale

This cell reads raw data from the *Files* section of the lakehouse, adds additional columns for different date parts and the same information is being used
table.

```python
from pyspark.sql.functions import col, year, month, quarter

table_name = 'fact_sale'

df = spark.read.format("parquet").load('Files/wwi-raw-data/full/fact_sale_1y_full')
df = df.withColumn('Year', year(col("InvoiceDateKey")))
df = df.withColumn('Quarter', quarter(col("InvoiceDateKey")))
df = df.withColumn('Month', month(col("InvoiceDateKey")))

df.write.mode("overwrite").format("delta").partitionBy("Year","Quarter").save("Tables/" + table_name)
```

...

Incremental Loading

Dataflows Gen2 Incremental Refresh

How it works:

Divides data into time-based buckets

Detects change via max(DateTime column)

Changed bucket → delete + re-insert

Requirements:

Source must support query folding

Destination: Lakehouse / Warehouse / Azure SQL

Fixed schema on destination table

Hard limits:

Update method = replace (no row-level upsert)

Max 50 buckets/query, 150/dataflow

OPTIMIZE / VACUUM blocked on IR tables

Fabric Notebook - Delta MERGE (full control)

```
from delta.tables import DeltaTable

# Load only rows newer than last run
last_ts = spark.sql(
  "SELECT MAX(updated_at) FROM sales_orders"
).collect()[0][0]

incremental = (
  spark.read.csv("Files/incremental.csv",
                 header=True)
       .filter(col("updated_at") > last_ts)
)

target = DeltaTable.forName(spark, "sales_orders")

target.alias("t").merge(
    incremental.alias("s"),
    "t.order_id = s.order_id"
).whenMatchedUpdateAll()
 .whenNotMatchedInsertAll()
 .execute()

# Then optimise + V-Order:
%%sql OPTIMIZE sales_orders VORDER
```

Incremental Loading

Dataflows Gen2 Incremental Refresh

```
// DataFlows Gen2 - Incremental Refresh
// Configured in UI (no code):
//   • DateTime column: updated_at
//   • Bucket size: 1 day
//   • Window: last 30 days
//   • Update method: REPLACE bucket

// What happens each run:
//   1. Check max(updated_at) per bucket
//   2. If changed: DELETE entire bucket
//   3. Re-INSERT all rows from source

// ⚠ Source MUST fully fold
// ⚠ Cannot run OPTIMIZE on this table
// ⚠ No row-level upsert - bucket replace only
```

Fabric Notebook - Full control

```
# Notebook - Watermark + Append (new rows only)
last_ts = spark.sql(
    "SELECT COALESCE(MAX(updated_at),'1900-01-01')"
    " FROM sales_orders"
).collect()[0][0]

new_rows = (
    spark.read.csv("Files/new.csv", header=True)
         .filter(col("updated_at") > last_ts)
)
new_rows.write.format("delta")\
    .mode("append")\
    .saveAsTable("sales_orders")

# Compact + V-Order - always safe here:
%%sql
OPTIMIZE sales_orders VORDER

# Full audit trail, zero extra work:
spark.sql("DESCRIBE HISTORY sales_orders")
```

Key Takeaways

1. Your M skills translate directly

split() ≈ Splitter · when/otherwise ≈ Conditional Column · Same logic, different syntax

2. Reusable functions beat repeated steps

One Python def replaces a dozen Dataflow steps - version-controlled, testable, shareable across any notebook

3. Dataflows Gen2 IR is real but limited

Bucket replace requires folding, blocks OPTIMIZE, and can't do row-level upsert — Delta MERGE gives you full control

4. V-Order is your first optimisation, always

Enable at session level or per table. Run OPTIMIZE ... VORDER after every significant load. Power BI Direct Lake especially benefits.

5. The transition is gradual - not all-or-nothing

Keep Dataflows for simple low-code ingestion. Reach for notebooks for incrementals, MERGE, performance control, or code reuse

Rate the Session

Transition from Data Analyst to Data Engineer: Upgrading Skills from DataFlows Gen2 to Notebooks

Thank You!

Thomas LeBlanc

- @PowerBIDude/BlueSky
- Thomaslleblanc/LinkedIn
- TheSmilingDBA@gmail.com

Nikola Ilic

- www.data-mozart.com
- learn.data-mozart.com
- nikola@data-mozart.com

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

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

Get Two Fabric Certifications for FREE

Attendees of FABCON 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 23, 2026.

https://aka.ms/fabcon/cert100