SPONSORS

altitudo

beanTech
IT moves your business

fluentis
the ERP ready to go live!

[stesi]
Powered by Innovation

Marco Pozzan

- Consulente e formatore in ambito business intelligence
- Nel 2020 Fondatore e CTO start-up per analisi dati e per insurance data analytics per le compagnie assicurative

  Dal 2017 mi occupo di architetture big Data e in generale di tutta la proposizione data platform di Microsoft.
- Docente all'Università di Pordenone per i corsi IFTS di analisi Big Data
- Community Lead di 1nn0va (www.innovazionefvg.net)
- MCP,MCSA,MCSE dal 2017 MCT e dal 2014 MVP per SQL Server e relatore in diverse conferenze sul tema.
  - marco.pozzan@regolofarm.com
  - @marcopozzan.it
  - www.marcopozzan.it
  - http://www.scoop.it/u/marco-pozzan
  - http://paper.li/marcopozzan/1422524394

MVP Microsoft Most Valuable Professional Reconnect

Microsoft MCT

Microsoft CERTIFIED AZURE FUNDAMENTALS

Microsoft CERTIFIED AZURE DATA ENGINEER ASSOCIATE

Microsoft CERTIFIED Professional

Microsoft CERTIFIED Solutions Associate SQL Server 2012/2014

Saturday innova community

section

Data Warehouse with Fabric on data lakehouse

What is Microsoft Fabric?

A unified SaaS platform for all your analytics needs

Real-time Analytics

Ingest & query real-time data within seconds

Store realtime data in OneLake

BI

Reports & Dashboards With Power BI

Data Science

Run notebooks within seconds without managing clusters

Train models and experiment between different models versions

Data Warehouse

Powered by Delta Parquet

Accessible through SQL and Spark

Decoupled storage & processing

Organized around workspaces

OneLake

Unified lakehouse With an open format

Data Integration

Use no-code or notebooks for ETL or ELT Jobs

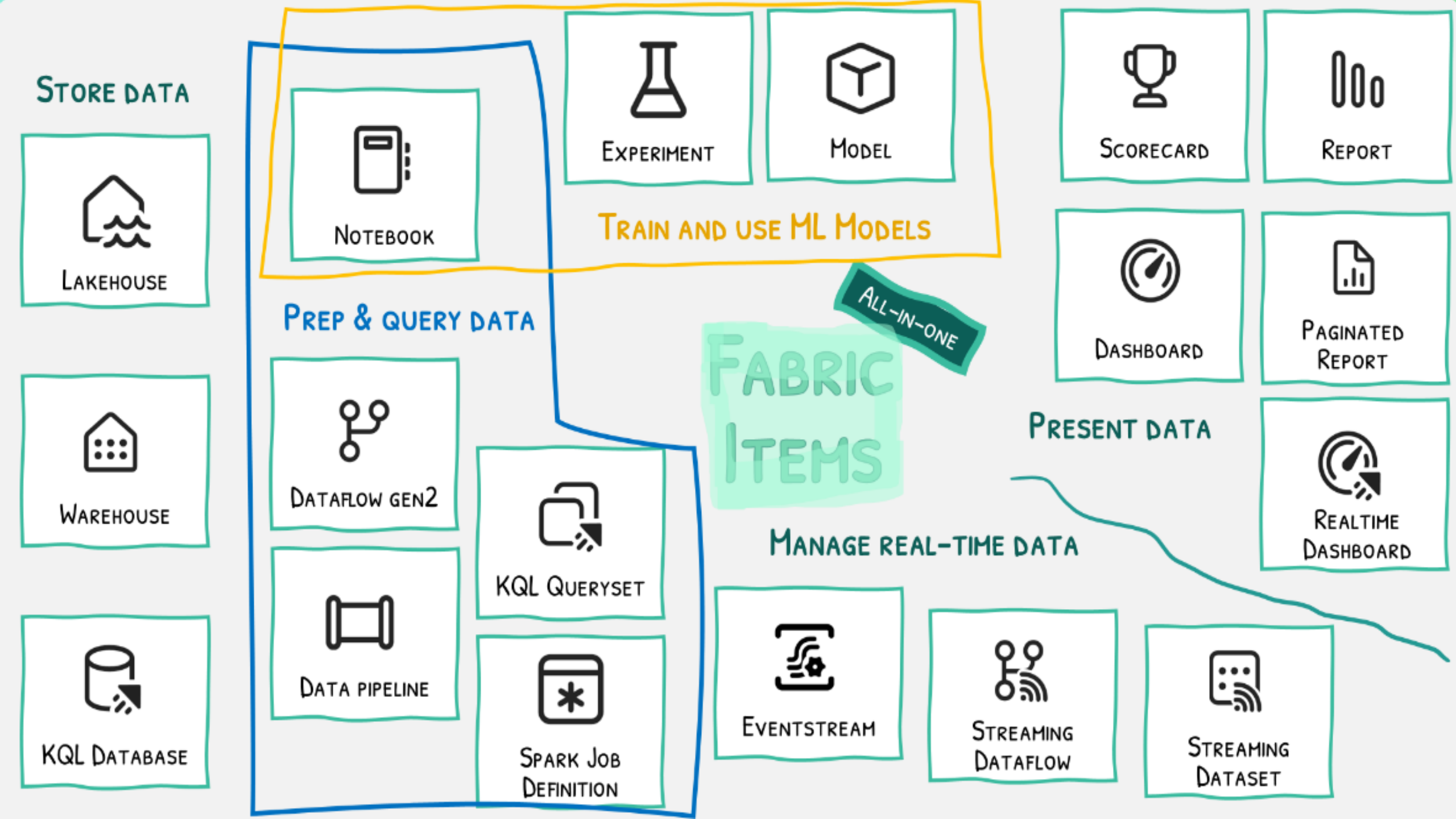
Schedule & run jobs without managing (spark) clusters

Unified security

Across all types of data stores & engines

Shortcuts

Easy access to data stored in other clouds


Store data
Lakehouse
Warehouse
KQL Database
Prep & query data
Notebook
Dataflow gen2
Data pipeline
KQL Queryset
Spark Job Definition
Train and use ML Models
Experiment
Model
All-in-one
Fabric Items
Manage real-time data
Eventstream
Streaming Dataflow
Streaming Dataset
Present data
Scorecard
Report
Dashboard
Paginated Report
Realtime Dashboard

OneLake

"The OneDrive for your data"

Visually Explained

All analytical engines can access OneLake Data

SQL | Spark | Power BI | KQL DB

DataLake + warehouse = Lakehouse

Each Fabric Organization has one Data Lake, entirely managed for you
It's unified across all regions & you pay per GB stored (no scaling needed).

With Fabric, your warehouse Data is also stored in OneLake, and accessible in an open format

This is a **Lakehouse**

It allows you to distribute ownership to organizations, and easily collaborate on data

It contains

A files area

A tables area

Shortcuts

Allow you to query data from one lakehouse to another without data movement

Shortcuts link to other clouds too!

The Files area can contain unstructured & semi-structured data.

It's not uncommon to organize them in three "areas": Bronze, Silver & Gold

OneLake

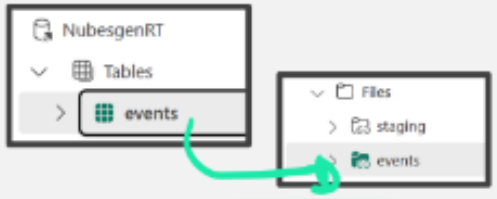
ADLS Gen2 SDK & APIs can natively access OneLake

OneLake Shortcuts

Shortcuts allow you to create a virtualized data lake, eliminating copies of data between organization domains, analytical engines, or clouds

Internal

From a lakehouse folder to a lakehouse folder



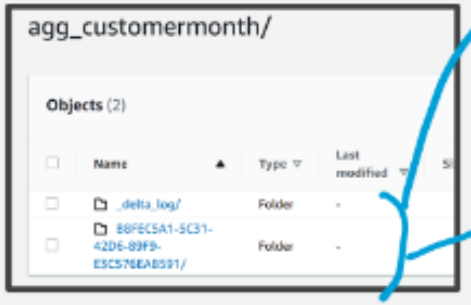
From a KQL Database Table to a lakehouse Folder

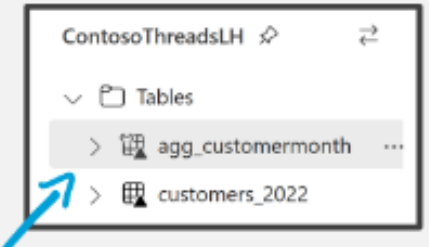
Two Types Of Shortcuts

From a lakehouse Table to a lakehouse Table

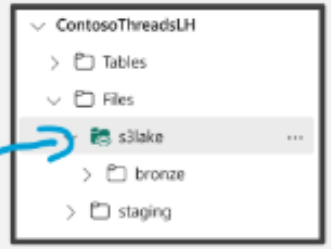
From a Warehouse Table to a lakehouse Table

External





From an ADLS Gen2/S3 folder to a lakehouse Folder



From an ADLS Gen2/S3 folder to a Table

S3 Shortcuts are read-only

Inside the Apache Parquet format

It's Open Source!

In traditional databases (Postgres, SQL Server*) and file types (CSV, JSON), Data is stored as rows...

first_name	last_name	street	isPremiumCustomer	Next Month Buying Probability
Abigail	Hayes	4009 9th Street	1	1
Amanda	Clark	87 5th Street	0	2
Melissa	Cl			
Sara	Ben			
Isaac	Hend			
Emma	Ste			
Sean	Mo			
Jason	San			
Alexander	Wa			
Megan	Mo			

Customers.csv

```
first_name,last_name,street,isPremiumCustomer,nextmbuyprob
Abigail,Hayes,4009 9th Street,1,0.26340968441351104
Amanda,Clark,87 5th Street,0,0.17177893540625921
Melissa,Clark,12650 5th Street North,1,0.5438465585898219
Sara,Bennett,203 State Street,0,0.15856312140709694
Isaac,Henderson,732 Magnolia Drive,1,0.19389081059172666
Emma,Stewart,47 Route 32,1,0.8833057932776623
Sean,Morris,254 Lincoln Avenue,0,0.4156333382834478
Jason,Sanchez,26050 Madison Avenue,0,0.5862717219123177
Alexander,Walker,54051 Valley Road,1,0.5941814808482648
Megan,Morris,31 Delaware Avenue,1,0.6289626353900487
```

...which is not optimal for analytical needs:

- For some file types (like CSV), there is no embedded schema
- Compression is not good (you're kinda limited to text compression)
- You need to scan(read) all the rows

*Unless you're using columnstore tables :)

In Parquet, data is stored as columns

Customers.parquet

first_name Abigail Amanda Melissa Sara Isaac Emma Sean Jason

last_name Hayes Clark Clark Bennett Henderson Stewart Morris Sanchez

isPremiumCustomer 1 0 1 0 1 x2 0 x2

Category 0.263409684 0.171778935 0.543846559 0.158563121 0.193890811 0.883305793 0.415633338 0.586271722

Col 1 STRING Col 2 STRING Col 3 BOOLEAN Col 4 UINT_32

File structure is simplified, omitting details like row groups.

- Parquet embeds the data schema
- Want the number of premium customers? You can compute that aggregate without reading any other unnecessary data!
- We can efficiently compress repetitive data like this one

With Fabric, you can easily read parquet files stored in OneLake or in other clouds

LAKEHOUSE VS WAREHOUSE

LAKEHOUSE		WAREHOUSE
Unlimited	Data Volume	Unlimited
Unstructured, semi-structured, structured	Type of data	Structured (Tables, Columns)
Data engineer, Data scientist	Who is using it?	Data warehouse developer, SQL engineer
Spark (Scala, PySpark, Spark SQL, R) Spark notebooks, spark job definitions	Primary skillset	SQL SQL Scripts
Spark & T-SQL (SQL Endpoint for Lakehouse, Limited T-SQL coverage)	How to read data?	Spark & T-SQL (great T-SQL coverage)
✗ No	Multi-Table transactions?	Yes ✓
Yes, files and tables can be sources	Shortcuts support	Yes – access data through lakehouse, Tables can be sources
Across lakehouse and warehouse tables & across lakehouses	Query across items	Across lakehouse and warehouse tables

Two SQL Endpoints

In your workspace

SQL endpoint for the lakehouse

	ContosoThreadsLH	SQL endpoint
	ContosoThreadsLH	Lakehouse

Automatically created with your lakehouse

Datawarehouse

Default API for Warehouse

	ContosoThreadsDWH	Warehouse

They both store data in the Open Delta Parquet format

Is read-only. Updates through Spark	Read-write	Full transactional DDL and DML support
Spark, pipelines, dataflows, shortcuts	Ingestion	SQL, Pipelines, dataflows
Medallion architecture (bronze, silver, gold)	Recommended structure	Analytical schema with Tables, views,...

You can query one from another with the "three dot" structure (e.g. Select * From dwh.dbo.customers)

Direct Lake

Power BI

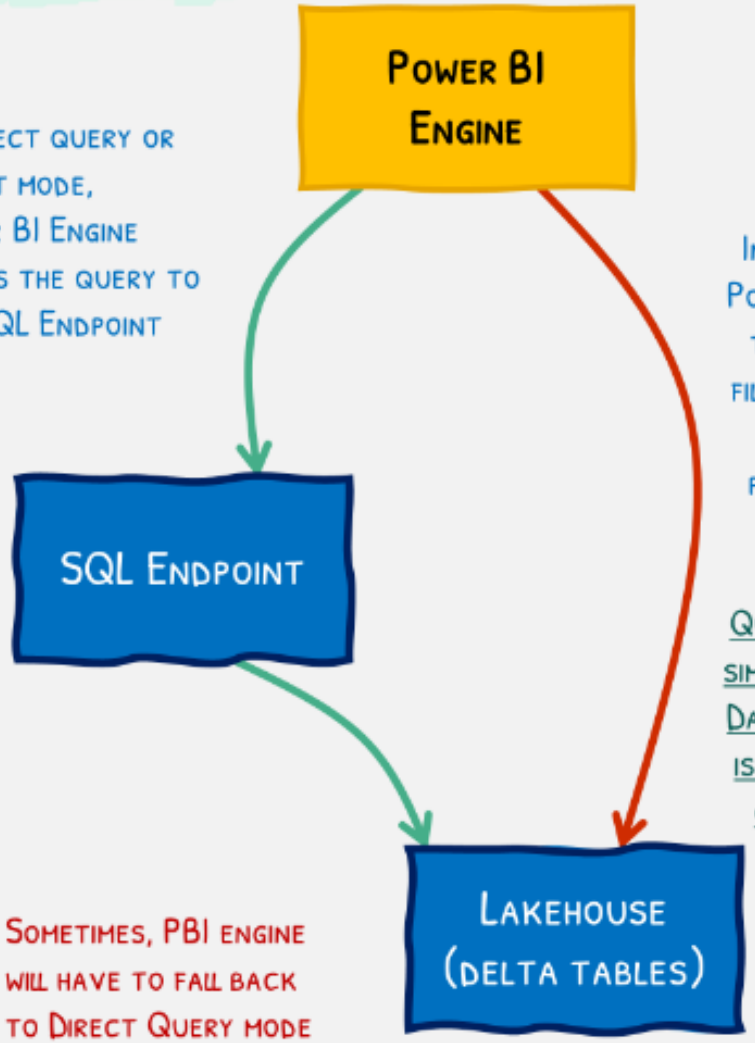
New superpower

Prerequisites

- Power BI Premium / Fabric F Capacity
- A Lakehouse + a SQL Endpoint
- Delta parquet tables with v-order

How it works?

In Direct query or Import mode, Power BI Engine passes the query to the SQL Endpoint



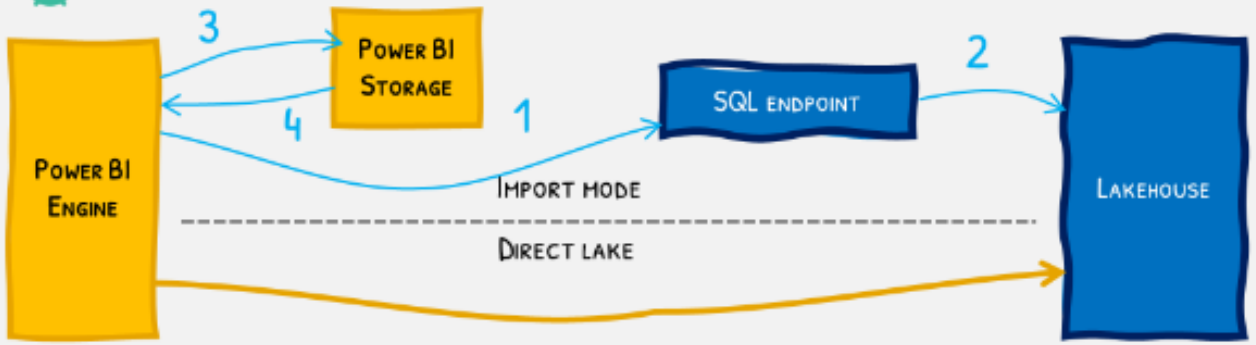
In Direct Lake mode, Power BI engine query the delta-parquet files in the Lakehouse directly, without passing through an SQL endpoint.

Query performance is similar to import mode, Data updates latency is identical to Direct query (no need to refresh!)

Sometimes, PBI engine will have to fall back to Direct Query mode

What's the magic trick?

- Parquet format is already optimized for analytics workloads
- Power BI can use parquet files instead of "PBI Analysis Services" storage



- "V-order" is a Fabric-optimized way of writing parquet files, providing better query performance, while still being compatible with parquet
- All Fabric engines write v-ordered parquet files by default

Try it yourself at aka.ms/directlake

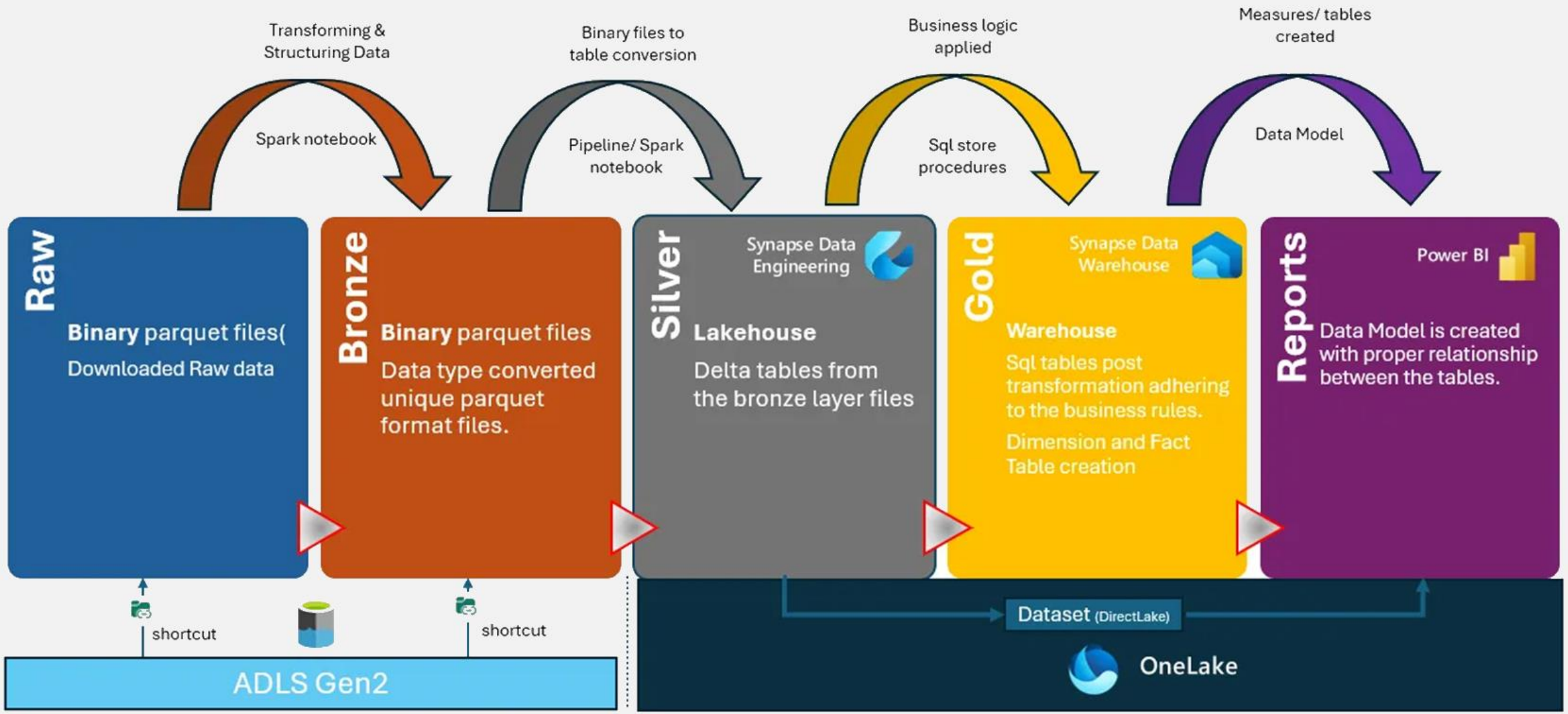
Medallion Architecture

Databricks recommends taking a layered approach to creating a single source of truth for enterprise data products.

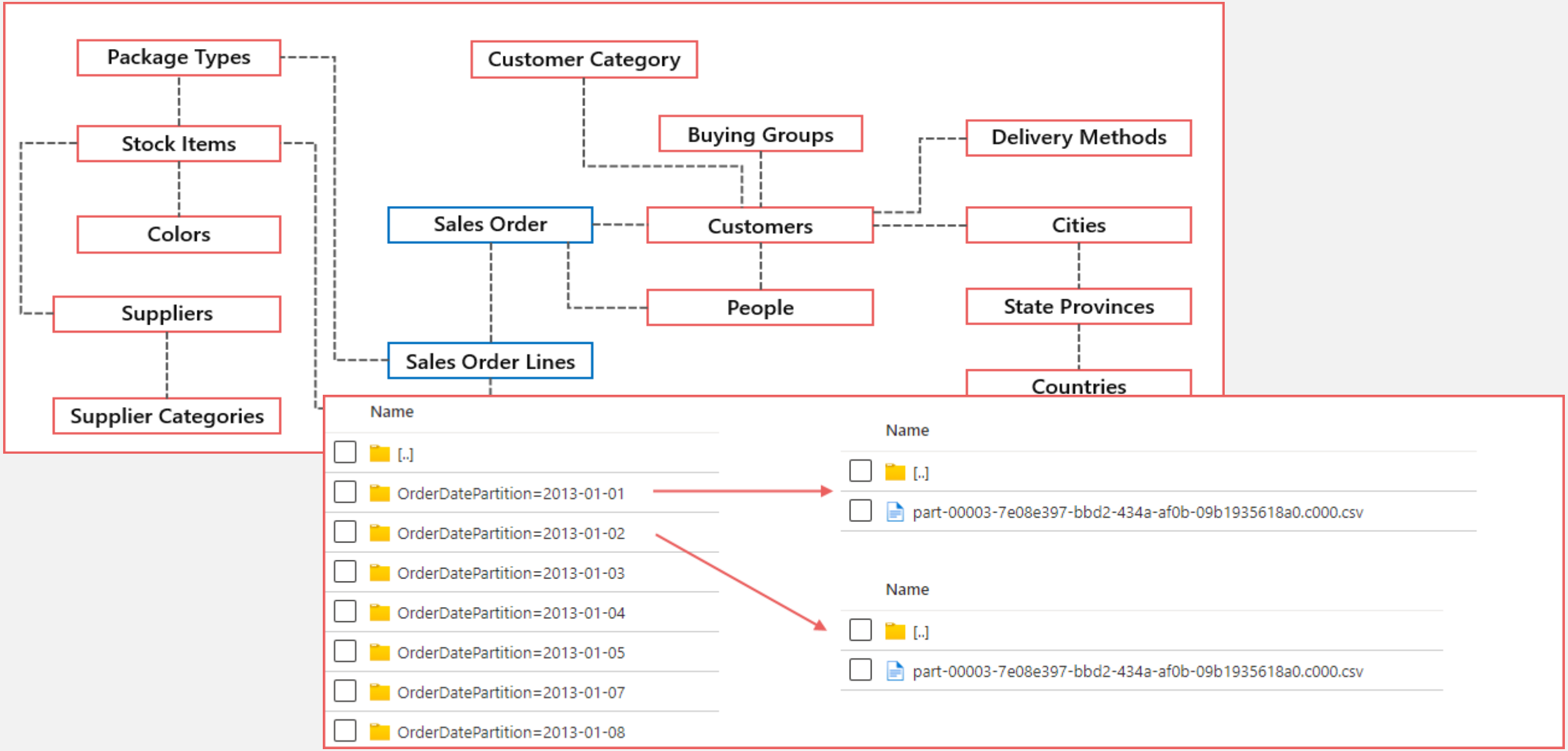
This architecture ensures atomicity, **consistency**, **isolation**, and **durability** as data passes through multiple levels of validations and transformations before being stored in a layout optimized for efficient analysis.

It is important to note that this medallion architecture is not a substitute for other dimensional modeling techniques. The schemas and tables within each tier can take a variety of forms and degrees of normalization depending on the frequency and nature of data updates and downstream use cases for the data.

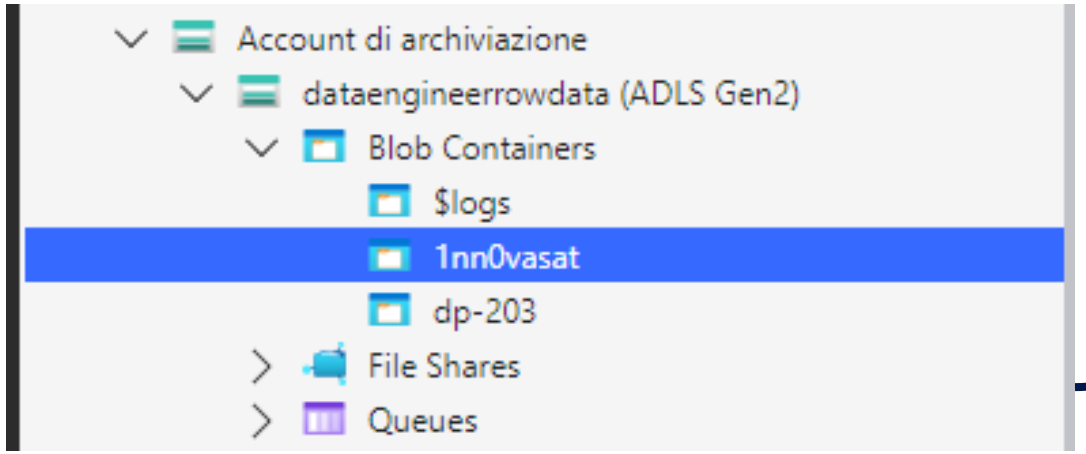
Medallion Architecture

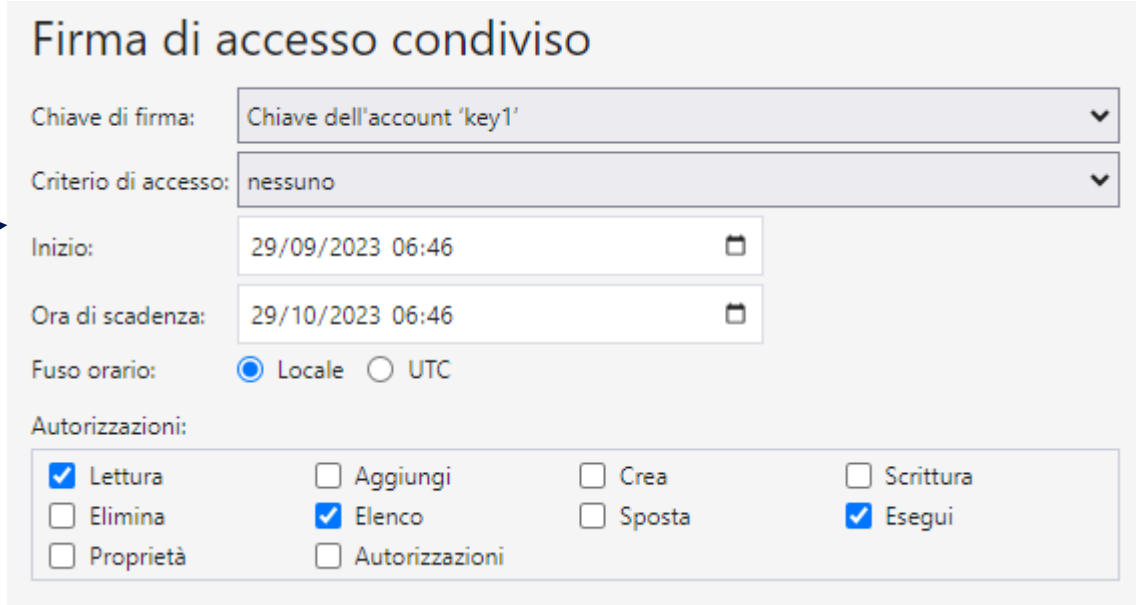


Demo


Package Types
Customer Category
Stock Items
Buying Groups
Delivery Methods
Colors
Sales Order
Customers
Cities
Suppliers
People
State Provinces
Sales Order Lines
Countries
Supplier Categories
Name
[..]
OrderDatePartition=2013-01-01
OrderDatePartition=2013-01-02
OrderDatePartition=2013-01-03
OrderDatePartition=2013-01-04
OrderDatePartition=2013-01-05
OrderDatePartition=2013-01-07
OrderDatePartition=2013-01-08
Name
[..]
part-00003-7e08e397-bbd2-434a-af0b-09b1935618a0.c000.csv
Name
[..]
part-00003-7e08e397-bbd2-434a-af0b-09b1935618a0.c000.csv

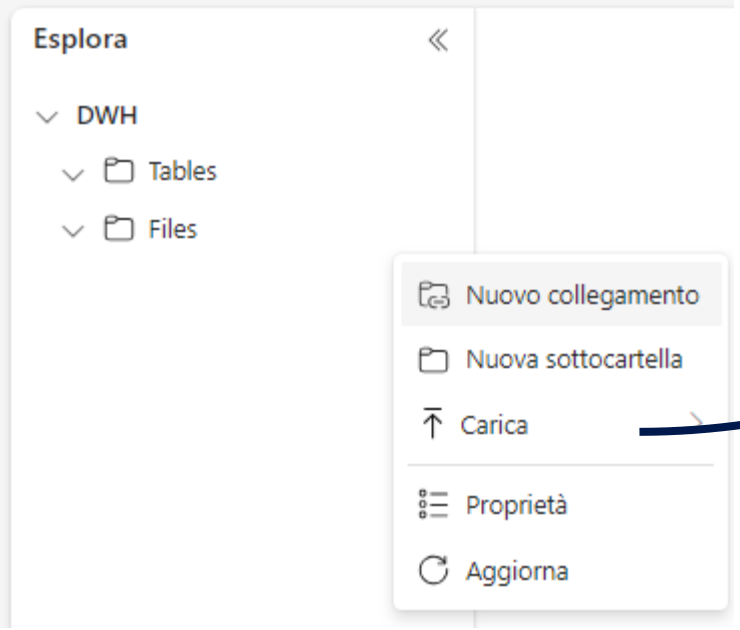
Demo - token SAS generation

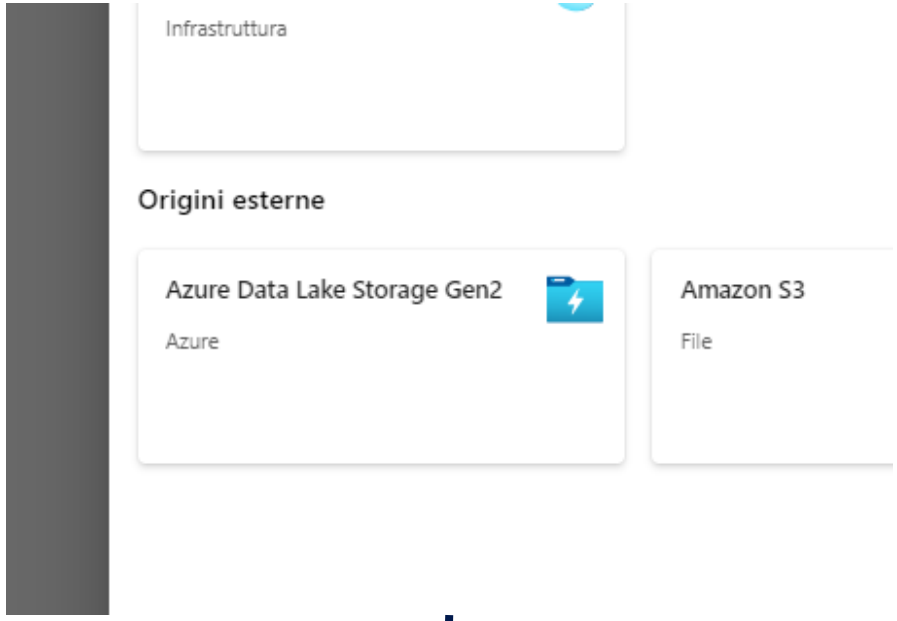


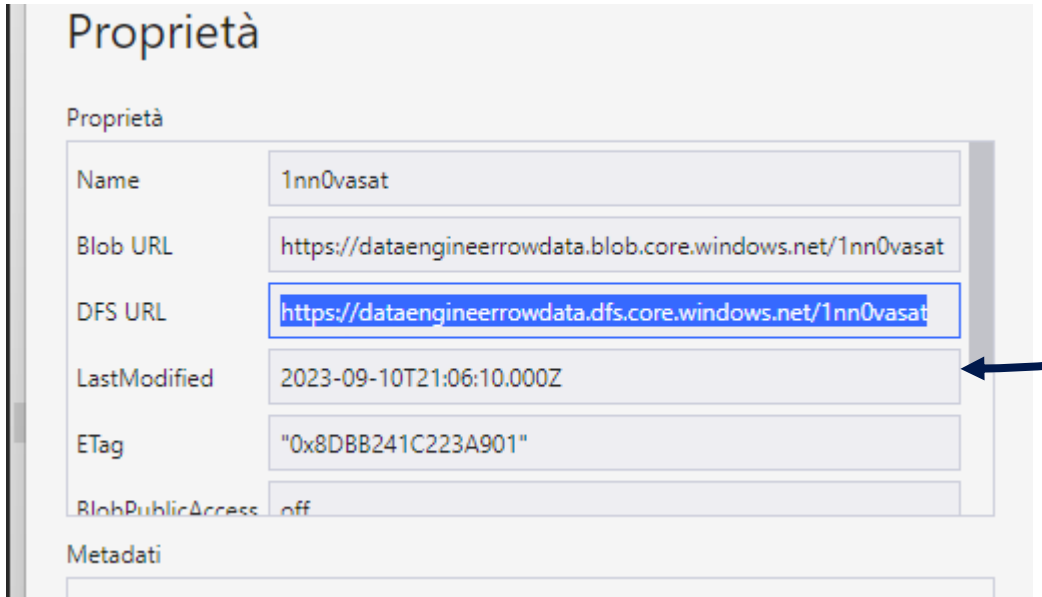


?sv=2020-02-10&st=2023-09-29T04%3A46%3A14Z&se=2023-10-29T05%3A46%3A00Z&sr=c&sp=rle&sig=uds%2BXKBHBBkelOkOFjeOblWlxQ5vPWBk0unBtysXgeY%3D

Demo - Shortcut configuration for bronze layer (part 1)

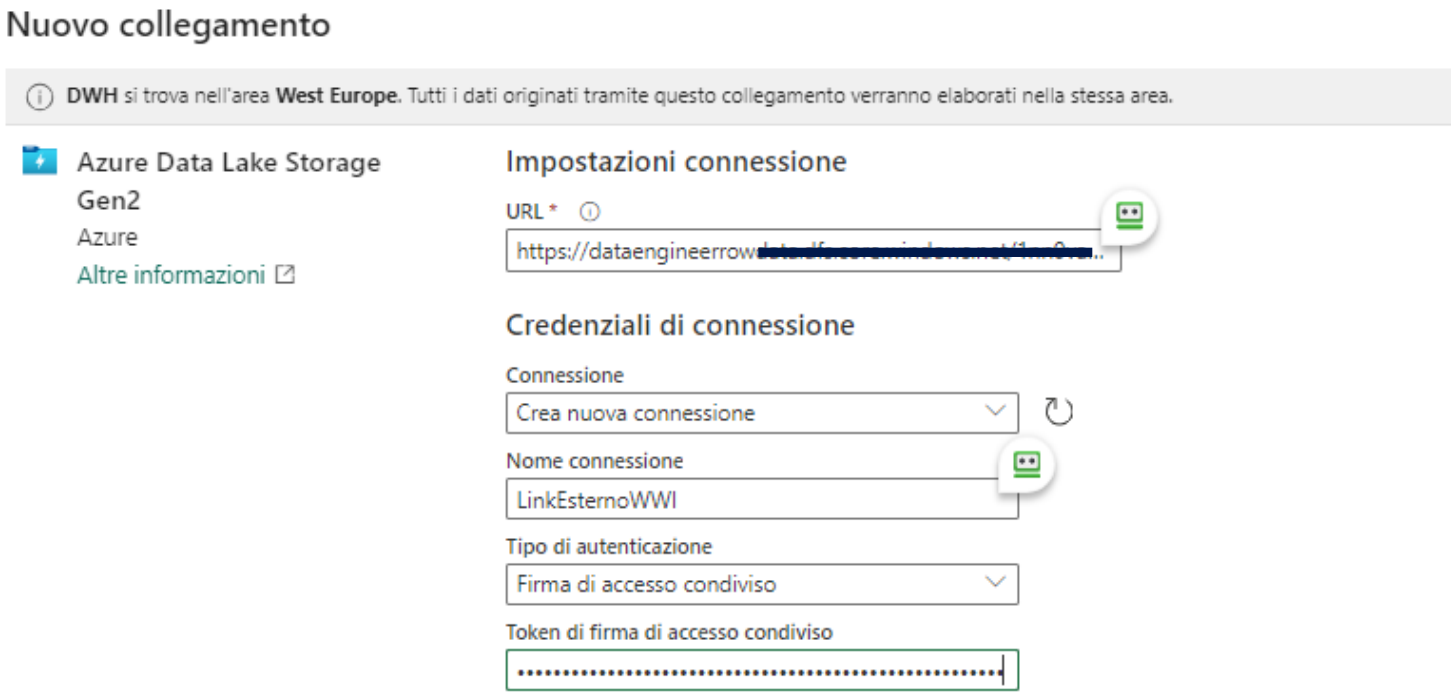






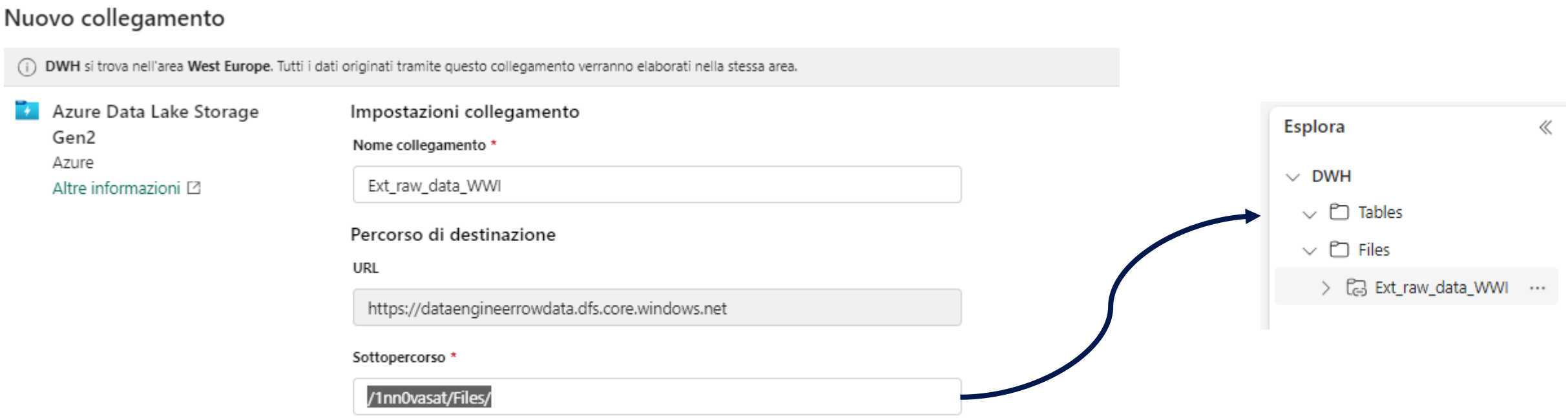

```
https://dataengineerrowdata.dfs.core.windows.net
/1nn0vasat
```

Demo - Shortcut configuration for bronze layer (part 2)



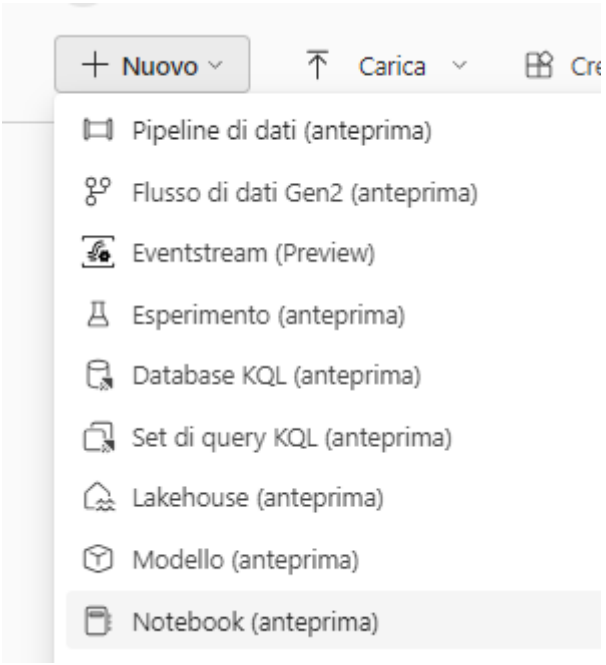
- https://dataengineerrowd[redacted]
- LinkEsternoWWI

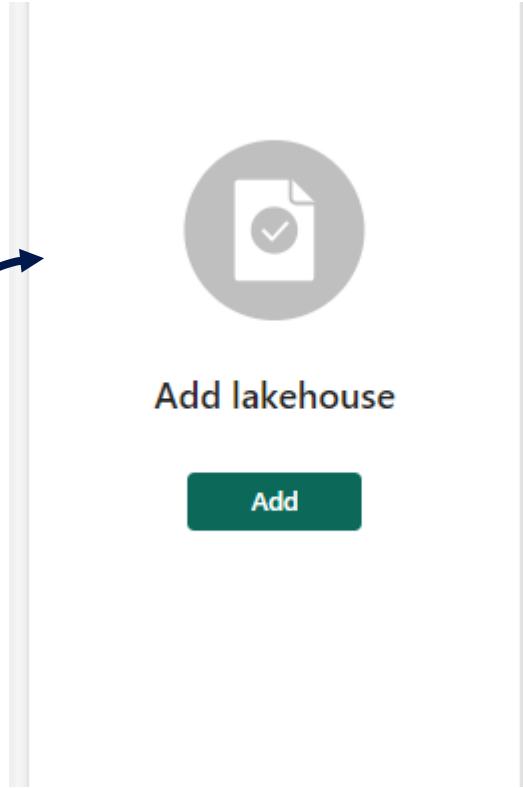
Demo - Shortcut configuration for bronze layer (part 3)

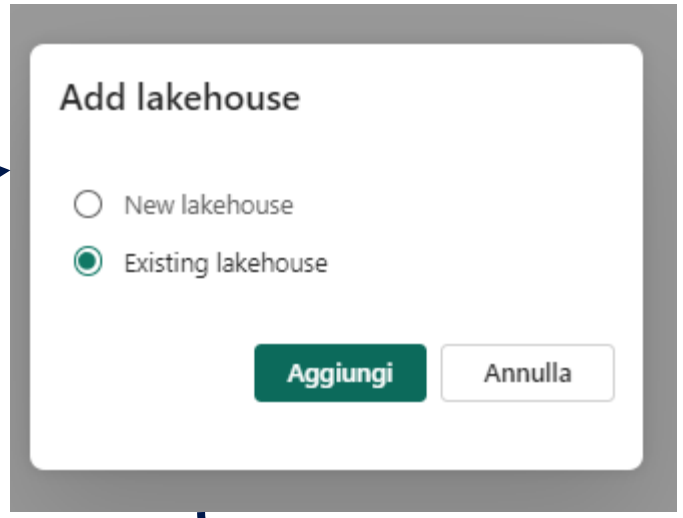


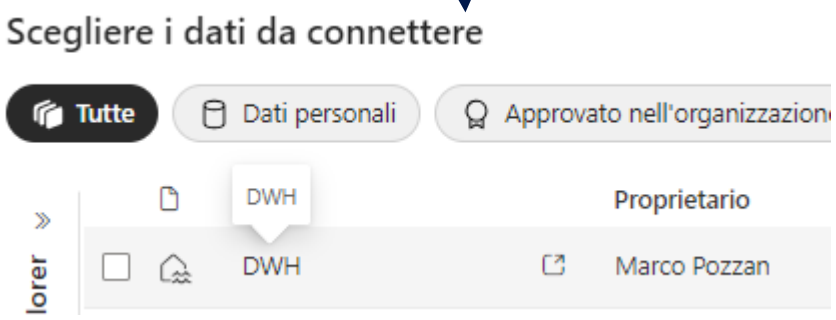
- Ext_raw_data_WWI
- /1nn0vasat/Files/

Demo - Load silver layer with Notebook





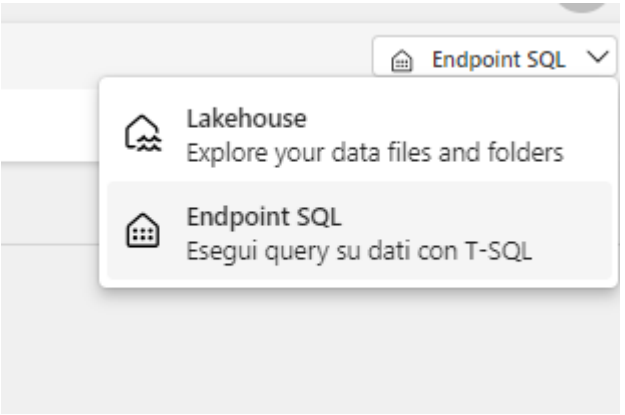




Demo – Load silver layer with Notebook

```
spark.conf.set("spark.sql.parquet.vorder.enabled", "true")
spark.conf.set("spark.microsoft.delta.optimizeWrite.enabled", "true")
#spark.conf.set("spark.microsoft.delta.optimizeWrite.binSize", "1073741824")


#Tabella City
df = spark.read.format("csv").option("delimiter", "|").option("header","true").load('Files/Ext_raw_data_WWI/Application_Cities.csv')
df.write.mode("overwrite").format("delta").saveAsTable("silver_ApplicationCities")

#Tabella Countries
df = spark.read.format("csv").option("delimiter", "|").option("header","true").load('Files/Ext_raw_data_WWI/Application_Countries.csv')
df.write.mode("overwrite").format("delta").saveAsTable("silver_ApplicationCountries")

#Tabella DeliveryMethods
df = spark.read.format("csv").option("delimiter", "|").option("header","true").load('Files/Ext_raw_data_WWI/Application_DeliveryMethods.csv')
df.write.mode("overwrite").format("delta").saveAsTable("silver_ApplicationDeliveryMethods ")

#Tabella People
df = spark.read.format("csv").option("delimiter", "|").option("header","true").load('Files/Ext_raw_data_WWI/Application_People.csv')
df.write.mode("overwrite").format("delta").saveAsTable("silver_ApplicationPeople")

#Tabella StateProvinces
df = spark.read.format("csv").option("delimiter", "|").option("header","true").load('Files/Ext_raw_data_WWI/Application_StateProvinces.csv')
df.write.mode("overwrite").format("delta").saveAsTable("silver_ApplicationStateProvinces")

#Tabella SupplierCategories
df = spark.read.format("csv").option("delimiter", "|").option("header","true").load('Files/Ext_raw_data_WWI/Purchasing_SupplierCategories.csv')
df.write.mode("overwrite").format("delta").saveAsTable("silver_SupplierCategories")

#Tabella Suppliers
df = spark.read.format("csv").option("delimiter", "|").option("header","true").load('Files/Ext_raw_data_WWI/Purchasing_Suppliers.csv')
df.write.mode("overwrite").format("delta").saveAsTable("silver_PurchasingSuppliers")

#Tabella PurchasingSuppliersCategories
df = spark.read.format("csv").option("delimiter", "|").option("header","true").load('Files/Ext_raw_data_WWI/Purchasing_SupplierCategories.csv')
df.write.mode("overwrite").format("delta").saveAsTable("silver_PurchasingSupplierCategories")
```

Demo - Test table

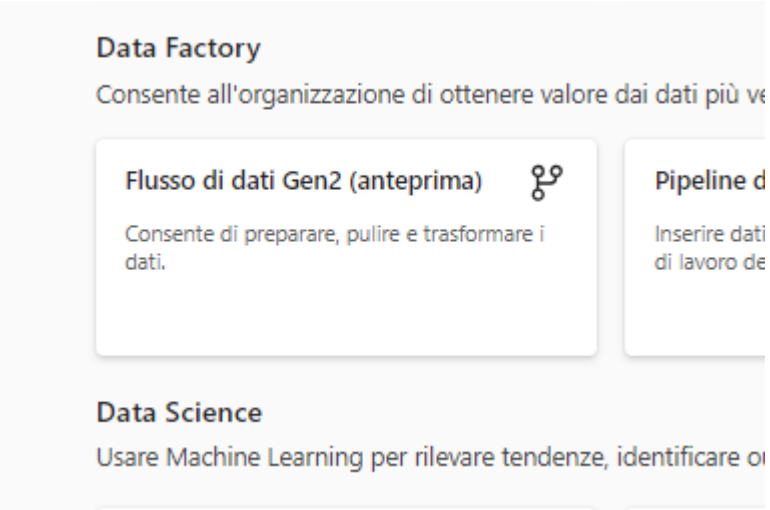


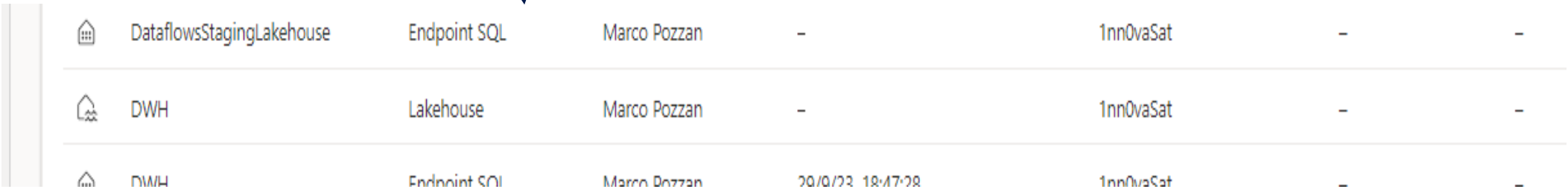

```sql
SELECT YEAR(SO.OrderDate) AS OrderDateYear,
COUNT(SO.OrderDate) AS TotalOrderCount
FROM [DWH].[dbo].[silver_salesorder] SO
GROUP BY YEAR(SO.OrderDate);
```

```sql
SELECT ISNULL(C.ColorName,'No Colour') AS ColourName,
SUM(cast(SOL.Quantity as int)) AS TotalOrderLineQuantity,
SUM(cast(SOL.UnitPrice as numeric)) AS TotalOrderLineUnitPrice
FROM [DWH].[dbo].[silver_salesorderline]  SOL
INNER JOIN [DWH].[dbo].[silver_stockitems] SI ON SI.StockItemID = SOL.StockItemID
LEFT JOIN [DWH].[dbo].[silver_colors] C ON C.ColorID = SI.ColorID
GROUP BY ISNULL(C.ColorName,'No Colour’);
```

```sql
SELECT
    YEAR(SO.OrderDate) AS OrderDateYear,
    SC.SupplierCategoryName,
    SUM(cast(SOL.Quantity as int)) AS TotalOrderLineQuantity,
    SUM(cast(SOL.UnitPrice as numeric)) AS TotalOrderLineUnitPrice
FROM [DWH].[dbo].[silver_salesorderline] SOL
INNER JOIN [DWH].[dbo].[silver_salesorder] SO ON SO.OrderID = SOL.OrderID
INNER JOIN [DWH].[dbo].[silver_stockitems] SI ON SI.StockItemID = SOL.StockItemID
INNER JOIN [DWH].[dbo].[silver_suppliers] S ON SI.SupplierID = S.SupplierID
INNER JOIN [DWH].[dbo].[silver_categories] SC ON SC.SupplierCategoryID = S.SupplierCategoryID
GROUP BY YEAR(SO.OrderDate),
        SC.SupplierCategoryName;
```

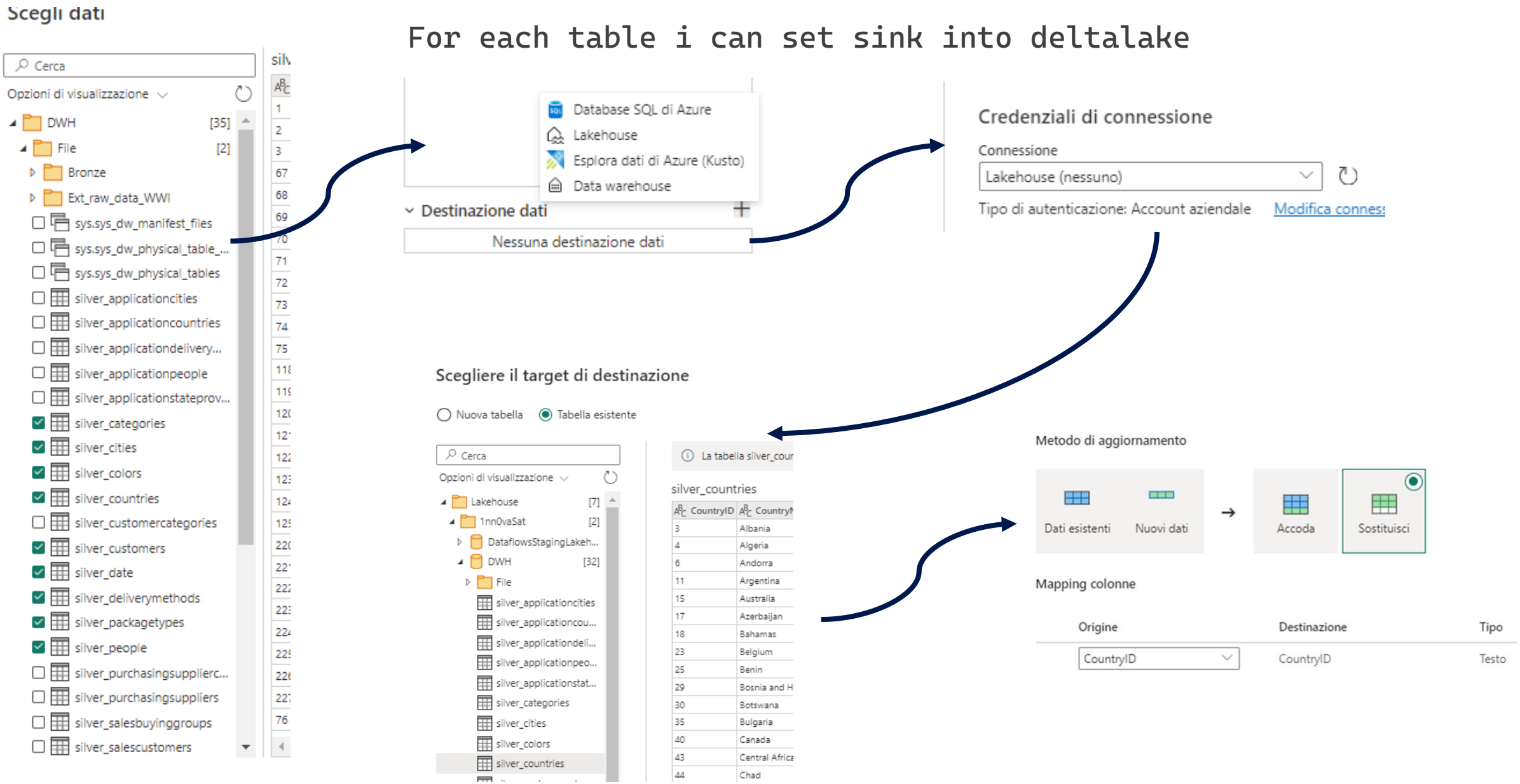
Demo – Load silver layer with DataFlow (Part 1)



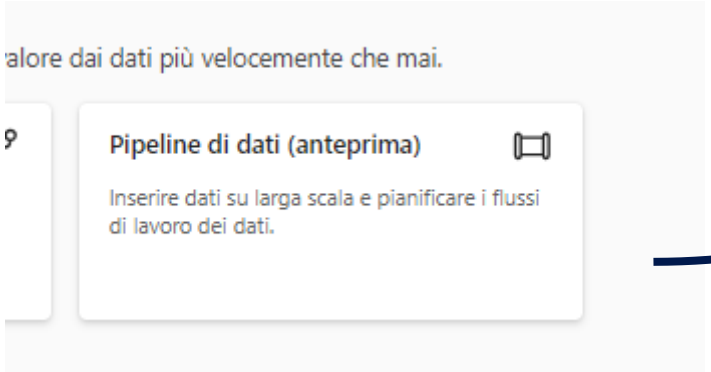


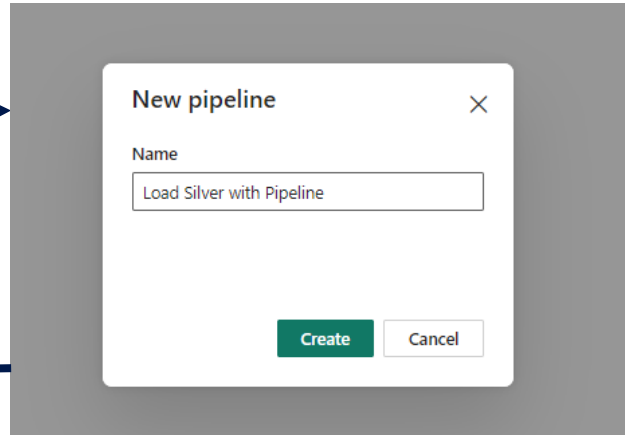
DataflowsStagingLakehouse	Endpoint SQL	Marco Pozzan	–	1nn0vaSat	–	–
DWH	Lakehouse	Marco Pozzan	–	1nn0vaSat	–	–
DWH	Endpoint SQL	Marco Pozzan	29/9/23, 18:47:28	1nn0vaSat	–	–

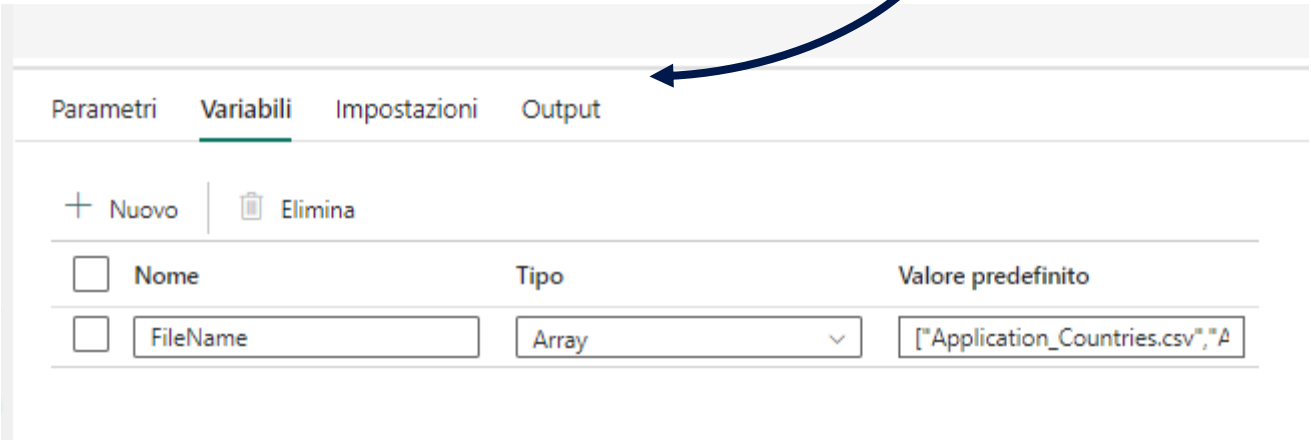
Demo - Load silver layer with DataFlow (Part 1)



Demo - Load silver layer with Pipeline (Part 1 define variables)



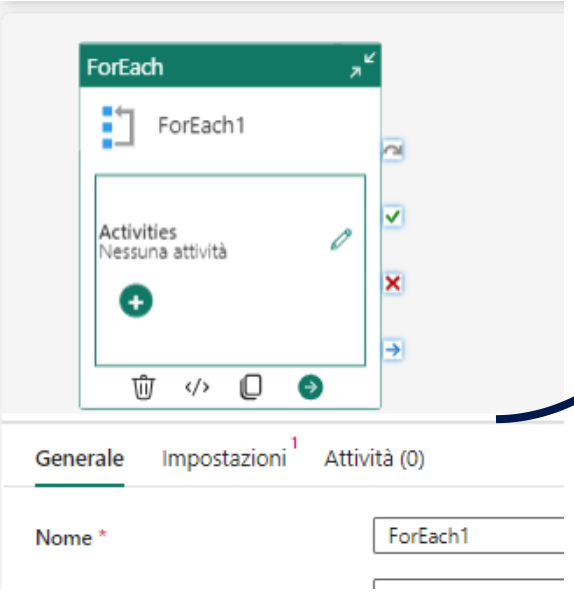


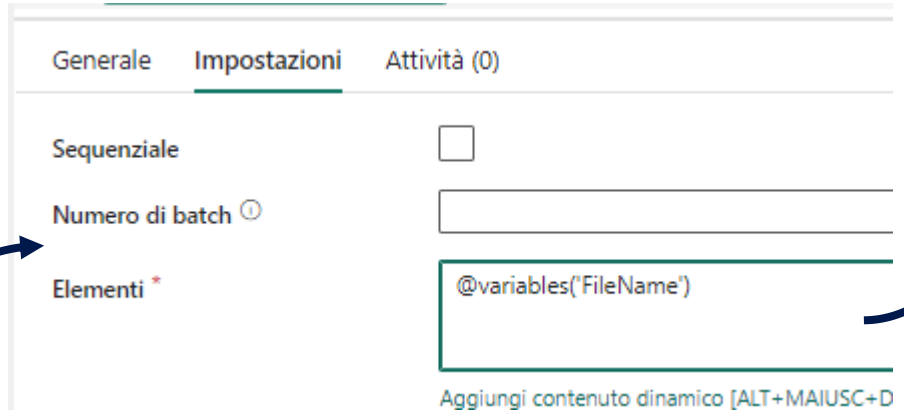


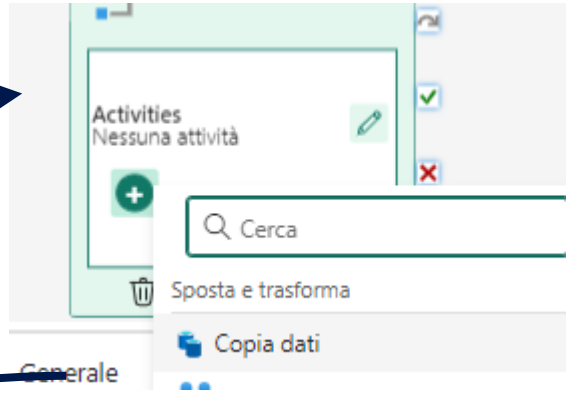
Set variables with this array

```
["Countries","DeliveryMethods","People","StateProvinces","SupplierCategories","Suppliers","BuyingGroups","CustomerCategories","Customers","Colors","PackageTypes","StockItems","Cities","Date"]
```

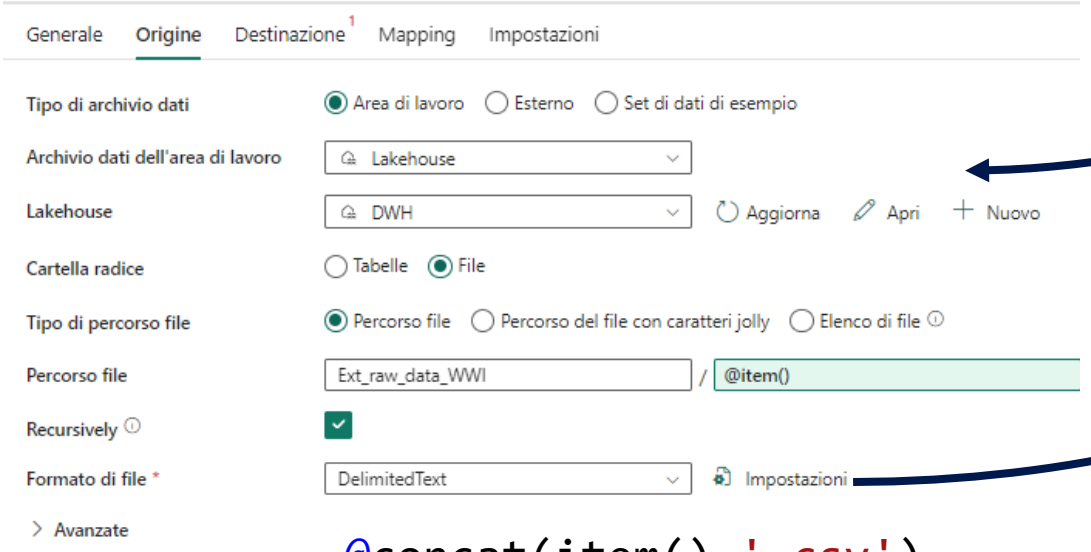
Demo - Load silver layer with Pipeline (Part 2 copy activity source)



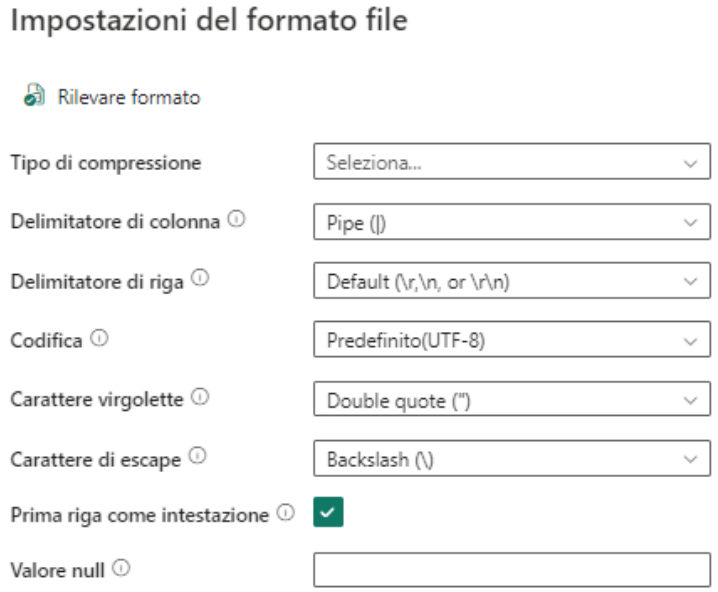




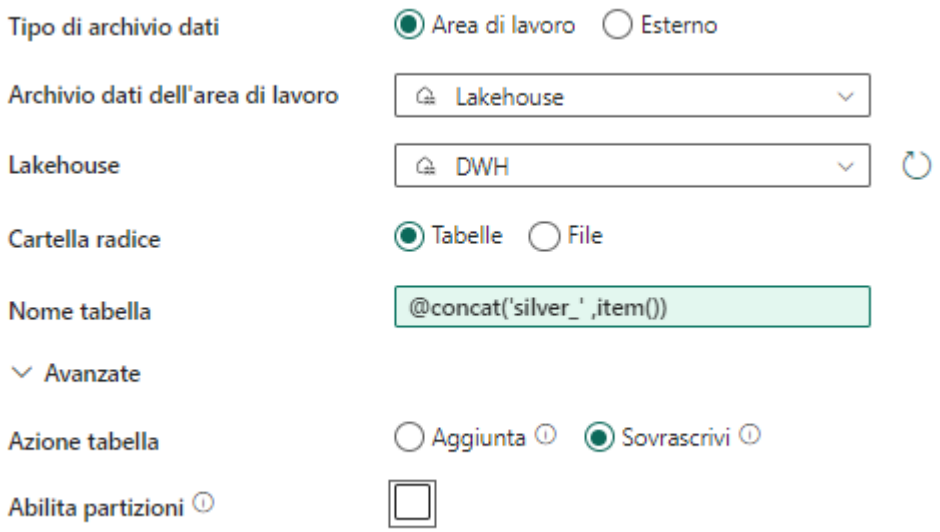

```
@variables('FileName')
```




```
@concat(item(),'.csv')
```

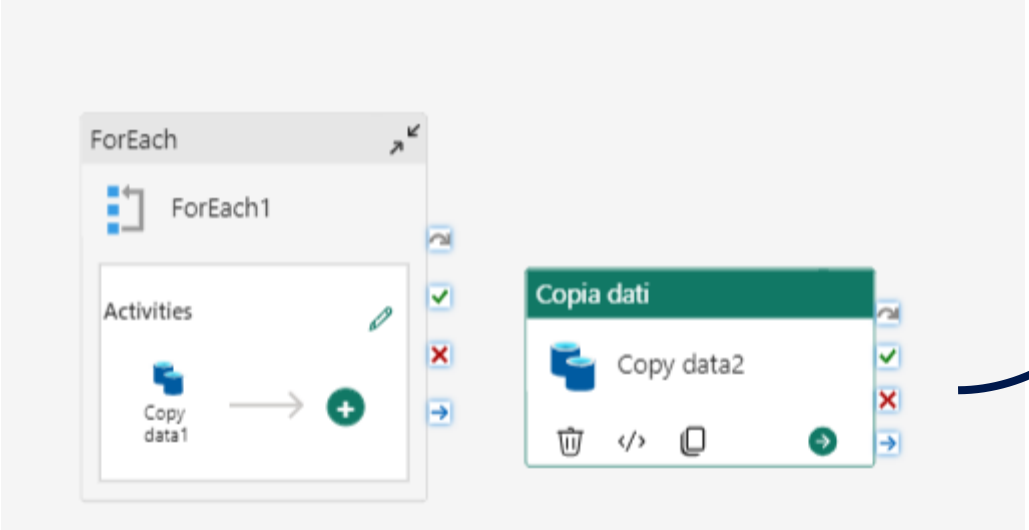


Demo - Load silver layer with Pipeline (Part 3 copy activity sink)




```
@concat('silver_' ,item())
```

Demo - Load silver layer with Pipeline (Part 4 copy activity source)



Generale | Origine | Destinazione[1] | Mapping | Impostazioni

Tipo di archivio dati	(●) Area di lavoro ( ) Esterno ( ) Set di dati di esempio
Archivio dati dell'area di lavoro	Lakehouse
Lakehouse	DWH — Aggiorna — Apri — Nuovo
Cartella radice	( ) Tabelle (●) File
Tipo di percorso file	( ) Percorso file (●) Percorso del file con caratteri jolly ( ) Elenco di file
Percorsi con caratteri jolly	Ext_raw_data_WWI/SalesOrder/* / * — Anteprima dati
Recursively	[x]
Formato di file *	DelimitedText — Impostazioni
> Avanzate	

Ext_raw_data_WWI/SalesOrder/*

Impostazioni del formato file

Rilevare formato

Tipo di compressione	Seleziona...
Delimitatore di colonna	Pipe (|)
Delimitatore di riga	Default (\r,\n, or \r\n)
Codifica	Predefinito(UTF-8)
Carattere virgolette	Double quote (")
Carattere di escape	Backslash (\)
Prima riga come intestazione	[x]
Valore null	

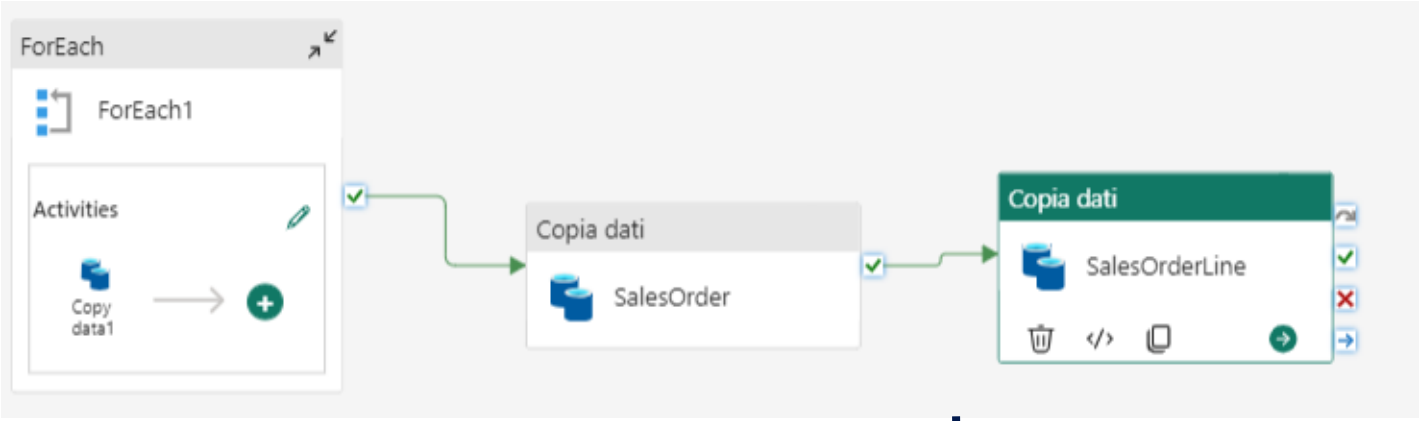
Demo - Load silver layer with Pipeline (Part 5 copy activity sink)

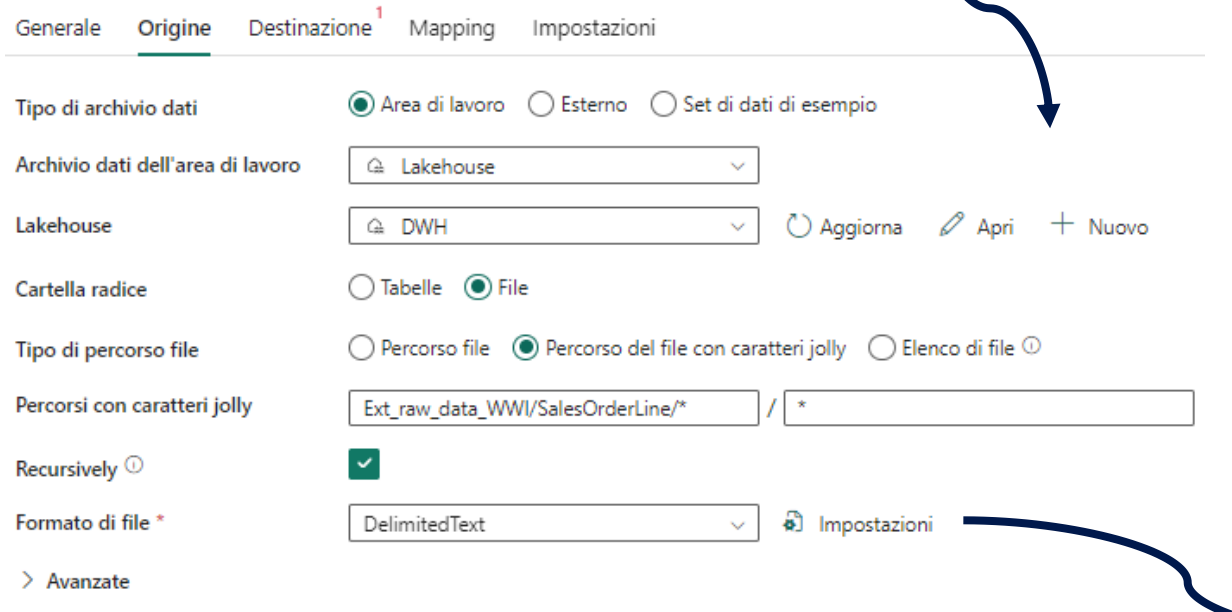
Generale | Origine | **Destinazione** | Mapping | Impostazioni

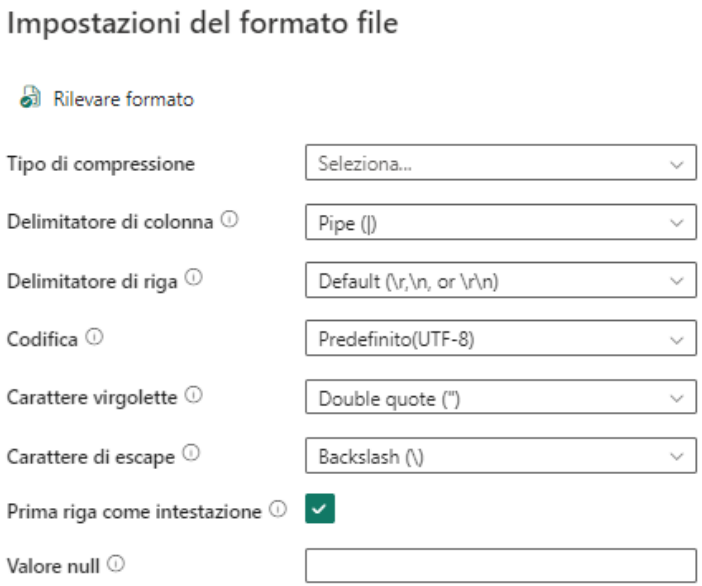
Tipo di archivio dati	(●) Area di lavoro ( ) Esterno
Archivio dati dell'area di lavoro	Lakehouse
Lakehouse	DWH Aggiorna Apri + Nuovo
Cartella radice	(●) Tabelle ( ) File
Nome tabella *	silver_salesorder Aggiorna Anteprima dati + Nuovo

> Avanzate

Demo - Load silver layer with Pipeline (Part 6 copy activity source)

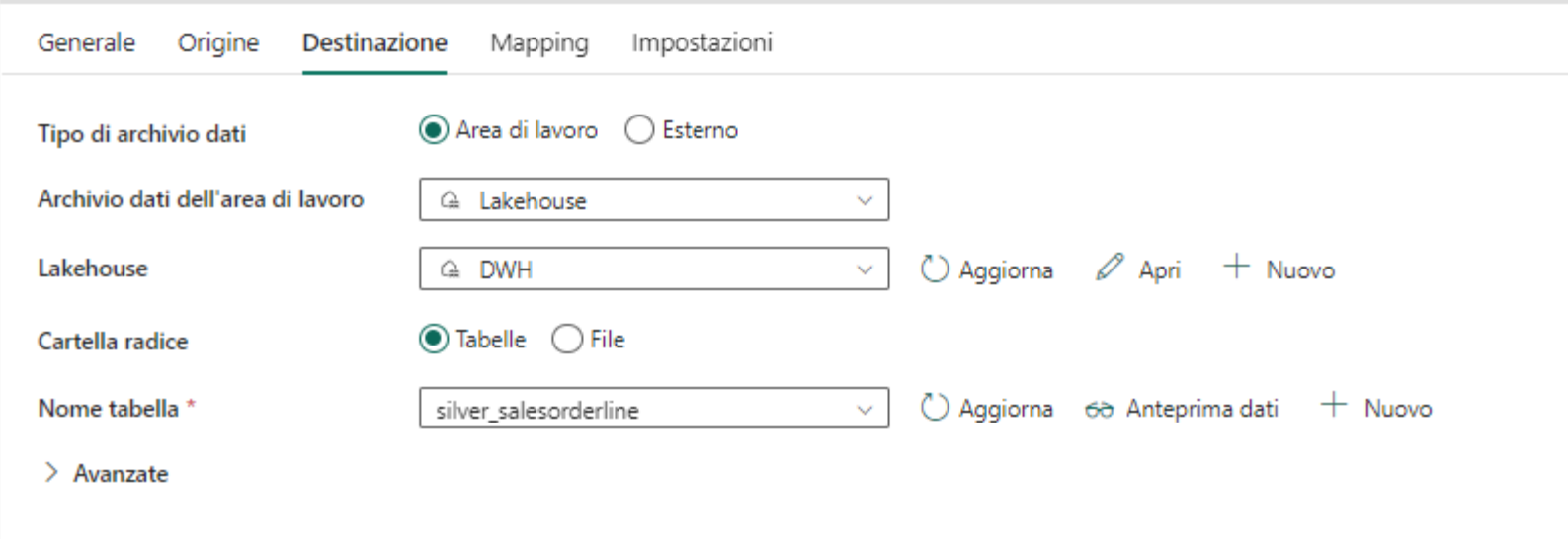




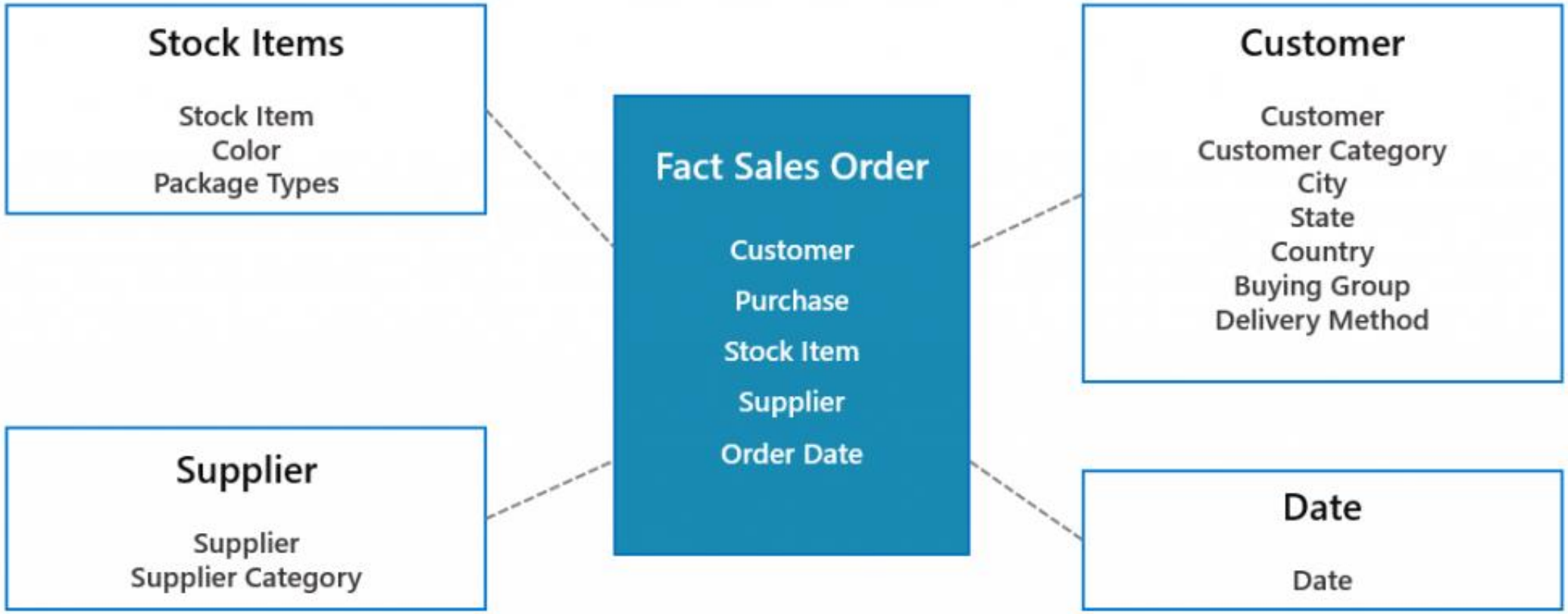



```
Ext_raw_data_WWI/SalesOrderLine/*
```

Demo - Load silver layer with Pipeline (Part 7 copy activity sink)



Demo - Load gold layer with sql queries



Demo - Load gold layer with sql queries

```sql
%%sql
create or replace table gold_DimCustomer
using delta PARTITIONED BY (CustomerID)
SELECT CAST(ROW_NUMBER() OVER(ORDER BY C.CustomerID) AS INT) AS CustomerKey,
        CAST(C.CustomerID AS INT) AS CustomerID,
        C.CustomerName,
        CC.CustomerCategoryName,
        BG.BuyingGroupName,
        DM.DeliveryMethodName,
        DC.CityName AS DeliveryCityName,
        DSP.StateProvinceName AS DeliveryStateProvinceName,
        DSP.SalesTerritory AS DeliverySalesTerritory,
        DCO.CountryName AS DeliveryCountry,
        DCO.Continent AS DeliveryContinent,
        DCO.Region AS DeliveryRegion,
        DCO.Subregion AS DeliverySubregion,
        CAST('2013-01-01' AS DATE) AS ValidFromDate
FROM silver_SalesCustomers C
LEFT JOIN silver_CustomerCategories CC On CC.CustomerCategoryID = C.CustomerCategoryID
LEFT JOIN silver_ApplicationCities  DC ON DC.CityID = C.DeliveryCityID
LEFT JOIN silver_ApplicationStateProvinces  DSP ON DSP.StateProvinceID = DC.StateProvinceID
LEFT JOIN silver_ApplicationCountries DCO ON DCO.CountryID = DSP.CountryID
LEFT JOIN silver_SalesBuyingGroups BG ON BG.BuyingGroupID = C.BuyingGroupID
LEFT JOIN silver_ApplicationDeliveryMethods DM ON DM.DeliveryMethodID = C.DeliveryMethodID
```

```sql
%%sql
create or replace table gold_DimStockItem
using delta  PARTITIONED BY (StockItemID)
SELECT CAST(ROW_NUMBER() OVER(ORDER BY SI.StockItemID) AS SMALLINT) AS StockItemKey,
CAST(SI.StockItemID AS SMALLINT) AS StockItemID,
SI.StockItemName,
SI.LeadTimeDays,
C.ColorName,
OP.PackageTypeName AS OuterPackageTypeName,
CAST('2013-01-01' AS DATE) AS ValidFromDate,
CAST(SI.SupplierID AS TINYINT) AS SupplierID
FROM silver_StockItems SI
LEFT JOIN silver_Colors C ON C.ColorID = SI.ColorID
LEFT JOIN silver_PackageTypes OP ON OP.PackageTypeID = SI.OuterPackageID
```

```sql
%%sql
create or replace table gold_DimSupplier
using delta PARTITIONED BY (SupplierKey)
SELECT CAST(ROW_NUMBER() OVER(ORDER BY S.SupplierID) AS TINYINT) AS SupplierKey,
CAST(S.SupplierID AS TINYINT) AS SupplierID,
S.SupplierName,
SC.SupplierCategoryName,
CAST('2013-01-01' AS DATE) AS ValidFromDate
FROM silver_Suppliers S
LEFT JOIN silver_SupplierCategories SC ON SC.SupplierCategoryID = S.SupplierCategoryID
```

```sql
%%sql
create or replace table gold_DimDate
using delta PARTITIONED DateKey
SELECT CAST(DateKey AS INT) AS DateKey,
        CAST(Date AS DATE) AS Date,
        CAST(Day AS TINYINT) AS Day,
        CAST(WeekDay AS TINYINT) AS WeekDay,
        WeekDayName,
        CAST(Month AS TINYINT) AS Month,
        MonthName,
        CAST(Quarter AS TINYINT) AS Quarter,
        CAST(Year AS SMALLINT) AS Year
        from silver_Date;
```

Demo - Create Serving Layer