

End-to-End Data Engineering Pipeline on Azure Databricks | Retail Analytics Data Warehouse

Laraib Khan | khan.laraib.laraib77@gmail.com | www.linkedin.com/in/khanofdataengineering

1. Project Objective

Built a production-grade, cloud-native data engineering pipeline on Azure Databricks using the **Medallion Architecture** (Bronze → Silver → Gold) to create a scalable retail analytics data warehouse. Implemented incremental ingestion, SCD Type 1 & Type 2, data quality checks, orchestration, and Power BI reporting to support historical tracking and business decision-making.

Tech Stack: Azure Databricks (PySpark, Delta Lake, Delta Live Tables), Azure Data Lake Gen2, Unity Catalog, Databricks Workflows, Power BI

The screenshot shows the Azure portal interface. At the top, there's a toolbar with buttons for 'Create', 'Manage view', 'Delete resource group', 'Refresh', 'Export to CSV', 'Open query', and 'Assign tags'. Below the toolbar, a sidebar shows a collapsed 'Essentials' section. The main area has tabs for 'Resources' (which is selected) and 'Recommendations'. There are filter options: 'Filter for any field...', 'Type equals all', 'Location equals all', and a 'Add filter' button. A table lists four resources: 'databricks17ete' (Storage account, Central US), 'databricks_ete' (Azure Databricks, Central US), and 'databricks_ete_connector' (Access Connector, East US). The table has columns for Name, Type, and Location.

	Name ↑	Type	Location
<input type="checkbox"/>	databricks17ete	Storage account	Central US
<input type="checkbox"/>	databricks_ete	Azure Databricks	Central US
<input type="checkbox"/>	databricks_ete_connector	Access Connector	East US

Azure Data Lake, Databricks, Databricks connector

2. Dataset Overview

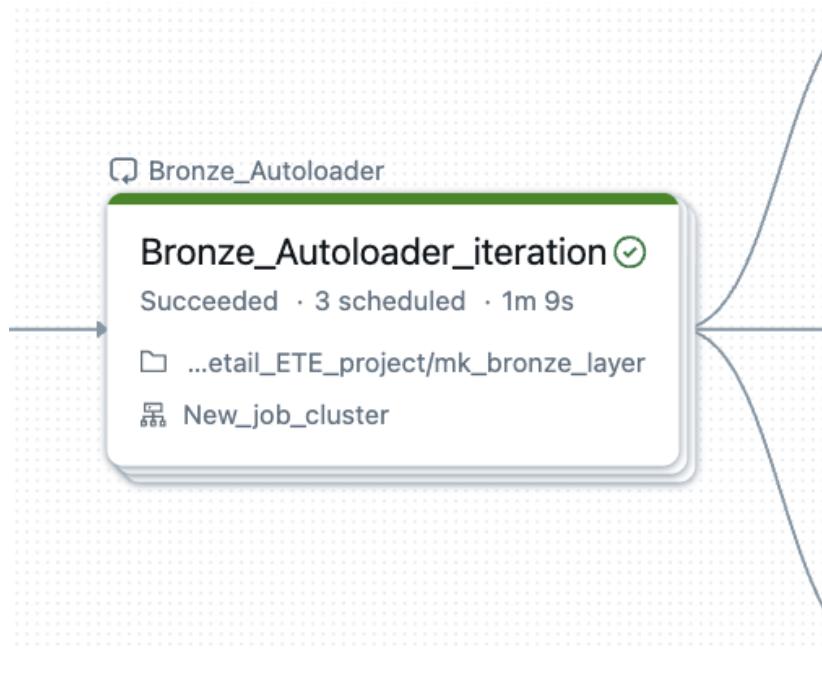
- Rows: ~12,500 incremental transactions (initial load ~10K orders + daily incremental files)

- Key Entities: Orders, Customers, Products, Regions

Key Dimensions & Measures

- Customer: customer_id, full_name, email, domain, registration_date
- Product: product_id, product_name, category, brand, price, discounted_price
- Orders: order_id, order_date, quantity, total_amount
- Region: region_id, region_name

Data Quality Note: Schema evolution and malformed records handled via Autoloader rescued_data column



▶ ✓ Nov 25, 2025 (1s) 4

```
df.display()
```

▶ (2) Spark Jobs

Table +

r_id	first_name	last_name	email	city	state	_rescued_data
1	Emily	Mooney	rushjeff@ryan.org	Johnsonmouth	MS	null
2	Andrea	Sellers	mccoykiara@kelly.com	Stephenfort	WY	null
3	Craig	Hayes	rebeccamiller@yahoo.com	South Stephenshire	LA	null
4	Bryan	Scott	lawrence05@campbell.info	Chrisland	ND	null
5	Sean	Vasquez	carrie45@yahoo.com	East Dennistown	RI	null
6	Kevin	Mccarthy	traceyramos@gmail.com	North Matthew	IN	null
7	Amanda	Doyle	scottallen@gmail.com	Joneshaven	VA	null
8	Paul	Campos	sullivanjeremy@horton-adams.com	South Nathanfurt	CT	null
9	Mary	Green	dennis03@yahoo.com	Kimberlyview	MD	null
10	James	Myers	charles58@muriilo.net	West Hector	OK	null
11	Jacob	Le	anita65@gmail.com	Houstonfurt	AR	null

rescued_data column

3. Data Engineering & ETL Pipeline (PySpark + Delta Lake + DLT)

Extract & Load (Bronze Layer)

- Used **Databricks Auto Loader** (cloudFiles) for schema inference and incremental ingestion from ADLS Gen2
 - Raw Parquet files → Bronze Delta tables with checkpointing for exactly-once processing
- Data Reading**

▶ ✓ Nov 20, 2025 (1s) 5

```
df = spark.readStream.format("cloudFiles")\
    .option("cloudFiles.format","parquet")\
    .option("cloudFiles.schemaLocation", f"abfss://bronze@databricks17ete.dfs.core.windows.net/checkpoint_{p_file_name}")\
    .load(f"abfss://source@databricks17ete.dfs.core.windows.net/{p_file_name}")
```

▶ df: pyspark.sql.dataframe.DataFrame = [order_id: string, customer_id: string ... 5 more fields]

Databricks Auto Loader reading new Parquet files from ADLS Gen2 into a streaming DataFrame

▶ ✓ Nov 25, 2025 (1s) 4

```
df.display()
```

▶ (2) Spark Jobs

Table +

r_id	first_name	last_name	email	city	state	_rescued_data
1	Emily	Mooney	rushjeff@ryan.org	Johnsonmouth	MS	null
2	Andrea	Sellers	mccoykiara@kelly.com	Stephenfort	WY	null
3	Craig	Hayes	rebeccamiller@yahoo.com	South Stephenshire	LA	null
4	Bryan	Scott	lawrence05@campbell.info	Chrisland	ND	null
5	Sean	Vasquez	carrie45@yahoo.com	East Dennistown	RI	null
6	Kevin	Mccarthy	traceyramos@gmail.com	North Matthew	IN	null
7	Amanda	Doyle	scottallen@gmail.com	Joneshaven	VA	null
8	Paul	Campos	sullivanjeremy@horton-adams.com	South Nathanfurt	CT	null
9	Mary	Green	dennis03@yahoo.com	Kimberlyview	MD	null
10	James	Myers	charles58@murillo.net	West Hector	OK	null
11	Jacob	Le	anita65@gmail.com	Houstonfurt	AR	null

_rescued_data column displayed

Silver_Orders x +

File Edit View Run Help Python v Excluded v Tabs: ON v Last edit was 5 days ago

▶ ✓ 5 days ago (<1s) 5

```
df.printSchema()
```

root
|-- order_id: string (nullable = true)
|-- customer_id: string (nullable = true)
|-- product_id: string (nullable = true)
|-- order_date: date (nullable = true)
|-- quantity: integer (nullable = true)
|-- total_amount: double (nullable = true)
|-- _rescued_data: string (nullable = true)

table schema

Transform (Silver Layer – Cleaning & Enrichment)

- Removed duplicates using window functions (row_number)
- Extracted domain from email, created full_name, applied 10% discount logic
- Added data quality expectations with @dlt.expect_or_drop / @dlt.expect_or_fail

▶ ✓ Nov 25, 2025 (1s) 6

```
df = df.withColumn("domains", split(col('email'), '@')[1])
df.display()
```

▶ (2) Spark Jobs

▶ df: pyspark.sql.dataframe.DataFrame = [customer_id: string, first_name: string ... 5 more fields]

Table +

customer_id	first_name	last_name	email	city	state	domains
1	Emily	Mooney		Johnsonmouth	MS	ryan.org
2	Andrea	Sellers		Stephenfort	WY	kelly.com
3	Craig	Hayes		South Stephenshire	LA	yahoo.com
4	Bryan	Scott		Chrisland	ND	campbell.info
5	Sean	Vasquez		East Dennistown	RI	yahoo.com
6	Kevin	Mccarthy		North Matthew	IN	gmail.com
7	Amanda	Doyle		Joneshaven	VA	gmail.com
8	Paul	Campos		South Nathanfurt	CT	horton-adams.com
9	Mary	Green		Kimberlyview	MD	yahoo.com

extracting email domains into a new "domains" column

▶ ✓ Nov 25, 2025 (1s) 9

```
df = df.withColumn("full_name", concat(col("first_name"), lit(" "), col("last_name")))
df = df.drop("first_name", "last_name")

df.display()
```

▶ (2) Spark Jobs

▶ df: pyspark.sql.dataframe.DataFrame = [customer_id: string, email: string ... 4 more fields]

Table +

customer_id	email	city	state	domains	full_name
C00001		Johnsonmouth	MS	ryan.org	Emily Mooney
C00002		Stephenfort	WY	kelly.com	Andrea Sellers
C00003		South Stephenshire	LA	yahoo.com	Craig Hayes
C00004		Chrisland	ND	campbell.info	Bryan Scott
C00005		East Dennistown	RI	yahoo.com	Sean Vasquez
C00006		North Matthew	IN	gmail.com	Kevin Mccarthy
C00007		Joneshaven	VA	gmail.com	Amanda Doyle
C00008		South Nathanfurt	CT	horton-adams.com	Paul Campos
C00009		Kimberlyview	MD	yahoo.com	Mary Green

Concatenating two columns into a full name column

▶ ✓ Nov 25, 2025 (17s) 8

```
df_gmail = df.filter(col('domains')=="gmail.com")
df_gmail.display()
time.sleep(5)

df_yahoo = df.filter(col('domains')=="yahoo.com")
df_yahoo.display()
time.sleep(5)

df_hotmail = df.filter(col('domains')=="hotmail.com")
df_hotmail.display()
time.sleep(5)

▶ (6) Spark Jobs
```

- ▶ df_gmail: pyspark.sql.dataframe.DataFrame = [customer_id: string, first_name: string ... 5 more fields]
- ▶ df_hotmail: pyspark.sql.dataframe.DataFrame = [customer_id: string, first_name: string ... 5 more fields]
- ▶ df_yahoo: pyspark.sql.dataframe.DataFrame = [customer_id: string, first_name: string ... 5 more fields]

Table +

	customer_id	first_name	last_name	email	city	state	domains
1	C00006	Kevin	Mccarthy	traceyramos@gmail.com	North Matthew	IN	gmail.com
2	C00007	Amanda	Doyle	scottallen@gmail.com	Joneshaven	VA	gmail.com

PySpark code filtering and displaying DataFrame rows by email domain

Silver_Orders × +

Python v Excluded v Tabs: ON v Last edit was 5 days ago

▶ ✓ 5 days ago (1s) 10

```
df1 = df.withColumn("flag",dense_rank().over(Window.partitionBy("year").orderBy(desc("total_amount"))))
df1.display()
```

▶ (2) Spark Jobs

- ▶ df1: pyspark.sql.dataframe.DataFrame = [order_id: string, customer_id: string ... 6 more fields]

Table +

	order_id	customer_id	product_id	order_date	quantity	1.
1	O00957	C01449	P0498	2023-10-05T00:00:00.000+00:00	5	
2	O01765	C01515	P0498	2023-05-11T00:00:00.000+00:00	5	
3	O03502	C00805	P0498	2023-09-24T00:00:00.000+00:00	5	
4	O03660	C01001	P0498	2023-10-08T00:00:00.000+00:00	5	
5	O06790	C01819	P0498	2023-02-27T00:00:00.000+00:00	5	
6	O03989	C00631	P0440	2023-11-20T00:00:00.000+00:00	5	
7	O07763	C00471	P0440	2023-04-12T00:00:00.000+00:00	5	
8	O03272	C01879	P0165	2023-02-15T00:00:00.000+00:00	5	
9	O03746	C01713	P0165	2023-10-26T00:00:00.000+00:00	5	

dense_rank() window function to flag top-spending orders per year by total amount

▶ ✓ Nov 25, 2025 (3s) 7

```
df.groupBy("domains").agg(count("customer_id").alias("total_customers")).orderBy(desc("total_customers")).display()
```

► (2) Spark Jobs

Table +

	domains	total_customers
1	gmail.com	374
2	hotmail.com	360
3	yahoo.com	331
4	brown.com	8
5	davis.com	8
6	smith.com	7
7	hernandez.com	5
8	johson.com	5

Counts distinct customers per email domain and shows the top domains by count

Load (Gold Layer – Dimensional Model)

Implemented **Star Schema** using Delta Live Tables:

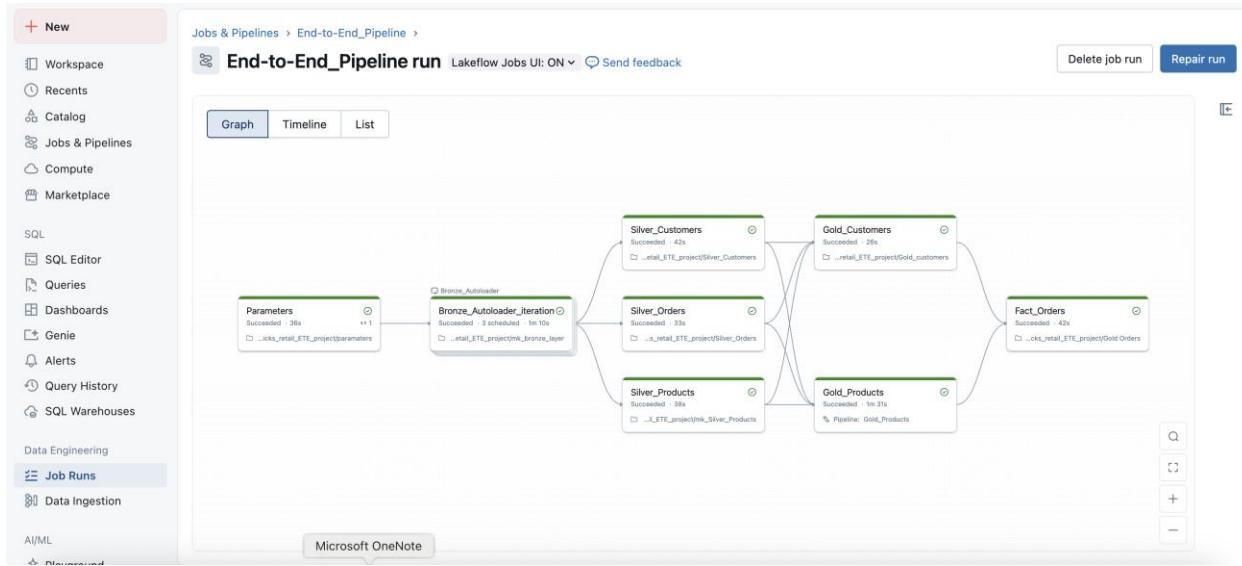
- Fact table: fact_orders
- Dimension tables:

→ dim_customers (SCD Type 2 – tracks email/domain changes over time)

→ dim_products (SCD Type 2)

→ dim_regions (SCD Type 1)

→ SQL-driven surrogate key generation and MERGE upserts to ensure incremental data integrity.



End-To -End Pipeline with successful run history

Jobs & Pipelines >

End-to-End_Pipeline

Lakeflow Jobs UI: ON ▾ [Send feedback](#)

[Run now](#)

[Runs](#) [Tasks](#)

Run

Parameters

Bronze_Autoloader

Silver_Customers

Silver_Orders

Silver_Products



Go to the latest successful run [Cancel runs](#)

Start time	Run ID	Launched	Duration	Status	Error code	Run parameters	⋮
Dec 02, 2025, 09:01 PM	232836370260711	Manually	8m 41s	✓ Succeeded			⋮
Dec 02, 2025, 06:47 PM	1020739002427372	Manually	13m 16s	✓ Succeeded			⋮
Dec 02, 2025, 01:54 PM	590099289493738	Manually	4m 45s	✓ Succeeded			⋮
Nov 20, 2025, 09:05 PM	318881347730835	Manually	54s	✓ Succeeded			⋮
Nov 20, 2025, 08:47 PM	563165573822971	Manually	1m 34s	✓ Succeeded			⋮
Nov 20, 2025, 08:47 PM	Microsoft Word ^934	Manually	59s	✓ Succeeded			⋮

Multiple successful runs

4. Advanced Analytics using Databricks SQL

Business Question	SQL Technique Used	Key Insight
-------------------	--------------------	-------------

Top 5 products by total sales within each category	DENSE_RANK() OVER (PARTITION BY category ORDER BY total_sales DESC)	Beauty dominates top 5 (e.g., “Economic Over” \$177k, “Clearly Its” \$166k); Clothing only appears once at #1 with \$149k
Total revenue by region	CASE WHEN state IN (...) THEN ‘EAST’ ... END + GROUP BY region	East: \$11.07M, West: \$7.99M, North: \$7.26M, South: \$5.05M (East = 35.3% of total)
Most profitable product categories	Derived from Ranked_Products + profit table (implied)	Beauty → Toys → Sports → Electronics → Home → Clothing (Clothing is ~23% behind Beauty)
Monthly sales trend & seasonal pattern	GROUP BY month (used in line chart)	Massive August spike (back-to-school), steep Nov-Dec drop (weak holiday push)
Regional revenue distribution	Donut chart built from Total_Sales_per_Region query	East 35.3%, West 25.5%, North 23.1%, South 16.1%

- Below are the additional Databricks SQL queries for this report.

The screenshot shows the Microsoft Azure Databricks interface. On the left is a sidebar with various navigation options like Workspace, Recents, Catalog, etc. The main area is titled 'Ranked_Products_'. It shows a SQL query with line numbers and syntax highlighting. Below the query is a table with 30 rows of data. The table has columns: product_name, category, total_sales, and products_ranked_by_category.

```

WITH Product_Sales AS (
    SELECT
        P.product_name,
        P.category,
        ROUND(SUM(F.total_amount),2) AS total_sales
    FROM mk_databricks_cat.gold.factorders AS F
    INNER JOIN mk_databricks_cat.silver.orders_silver AS Orders USING (order_id)
    INNER JOIN mk_databricks_cat.gold.dimproducts AS P USING (product_id)
    GROUP BY P.product_name, P.category
),
Ranked_Products AS (
    SELECT
        product_name,
        category,
        total_sales,
        DENSE_RANK() OVER (PARTITION BY category ORDER BY total_sales DESC) AS products_ranked_by_category
    FROM Product_Sales
)
SELECT *
FROM Ranked_Products
WHERE products_ranked_by_category <= 5
AND total_sales > (
    SELECT AVG(total_sales)
    FROM Product_Sales PS2
    WHERE PS2.category = Ranked_Products.category
)
ORDER BY category, products_ranked_by_category, total_sales DESC

```

	product_name	category	total_sales	products_ranked_by_category
1	Economic Over	Beauty	177161.62	1
2	Clearly Its	Beauty	166300.06	2
3	Piece Raise	Beauty	152588.88	3
4	Study Heavy	Beauty	151654.08	4
5	Whom Century	Beauty	142487.2	5
6	Song Approach	Clothing	147714	1
7	Space Person	Clothing	134986.6	2

Used window function `DENSE_RANK() OVER (PARTITION BY category ORDER BY total_sales DESC)` to rank products within each category

The screenshot shows the Microsoft Azure Databricks interface. On the left, there's a sidebar with various navigation options like Workspace, Recents, Catalog, Jobs & Pipelines, Compute, Marketplace, and Data Engineering. The main area is titled "Total_SalesAmount_per_region" and contains a SQL query:

```

1 SELECT
2     --C.state,
3     CASE
4         WHEN C.state IN ('ME','NH','VT','MA','RI','CT','NY','NJ','PA','DE','MD','DC','VA','WV','NC','SC','GA','FL') THEN 'EAST'
5         WHEN C.state IN ('CA','OR','WA','AK','HI','NV','ID','MT','WY','UT','CO','AZ','NM') THEN 'WEST'
6         WHEN C.state IN ('ND','SD','NE','KS','MN','IA','MO','WI','IL','IN','MI','OH') THEN 'NORTH'
7         WHEN C.state IN ('AL','AR','KY','LA','MS','OK','TN','TX') THEN 'SOUTH'
8         ELSE 'UNKNOWN'
9     END AS region,
10    ROUND(SUM(0.total_amount),2) AS total_revenue
11
12    FROM
13        mk_databricks_cat.gold.dimcustomers AS C
14        FULL JOIN mk_databricks_cat.gold.factorders AS O
15        ON C.DimCustomerKey = O.DimCustomerKey
16        FULL JOIN mk_databricks_cat.gold.dimproducts AS P
17        ON O.DimProductKey = P.product_id
18        GROUP BY region--, C.state
19        ORDER BY total_revenue DESC;
20

```

Below the query, there's a table titled "Add parameter" with a single row of data:

region	total_revenue
EAST	11071192.28
WEST	7991939.85
NORTH	7255411.25
SOUTH	5041612.7

At the bottom, it says "4 rows | 0.65s runtime".

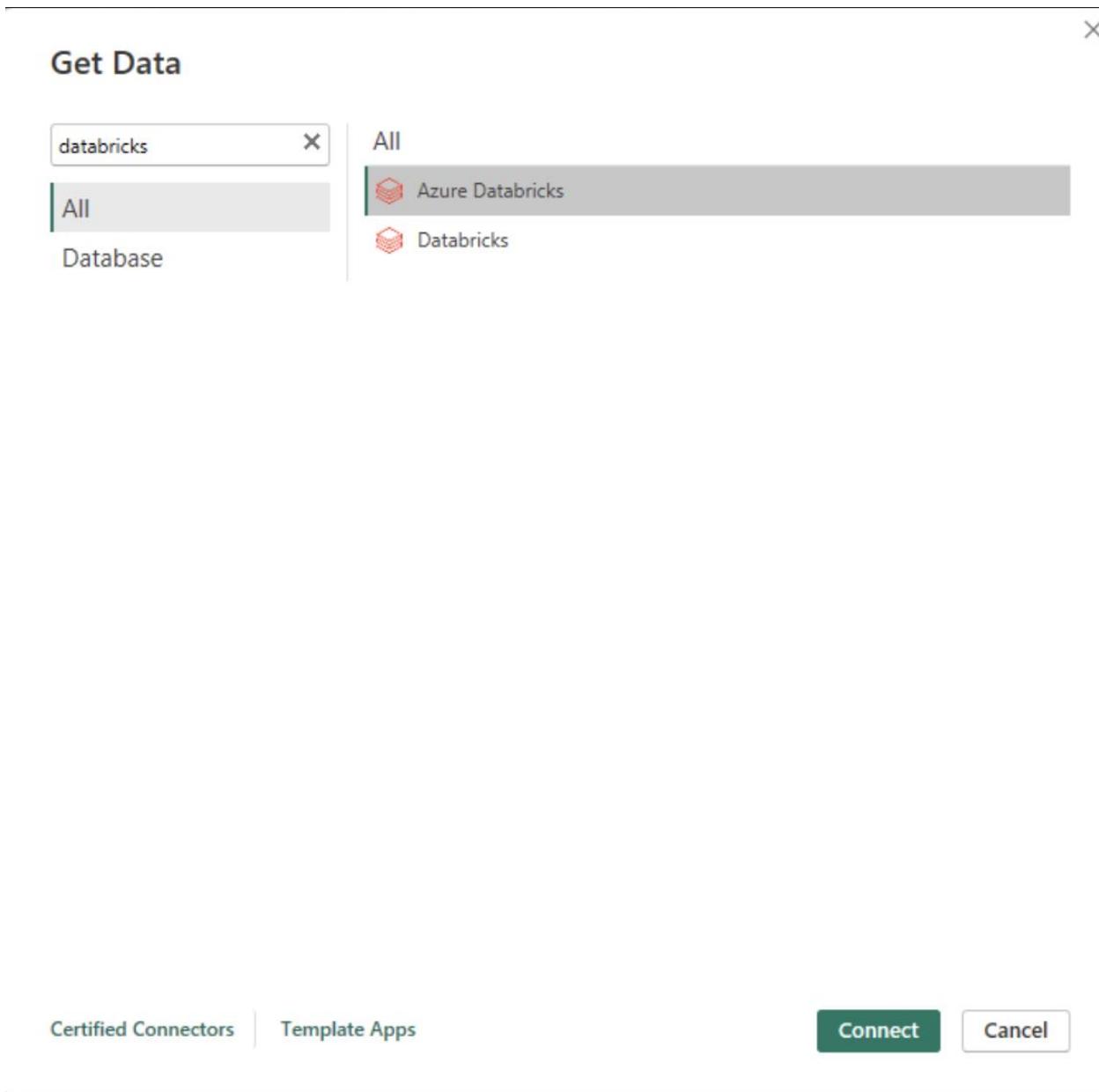
SQL query grouping total revenue by U.S. region using a CASE statement on state, ordered descending

5. Data Visualization – Interactive Power BI Dashboard

- Connected via **Azure Databricks** in Power BI's Get Data feature
- Fully interactive with slicers: Year, Category, Brand, Region, Domain

Key Visuals Included:

- Regional revenue donut (East 35.3%, West 25.5%, North 23.1%, South 16.1%)
- Category profit bar chart (Beauty → Clothing)
- Full-year monthly sales line chart showing August peak & Q4 drop
- Most profitable regions bar chart (East 11.07M, etc.)
- Individual Product Slicer



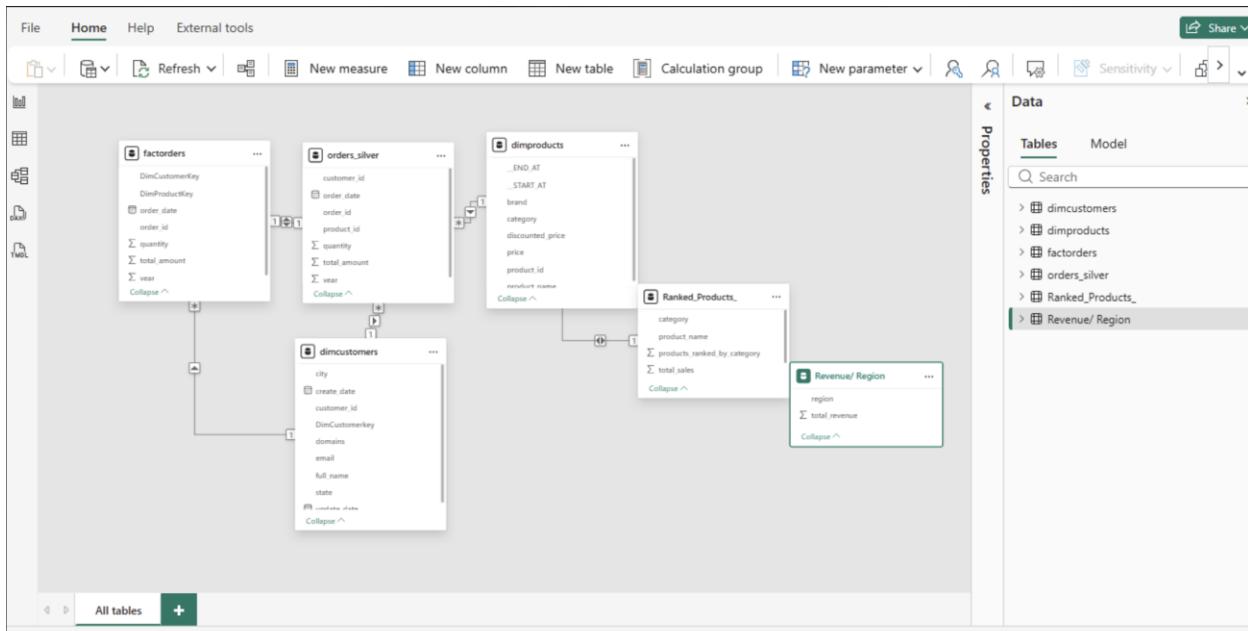
Connected via Azure Databricks in Power BI's Get Data feature

Queries [6]

factorders

order_id	order_date	quantity	total_amount	year	DimCustom
000001	3/22/2023 12:00:00 AM	3	2022.87	2023	
000002	6/30/2023 12:00:00 AM	2	3560.74	2023	
000003	11/6/2023 12:00:00 AM	3	5903.52	2023	
000004	2/27/2024 12:00:00 AM	3	4107.99	2024	
000005	10/13/2024 12:00:00 AM	5	5784.95	2024	
000006	5/17/2023 12:00:00 AM	5	4077.75	2023	
000007	1/18/2024 12:00:00 AM	4	4907.64	2024	
000008	1/10/2023 12:00:00 AM	4	7037.88	2023	
000009	4/20/2023 12:00:00 AM	3	4076.97	2023	
000010	7/7/2023 12:00:00 AM	4	5695.64	2023	
000011	3/2/2023 12:00:00 AM	1	1633.64	2023	
000012	5/2/2024 12:00:00 AM	4	7452.32	2024	
000013	8/5/2024 12:00:00 AM	1	1286.8	2024	
000014	4/10/2023 12:00:00 AM	1	958.75	2023	
000015	9/12/2023 12:00:00 AM	2	2388.44	2023	
000016	8/7/2024 12:00:00 AM	5	3902.25	2024	
000017	9/25/2024 12:00:00 AM	1	1203.96	2024	
000018	2/17/2023 12:00:00 AM	5	6931.6	2023	
000019	12/14/2024 12:00:00 AM	1	1689.45	2024	
000020	3/5/2024 12:00:00 AM	5	4699.5	2024	
000021	5/9/2024 12:00:00 AM	5	6476.85	2024	

Transformed columns to correct data types



Reviewed the model to ensure table relationships were set



Completed Product & Category Insights Dashboard

6. Key Business Recommendations (Backed by Data)

- Which categories win vs. drag?** Beauty (20% sales) leads to profit at 790K; Clothing (14% sales) lags at 643K (-23%). → Double down on Beauty (expand SKUs, hero placement); fix margins on Clothing or phase out in weak regions.
- Where should we invest in 2026?** East (11.07M) + North (7.26M) = 55% of revenue; West & South underperform. → Allocate 55–60% of budget to East/North; test turnaround campaigns in South/West or de-prioritize.
- Why did the August spike & Nov-Dec crash?** Heavy back-to-school push, zero holiday follow-through. → Shift major promo budget to Nov-Dec; build 3–4 months Beauty inventory for Q4 2026.
- Quick-win opportunities**
- Create high-margin Beauty + Toys + Sports gift bundles for holiday (top 3 profit categories, overlapping buyers).
- Run regional category heatmaps: e.g., reduce Clothing stock in West/South, overload Toys in North/East for Q4.

Key Technical Skills Demonstrated

- Medallion Architecture (Bronze → Silver → Gold)
- Incremental Ingestion with Autoloader & Checkpointing

- SCD Type 1 & Type 2 Implementation in Delta Lake
- Data Quality with Delta Live Tables Expectations
- Star Schema Modeling + Surrogate Keys
- Pipeline Orchestration with Databricks Workflows
- Unity Catalog Governance & External Locations
- Power BI Integration via Azure Databricks Connector
- Reusable PySpark Functions & OOP Patterns