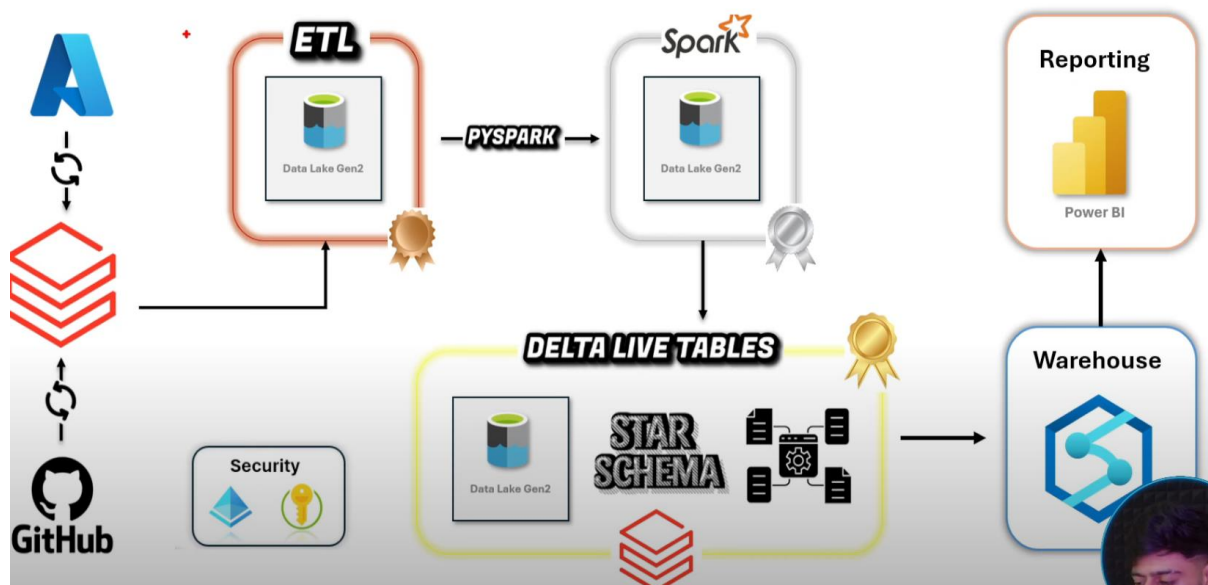


End to End Databricks Project (Retail)



Ingest data from Azure & GitHub, ingest incremental data(including idempotency) in Bronze using Spark Structured Streaming and automated Ingestion Pipeline in ADB.

Benefits of Parquet over csv → Compression, Parquet holds schema at the footer of file

Create 5 containers in Storage account → source, bronze, silver, gold, metastore(for UM)

Upload only first day files in source folder

Authentication method: Access key (Switch to Microsoft Entra user account)

Location: source

Search blobs by prefix (case-sensitive)

Name	Modified	Access tier
<input type="checkbox"/> customers	26/4/2025, 3:03:24 pm	
<input type="checkbox"/> orders	26/4/2025, 3:02:44 pm	
<input type="checkbox"/> products	26/4/2025, 3:03:42 pm	
<input type="checkbox"/> regions	26/4/2025, 3:04:07 pm	

While creating ADB instance, Managed Resource Group name is not a mandate if we are using UC. Whenever we create an ADB instance, we will get one UC for free.(Better to use self created ones)

Compute

All-purpose compute Job compute SQL warehouses **Vector Search** Pools Policies Apps

Create

Status Name Type Indexes Creator

Used by Data Scientists → translated into multiple dimensions

Compute

All-purpose compute Job compute SQL warehouses Vector Search **Policies** Apps

Filter policies Create policy

Name	Definitions	Policy Family	Compute type
Job Compute	9	Job Compute	job
Legacy Shared Compute	13	Legacy Shared Compute	all-purpose
Personal Compute	11	Personal Compute	all-purpose
Power User Compute	12	Power User Compute	all-purpose
Shared Compute	13	Shared Compute	all-purpose

Prebuilt policies

Create a UM, a Catalog, 4 External Locations(source, bronze, silver, gold)
Create bronze schema, will pull raw data over here

databricks Search data, notebooks, recents, and more... CTRL + P databricks_ete

Catalog Serverless Starter Warehouse Serverless S

Type to search...

- My organization
 - system
 - databricks_cata
 - bronze**
 - No data
 - default
 - information_schema
 - main
 - Delta Shares Received
 - samples
 - Legacy
 - hive_metastore

Catalog Explorer > databricks_cata > **bronze** Use with BI tools Create

Overview Details Permissions

About this schema
Owner Ansh Lamba

Tags
Add tags

Description
AI generate Add

Filter tables Tables 0 Volumes 0 Models 0 Functions 0 Sort

Bronze Ingestion

M1: For one time data load/no incremental/static load → Lets use a no code solution to pull data from source to bronze

Microsoft Azure **databricks** Search data, notebooks, recents, and more... CTRL + P

Marketplace

SQL

- SQL Editor
- Queries
- Dashboards
- Genie
- Alerts
- Query History
- SQL Warehouses

Data Engineering

- Data Ingestion**
- Pipelines

Machine Learning

- Playground
- Experiments
- Features

Add data
Get started by connecting to a data source or uploading a local file.

Search data sources

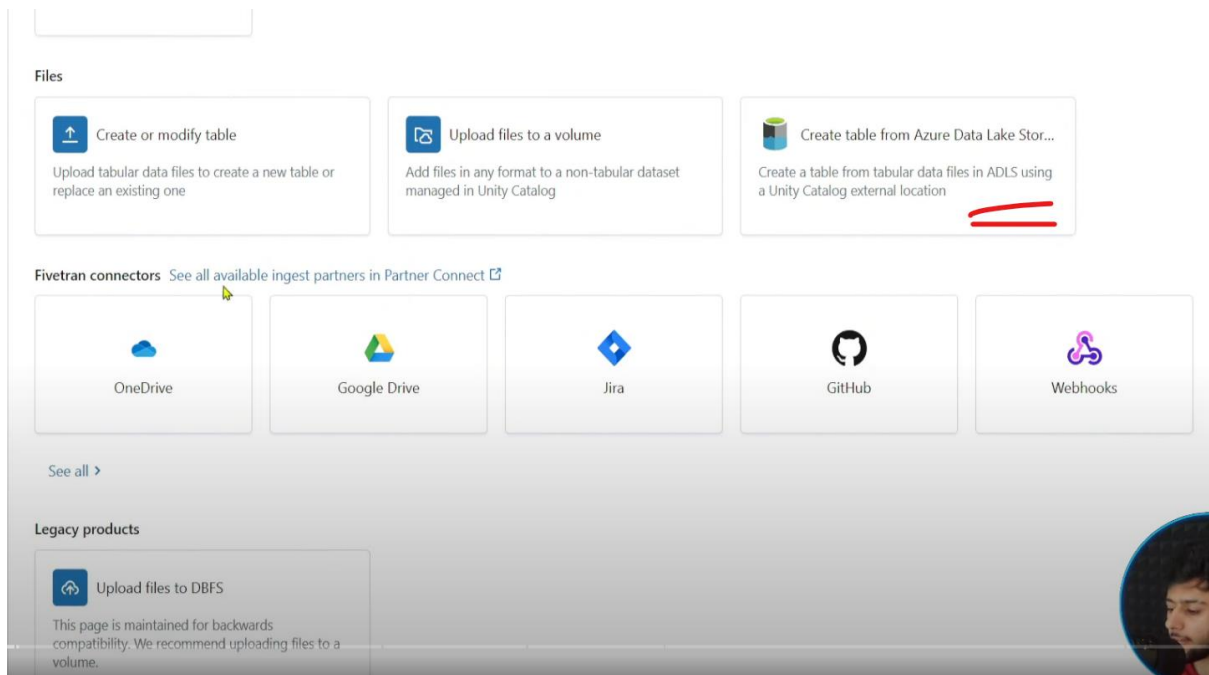
Databricks connectors

Salesforce

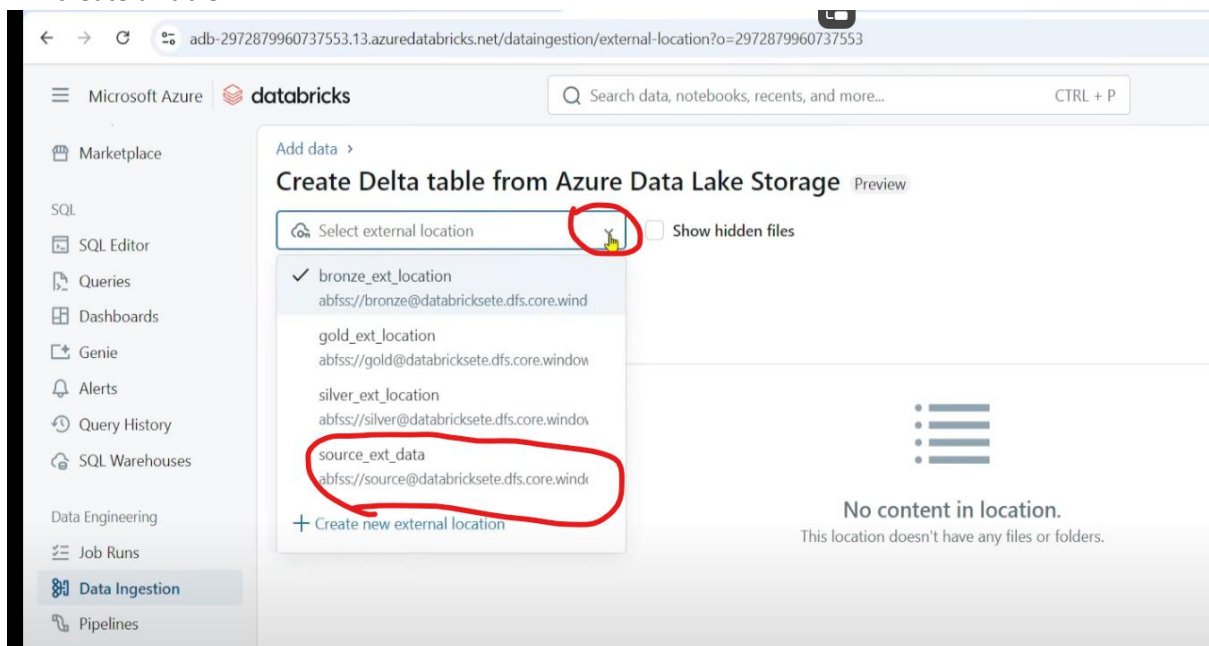
Files

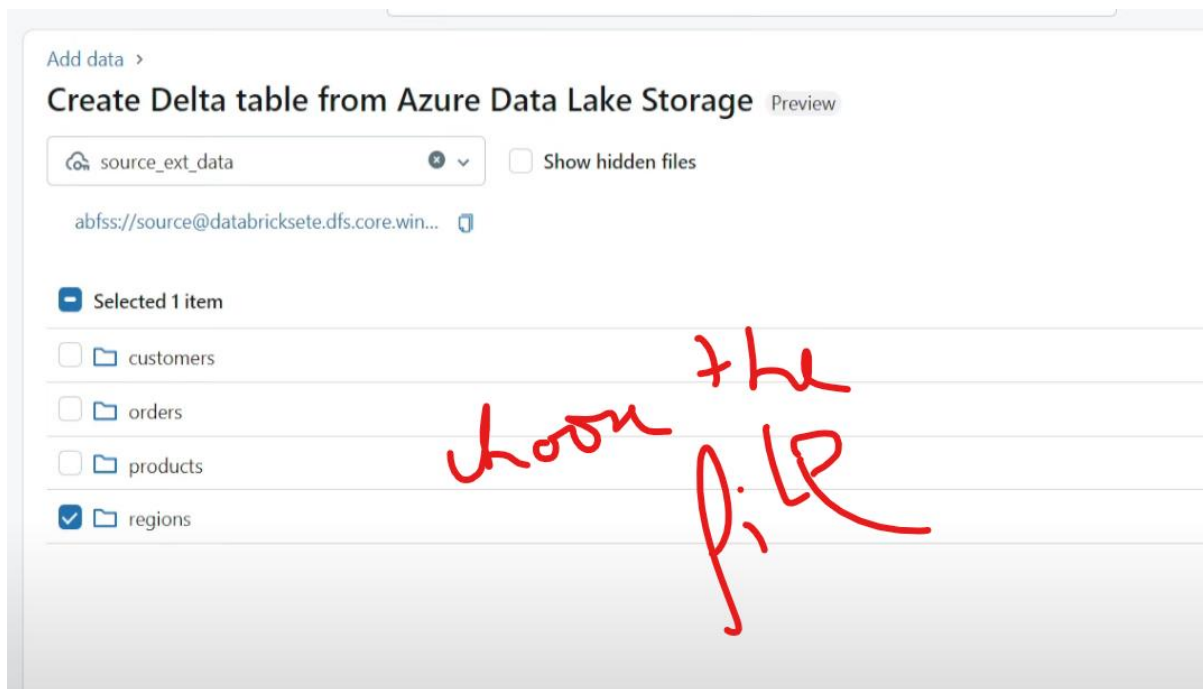
- Create or modify table
Upload tabular data files to create a new table or replace an existing one
- Upload files to a volume
Add files in any format to a non-tabular dataset managed in Unity Catalog
- Create table from Azure Data Lake Stor...
Create a table from tabular data files in ADLS using a Unity Catalog external location

Fivetran connectors See all available ingest partners in Partner Connect

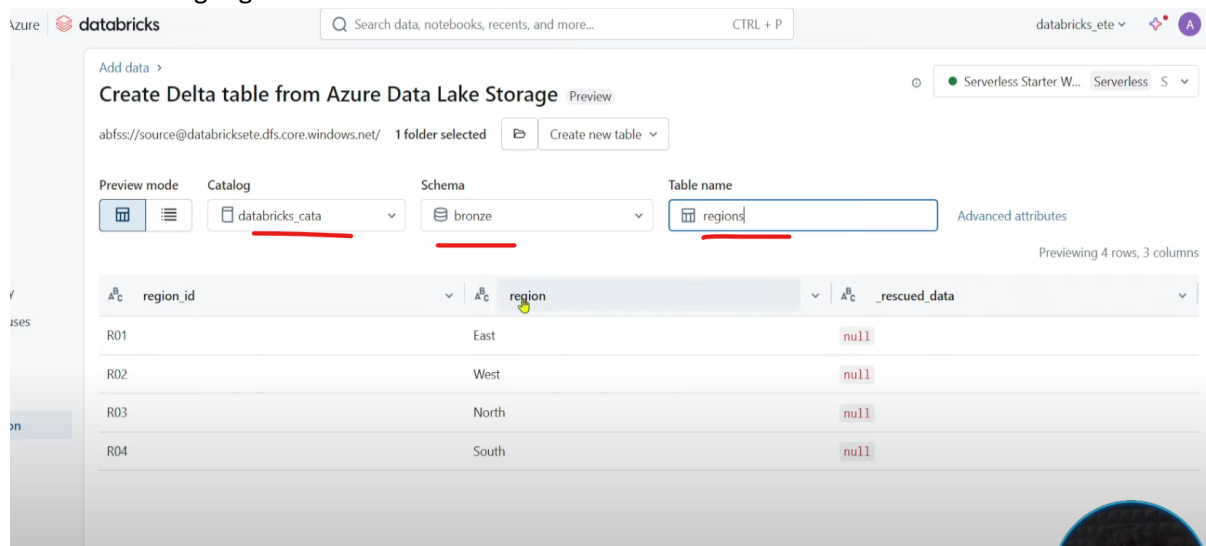


Clicking on that option will simply ingest data from DL and push that data to Managed Location and will create a Table

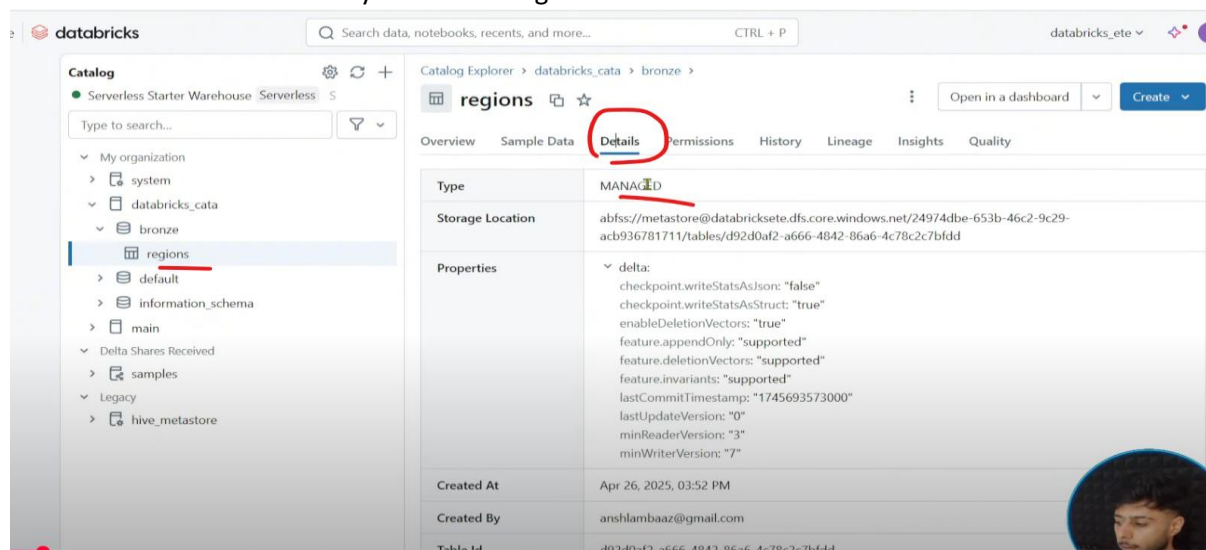


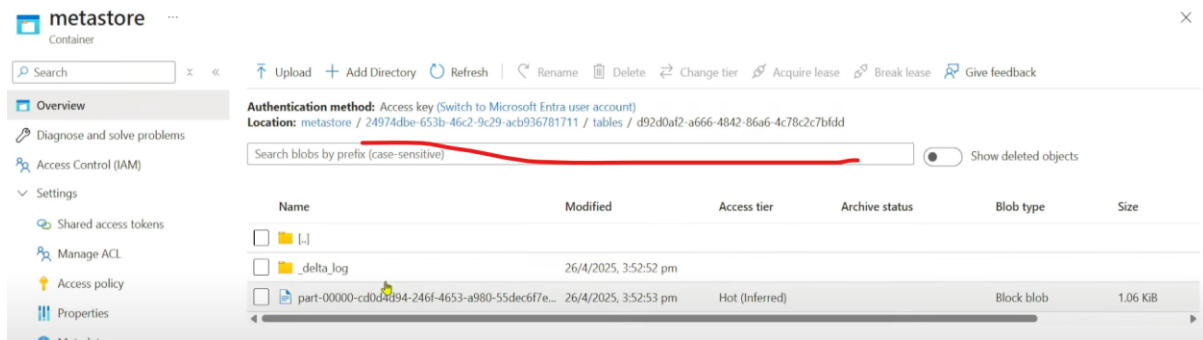


We are choosing regions as it is a static file with no incremental data



rescued data column is for any schema changes in future





M2: for all other files, choose Autoloader & also make it dynamic

Dynamic Capabilities

	+ Code	+ Text
<p>▶ ✓ Apr 27, 2025 (<1s)</p> <pre>dbutils.widgets.text("file_name", "")</pre>	2	
<p>▶ ✓ Apr 27, 2025 (<1s)</p> <pre>p_file_name = dbutils.widgets.get("file_name")</pre>	3	

Data Reading

<p>▶ ✓ Apr 27, 2025 (<1s)</p> <pre>df = spark.readStream.format("cloudFiles")\ .option("cloudFiles.format", "parquet")\ .option("cloudFiles.schemaLocation", f"abfss://bronze@databricksete.dfs.core.windows.net/checkpoint_{p_file_name}")\ .load(f"abfss://source@databricksete.dfs.core.windows.net/{p_file_name}")</pre> <p>df: pyspark.sql.dataframe.DataFrame = [order_id: string, customer_id: string ... 5 more fields]</p>	5	
--	---	--

DATA WRITING

<p>▶ Apr 27, 2025</p> <pre>df.writeStream.format("parquet")\ .outputMode("append")\ .option("checkpointLocation", f"abfss://bronze@databricksete.dfs.core.windows.net/checkpoint_{p_file_name}")\ .option("path", f"abfss://bronze@databricksete.dfs.core.windows.net/{p_file_name}")\ .trigger(once=True)\ .start()</pre> <p>☹ Stream stopped...</p> <p><pyspark.sql.streaming.query.StreamingQuery at 0x7f3f0c2f8890></p>	7	
<p>▶ ✓ Apr 27, 2025 (2s)</p> <pre>df = spark.read.format("parquet")\ .load(f"abfss://bronze@databricksete.dfs.core.windows.net/{p_file_name}")</pre> <p>display(df)</p>	8	

once = True in Triggers → simply read all the files not processed before, perform data loads for those data and immediately stops the Streaming query.

Bronze_LayerPython

FileEditViewRunHelpLast edit was now

file_name

orders

Authentication method: Access key (Switch to Microsoft Entra user account)

Location: bronze / orders

Search blobs by prefix (case-sensitive)

Show deleted objects

Name	Modified	Access tier	Archive status	Blob type	Size
[.]					
_spark_metadata	26/4/2025, 4:35:23 pm				
part-00000-ff5ec052-0fa8-411a-b27d-07b9c8d38...	26/4/2025, 4:35:29 pm	Hot (Inferred)		Block blob	125.91 KiB

UploadAdd DirectoryRefreshRenameDeleteChange tierAcquire leaseBreak leaseGive feedback

Authentication method: Access key (Switch to Microsoft Entra user account)

Location: bronze / checkpoint_orders / sources / 0

Search blobs by prefix (case-sensitive)

Show deleted objects

Name	Modified	Access tier	Archive status	Blob type	Size
[.]					
_tmp_path_dir	26/4/2025, 4:35:25 pm				
rocksdb	26/4/2025, 4:35:24 pm				
metadata	26/4/2025, 4:35:25 pm	Hot (Inferred)		Block blob	133 B

Best way to create a separate Notebook containing file names

parametersPython

FileEditViewRunHelp

Workspace

Databricks ETE Project

Bronze_LayerGold OrdersGold ProductsGold_CustomersparametersSilver_CustomersSilver_OrdersSilver_ProductsSilver_Regions

datasets = [

{

"file_name" : "orders"

},

{

"file_name" : "customers"

},

{

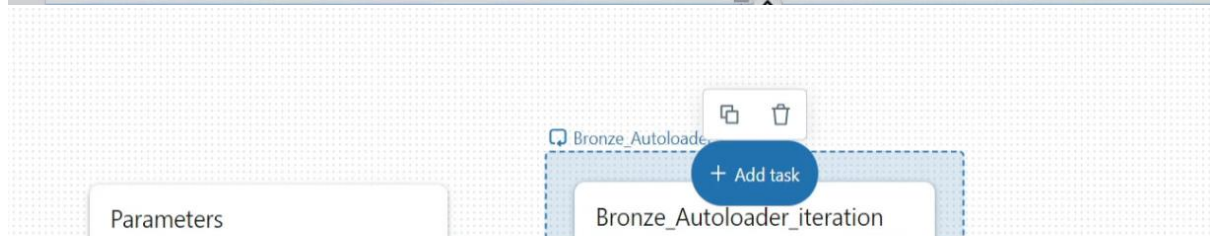
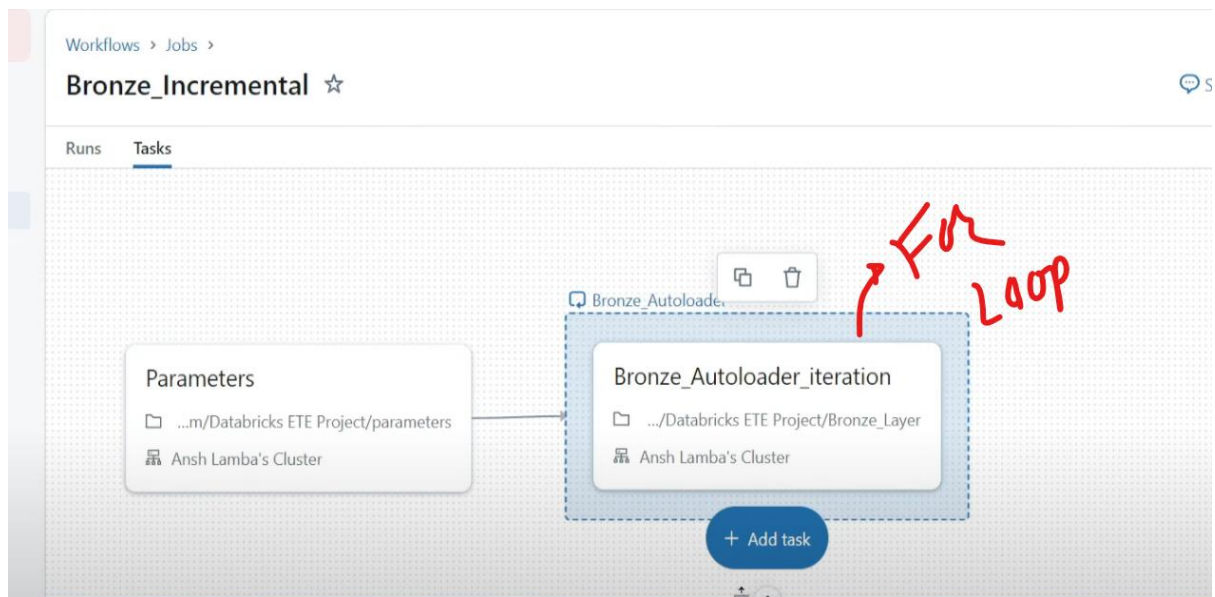
"file_name" : "products"

}

]

dbutils.jobs.taskValues.set("output_datasets", datasets)

Workflow orchestration → Create a job in Workflows



Task name* ⓘ Bronze_Autoloader

Type* Notebook ▾

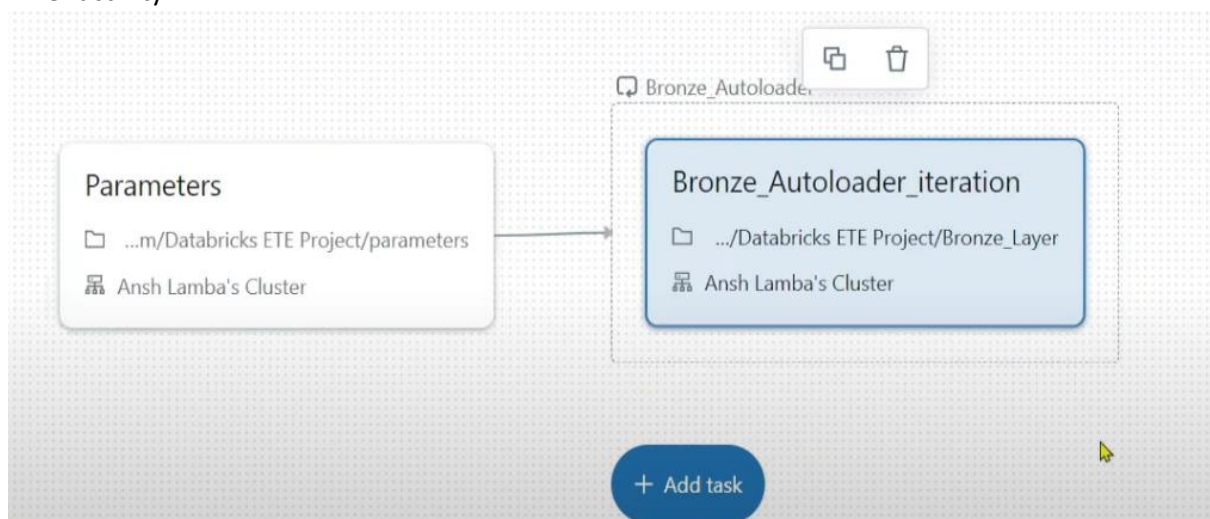
Inputs* ⓘ `{{tasks.Parameters.values.output_datasets}}` {}

Concurrency (optional) ⓘ

Depends on Parameters X ▾

Run if dependencies ⓘ All succeeded ▾

Inner activity



Dependent libraries ⓘ

+ Add

Parameters ⓘ

UIJSON

Key

Value

file_name

{{input.file_name}}

{ }

+ Add

Notifications ⓘ

+ Add

Authentication method: Access key (Switch to Microsoft Entra user account)
Location: bronze

Search blobs by prefix (case-sensitive)

	Name	Modified	Access tier	Archive status	Blob type	Size
<input type="checkbox"/>	checkpoint_customers	26/4/2025, 5:00:47 pm				
<input type="checkbox"/>	checkpoint_orders	26/4/2025, 4:30:27 pm				
<input type="checkbox"/>	checkpoint_products	26/4/2025, 5:01:01 pm				
<input type="checkbox"/>	customers	26/4/2025, 5:00:47 pm				
<input type="checkbox"/>	orders	26/4/2025, 4:35:23 pm				
<input type="checkbox"/>	products	26/4/2025, 5:01:01 pm				

Silver Transformations

Silver_OrdersPython▼Tabs: OFF▼☆

FileEditViewRunHelp

Workspace

← Databricks ETE Project

📁 Bronze_Layer

📁 Gold Orders

📁 Gold Products

📁 Gold_Customers

📁 parameters

📁 Silver_Customers

📁 Silver_Orders

📁 Silver_Products

📁 Silver_Regions

▶ Apr 27, 2025 (<1s)

from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql.window import Window

Data Reading

▶ Apr 27, 2025 (1s)

df = spark.read.format("parquet")\
 .load("abfss://bronze@databricksete.dfs.core.windows.net/orders")
df: pyspark.sql.dataframe.DataFrame = [order_id: string, customer_id: string ... 5 more fields]

Table▼+

	order_id	customer_id	product_id	order_date	quantity	total_amount	rescued_data
1	O00001	C00710	P0159	2023-03-22	3	2022.87	null
2	O00002	C00954	P0036	2023-06-30	2	3560.74	null
3	O00003	C01578	P0427	2023-11-06	3	5903.52	null
4	O00004	C00962	P0332	2024-02-27	3	4107.99	null
5	O00005	C00156	P0038	2024-10-13	5	5784.95	null
6	O00006	C00521	P0174	2023-05-17	5	407.75	null
7	O00007	C00982	P0352	2024-01-18	4	4907.64	null
8	O00008	C00976	P0172	2023-01-10	4	7037.88	null
9	O00009	C01001	P0238	2023-04-20	3	4076.97	null
10	O00010	C00702	P0258	2023-07-07	4	5695.64	null

▶ ✓ Apr 27, 2025 (<1s)

```
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)
```

▶ ✓ Apr 27, 2025 (<1s)

```
df = df.withColumnRenamed("_rescued_data", "rescued_data")
```

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

▶ ✓ Apr 27, 2025 (<1s)

```
df = df.drop("rescued_data")
df.display()
```

▶ ✓ Apr 27, 2025 (<1s)

8

```
df = df.withColumn("order_date", to_timestamp(col('order_date')))
df.display()
```

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

Table ▾ +

	^A _C order_id	^A _C customer_id	^A _C product_id	^C _D order_date	¹ ₃ quantity	¹ ₂ total_amount
1	O00001	C00710	P0159	2023-03-22T00:00:00.000+00:...	3	2022.87
2	O00002	C00954	P0036	2023-06-30T00:00:00.000+00:...	2	3560.74
3	O00003	C01578	P0427	2023-11-06T00:00:00.000+00:...	3	5903.52
4	O00004	C00962	P0332	2024-02-27T00:00:00.000+00:...	3	4107.99
5	O00005	C00156	P0038	2024-10-13T00:00:00.000+00:...	5	5784.95

▶ ✓ Apr 27, 2025 (<1s)

9

```
df = df.withColumn("year", year(col('order_date')))
df.display()
```

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

Table ▾ +

	^A _C order_id	^A _C customer_id	^A _C product_id	^C _D order_date	¹ ₃ quantity	¹ ₂ total_amount	¹ ₃ year
1	O00001	C00710	P0159	2023-03-22T00:00:00.000+00:...	3	2022.87	2023
2	O00002	C00954	P0036	2023-06-30T00:00:00.000+00:...	2	3560.74	2023
3	O00003	C01578	P0427	2023-11-06T00:00:00.000+00:...	3	5903.52	2023
4	O00004	C00962	P0332	2024-02-27T00:00:00.000+00:...	3	4107.99	2024
5	O00005	C00156	P0038	2024-10-13T00:00:00.000+00:...	5	5784.95	2024
6	O00006	C00521	P0174	2023-05-17T00:00:00.000+00:...	5	407.75	2023
7	O00007	C00982	P0352	2024-01-18T00:00:00.000+00:...	4	4907.64	2024
8	O00008	C00976	P0172	2023-01-10T00:00:00.000+00:...	4	7037.88	2023

▶

✓ Apr 27, 2025 (1s)

10

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

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

Table

+

	order_id	customer_id	product_id	order_date	quantity	total_amount	year	flag
1	O00957	C01449	P0498	2023-10-05T00:00:00.000+00:...	5	9952.9	2023	1
2	O01765	C01515	P0498	2023-05-11T00:00:00.000+00:...	5	9952.9	2023	1
3	O03502	C00805	P0498	2023-09-24T00:00:00.000+00:...	5	9952.9	2023	1
4	O03660	C01001	P0498	2023-10-08T00:00:00.000+00:...	5	9952.9	2023	1
5	O06790	C01819	P0498	2023-02-27T00:00:00.000+00:...	5	9952.9	2023	1
6	O03989	C00631	P0440	2023-11-20T00:00:00.000+00:...	5	9916.25	2023	2
7	O07763	C00471	P0440	2023-04-12T00:00:00.000+00:...	5	9916.25	2023	2
8	O03272	C01879	P0165	2023-02-15T00:00:00.000+00:...	5	9858.85	2023	3
9	O03746	C01713	P0165	2023-10-26T00:00:00.000+00:...	5	9858.85	2023	3
10	O08261	C00988	P0165	2023-11-29T00:00:00.000+00:...	5	9858.85	2023	3
11	O00412	C00748	P0427	2023-03-23T00:00:00.000+00:...	5	9839.2	2023	4
12	O04478	C01560	P0427	2023-10-04T00:00:00.000+00:...	5	9839.2	2023	4

▶

✓ Apr 27, 2025 (1s)

11

```
df1 = df1.withColumn("rank_flag",rank().over(Window.partitionBy("year").orderBy(desc("total_amount"))))
df1.display()
```

df1: pyspark.sql.dataframe.DataFrame = [order_id: string, customer_id: string ... 7 more fields]

Table

+

	order_id	customer_id	product_id	order_date	quantity	total_amount	year	flag	rank_flag
1	O00957	C01449	P0498	2023-10-05T00:00:00.000+00:...	5	9952.9	2023	1	1
2	O01765	C01515	P0498	2023-05-11T00:00:00.000+00:...	5	9952.9	2023	1	1
3	O03502	C00805	P0498	2023-09-24T00:00:00.000+00:...	5	9952.9	2023	1	1
4	O03660	C01001	P0498	2023-10-08T00:00:00.000+00:...	5	9952.9	2023	1	1
5	O06790	C01819	P0498	2023-02-27T00:00:00.000+00:...	5	9952.9	2023	1	1
6	O03989	C00631	P0440	2023-11-20T00:00:00.000+00:...	5	9916.25	2023	2	6
7	O07763	C00471	P0440	2023-04-12T00:00:00.000+00:...	5	9916.25	2023	2	6
8	O03272	C01879	P0165	2023-02-15T00:00:00.000+00:...	5	9858.85	2023	3	8
9	O03746	C01713	P0165	2023-10-26T00:00:00.000+00:...	5	9858.85	2023	3	8
10	O08261	C00988	P0165	2023-11-29T00:00:00.000+00:...	5	9858.85	2023	3	8
11	O00412	C00748	P0427	2023-03-23T00:00:00.000+00:...	5	9839.2	2023	4	11

▶

✓ Apr 27, 2025 (1s)

12

```
df1 = df1.withColumn("row_flag",row_number().over(Window.partitionBy("year").orderBy(desc("total_amount"))))
df1.display()
```

df1: pyspark.sql.dataframe.DataFrame = [order_id: string, customer_id: string ... 8 more fields]

Table

+

	order_id	customer_id	product_id	order_date	quantity	total_amount	year	flag	rank_flag	row_flag
1	O00957	C01449	P0498	2023-10-05T00:00:00.000+00:...	5	9952.9	2023	1	1	1
2	O01765	C01515	P0498	2023-05-11T00:00:00.000+00:...	5	9952.9	2023	1	1	2
3	O03502	C00805	P0498	2023-09-24T00:00:00.000+00:...	5	9952.9	2023	1	1	3
4	O03660	C01001	P0498	2023-10-08T00:00:00.000+00:...	5	9952.9	2023	1	1	4
5	O06790	C01819	P0498	2023-02-27T00:00:00.000+00:...	5	9952.9	2023	1	1	5
6	O03989	C00631	P0440	2023-11-20T00:00:00.000+00:...	5	9916.25	2023	2	6	6
7	O07763	C00471	P0440	2023-04-12T00:00:00.000+00:...	5	9916.25	2023	2	6	7
8	O03272	C01879	P0165	2023-02-15T00:00:00.000+00:...	5	9858.85	2023	3	8	8
9	O03746	C01713	P0165	2023-10-26T00:00:00.000+00:...	5	9858.85	2023	3	8	9
10	O08261	C00988	P0165	2023-11-29T00:00:00.000+00:...	5	9858.85	2023	3	8	10

Classes - OOP

▶ ✓ Apr 27, 2025 (<1s) 14

```
class windows:

    def dense_rank(self,df):

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

        return df_dense_rank

    def rank(self,df):

        df_rank = df.withColumn("rank_flag",rank().over(Window.partitionBy("year").orderBy(desc("total_amount"))))

        return df_rank

    def row_number(self,df):

        df_row_number = df.withColumn("row_flag",row_number().over(Window.partitionBy("year").orderBy(desc("total_amount"))))

        return df_row_number
```

▶ ✓ Apr 27, 2025 (<1s) 15

```
df_new = df
```

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

▶ ✓ Apr 27, 2025 (<1s) 16

```
df_new.display()
```

Table ▾ +

	^A _C order_id	^A _C customer_id	^A _C product_id	^A _C order_date	¹ ₃ quantity	¹ ₂ total_amount	¹ ₃ year
1	O00001	C00710	P0159	2023-03-22T00:00:00.000+00:...	3	2022.87	2023
2	O00002	C00954	P0036	2023-06-30T00:00:00.000+00:...	2	3560.74	2023
3	O00003	C01578	P0427	2023-11-06T00:00:00.000+00:...	3	5903.52	2023
4	O00004	C00962	P0332	2024-02-27T00:00:00.000+00:...	3	4107.99	2024
5	O00005	C00156	P0038	2024-10-13T00:00:00.000+00:...	5	5784.95	2024
6	O00006	C00521	P0174	2023-05-17T00:00:00.000+00:...	5	407.75	2023
7	O00007	C00982	P0352	2024-01-18T00:00:00.000+00:...	4	4907.64	2024

▶

✓ Apr 27, 2025 (<1s)

17

```
obj = windows()
```

▶

✓ Apr 27, 2025 (1s)

18

```
df_result = obj.dense_rank(df_new)
df_result.display()
```

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

Table +

	order_id	customer_id	product_id	order_date	quantity	total_amount	year	flag
1	O00957	C01449	P0498	2023-10-05T00:00:00.000+00:...	5	9952.9	2023	1
2	O01765	C01515	P0498	2023-05-11T00:00:00.000+00:...	5	9952.9	2023	1
3	O03502	C00805	P0498	2023-09-24T00:00:00.000+00:...	5	9952.9	2023	1
4	O03660	C01001	P0498	2023-10-08T00:00:00.000+00:...	5	9952.9	2023	1
5	O06790	C01819	P0498	2023-02-27T00:00:00.000+00:...	5	9952.9	2023	1
6	O03989	C00631	P0440	2023-11-20T00:00:00.000+00:...	5	9916.25	2023	2
7	O07763	C00471	P0440	2023-04-12T00:00:00.000+00:...	5	9916.25	2023	2
8	O03272	C01879	P0165	2023-02-15T00:00:00.000+00:...	5	9858.85	2023	3
9	O03746	C01713	P0165	2023-10-26T00:00:00.000+00:...	5	9858.85	2023	3
10	O08261	C00988	P0165	2023-11-29T00:00:00.000+00:...	5	9858.85	2023	3
11	O00412	C00748	P0427	2023-03-23T00:00:00.000+00:...	5	9839.2	2023	4
12	O04478	C01560	P0427	2023-10-04T00:00:00.000+00:...	5	9839.2	2023	4
13	O05528	C01500	P0427	2023-02-02T00:00:00.000+00:...	5	9839.2	2023	4

Data Writing

▶

✓ Apr 27, 2025 (11s)

20

```
df.write.format("delta").mode("overwrite").save("abfss://silver@databricksete.dfs.core.windows.net/orders")
```

▶

✓ Apr 27, 2025 (3s)

21

```
%sql
CREATE TABLE IF NOT EXISTS databricks_cata.silver.orders_silver
USING DELTA
LOCATION 'abfss://silver@databricksete.dfs.core.windows.net/orders'
```

OK

Waiting

```
%sql
CREATE SCHEMA databricks_cata.silver
```

Silver_CustomersPythonTabs: OFF☆
File Edit View Run Help

Workspace

Databricks ETE Project

- Bronze_Layer
- Gold Orders
- Gold Products
- Gold_Customers
- parameters
- Silver_Customers
- Silver_Orders
- Silver_Products
- Silver_Regions

▶ Apr 27, 2025 (<1s)

from pyspark.sql.functions import *
from pyspark.sql.types import *
import time

Data Reading

▶ Apr 27, 2025 (1s)
df = spark.read.format("parquet")\
| | | .load("abfss://bronze@databricksete.dfs.core.windows.net/customers")
▶ df: pyspark.sql.dataframe.DataFrame = [customer_id: string, first_name: string ... 5 more fields]

	customer_id	first_name	last_name	email	city	state	_rescued_data
1	C00001	Emily	Mooney	rushjeff@ryan.org	Johnsonmouth	MS	null
2	C00002	Andrea	Sellers	mccoikiara@kelly.com	Stephenfort	WY	null
3	C00003	Craig	Hayes	rebeccamiller@yahoo.com	South Stephenshire	LA	null
4	C00004	Bryan	Scott	lawrence05@campbell.info	Chrisland	ND	null
5	C00005	Sean	Vasquez	carrie45@yahoo.com	East Dennistown	RI	null
6	C00006	Kevin	Mccarthy	traceyramos@gmail.com	North Matthew	IN	null
7	C00007	Amanda	Doyle	scottallen@gmail.com	Joneshaven	VA	null
8	C00008	Paul	Campos	sullivanjeremy@horton-adams.com	South Nathanfurt	CT	null
9	C00009	Mary	Green	dennis03@yahoo.com	Kimberlyview	MD	null
10	C00010	James	Myers	charles58@murillo.net	West Hector	OK	null

▶ Apr 27, 2025 (<1s)

df = df.drop("_rescued_data")
df.display()

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

▶ Apr 27, 2025 (<1s)6

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

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

	customer_id	first_name	last_name	email	city	state	domains
1	C00001	Emily	Mooney	rushjeff@ryan.org	Johnsonmouth	MS	ryan.org
2	C00002	Andrea	Sellers	mccoikiara@kelly.com	Stephenfort	WY	kelly.com
3	C00003	Craig	Hayes	rebeccamiller@yahoo.com	South Stephenshire	LA	yahoo.com
4	C00004	Bryan	Scott	lawrence05@campbell.info	Chrisland	ND	campbell.info
5	C00005	Sean	Vasquez	carrie45@yahoo.com	East Dennistown	RI	yahoo.com

▶ ✓ Apr 27, 2025 (1s) 7

```
df.groupby("domains").agg(count("customer_id").alias("total_customers")).sort("total_customers",ascending=False).display()
```

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	johnson.com	5

< >

▶

```
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)
```


▶ ✓ Apr 27, 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()
```

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

Table ▾ +

	customer_id	email	city	state	domains	full_name
1	C00001	rushjeff@ryan.org	Johnsonmouth	MS	ryan.org	Emily Mooney
2	C00002	mccoykiara@kelly.com	Stephenfort	WY	kelly.com	Andrea Sellers
3	C00003	rebeccamiller@yahoo.com	South Stephenshire	LA	yahoo.com	Craig Hayes
4	C00004	lawrence05@campbell.info	Chrisland	ND	campbell.info	Bryan Scott
5	C00005	carrie45@yahoo.com	East Dennistown	RI	yahoo.com	Sean Vasquez
6	C00006	traceyramos@gmail.com	North Matthew	IN	gmail.com	Kevin Mccarthy
7	C00007	scottallen@gmail.com	Joneshaven	VA	gmail.com	Amanda Doyle
8	C00008	sullivanjeremy@horton-adams.com	South Nathanfurt	CT	horton-adams.com	Paul Campos

▶ ✓ Apr 27, 2025 (5s)

10

```
df.write.mode("overwrite").format("delta").save("abfss://silver@databricksete.dfs.core.windows.net/customers")
```

▶ ✓ Apr 27, 2025 (<1s)

11

```
%sql
CREATE TABLE IF NOT EXISTS databricks_cata.silver.customers_silver
USING DELTA
LOCATION 'abfss://silver@databricksete.dfs.core.windows.net/customers'
```

OK

Silver_ProductsPythonTabs: OFF☆
File Edit View Run Help

Workspace

← Databricks ETE Project

Bronze_Layer

Gold Orders

Gold Products

Gold_Customers

parameters

Silver_Customers

Silver_Orders

Silver_Products

Silver_Regions

▶ ✓ Apr 27, 2025 (<1s)

from pyspark.sql.functions import *
from pyspark.sql.types import *

Data Reading

▶ ✓ Apr 27, 2025 (6s)

df = spark.read.format("parquet")\
| | | | .load("abfss://bronze@databricksete.dfs.core.windows.net/products")

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

Table +						
	product_id	product_name	category	brand	price	rescued_data
1	P0001	Clearly Its	Beauty	Nike	1868.54	null
2	P0002	Production Clear	Beauty	Apple	587.13	null
3	P0003	Culture Coach	Home	Revlon	1599.24	null
4	P0004	Movement Part	Sports	LG	651.71	null
5	P0005	Fact Name	Clothing	Samsung	1861.78	null
6	P0006	Usually Stop	Toys	Adidas	936.36	null
7	P0007	Reveal Current	Sports	Adidas	1954.02	null
8	P0008	Force Language	Beauty	Puma	1251.26	null
9	P0009	Stage Leg	Clothing	Samsung	1247.15	null
10	P0010	Leader Than	Sports	Sony	975.53	null

▶ ✓ Apr 27, 2025 (<1s)

df = df.drop("_rescued_data")

▶ df: pyspark.sql.dataframe.DataFrame = [product_id: string, product_name: string ... 3 more fields]

▶ ✓ Apr 27, 2025 (<1s)

df.createOrReplaceTempView("products")

Silver_Products Python ☆

File Edit View Run Help Last edit was now

▶ Run all ● An

Catalog

Type to search...

For you All

- My organization
 - system
 - databricks_cata
 - bronze
 - regions
 - default
 - information_schema
 - main

Open in Catalog Explorer

Copy schema path

Add to favorites

	product_id	product_name	category	brand	price
1	P0001	Clearly Its	Beauty	Nike	1868.54
2	P0002	Production Clear	Beauty	Apple	587.13
3	P0003	Culture Coach	Home	Revlon	1599.24
4	P0004	Movement Part	Sports	LG	651.71
5	P0005	Fact Name	Clothing	Samsung	1861.78
		Usually Stop	Toys	Adidas	936.36
		Reveal Current	Sports	Adidas	1954.02
		Force Language	Beauty	Puma	1251.26
		Stage Leg	Clothing	Samsung	1247.15
10	P0010	Leader Then	Sports	Sony	975.53

Functions can be created inside Schema, we can reuse it (UDF is only till program level).

2 Types → 1. Scalar Function 2. Table Function

SQL way

Functions

```

▶ ✓ Apr 27, 2025 (1s)

%sql
CREATE OR REPLACE FUNCTION databricks_cata.bronze.discount_func(p_price DOUBLE)
RETURNS DOUBLE
LANGUAGE SQL
RETURN p_price * 0.90
  
```

OK

Catalog

Type to search...

For you All

- My organization
 - system
 - databricks_cata
 - bronze
 - Tables(1)
 - Functions(1)
 - discount_func
 - default
 - information_schema
 - main

This result is stored as _sqlidf and can be used in other Python cells.

```

▶ ✓ Just now (1s) 9

%sql
select product_id, price ,databricks_cata.bronze.discount_func(price) as discounted_price
FROM products

▶ (2) Spark Jobs

▶ _sqlidf: pyspark.sql.dataframe.DataFrame = [product_id: string, price: double ... 1 more field]
  
```

discount_func

databricks_cata.bronze

Owner Ansh Lamba

Updated 2 minutes ago

price
1681.686

▶ ▾ ✓ Apr 27, 2025 (1s)

```
%sql
select product_id, price ,databricks_cata.bronze.discount_func(price) as discounted_price
FROM products
```

▶ 📄 _sqldf: pyspark.sql.dataframe.DataFrame = [product_id: string, price: double ... 1 more field]

Table ▾ +

	A _C product_id	1.2 price	1.2 discounted_price
1	P0001	1868.54	1681.686
2	P0002	587.13	528.417
3	P0003	1599.24	1439.316
4	P0004	651.71	586.5390000000001
5	P0005	1861.78	1675.602
6	P0006	936.36	842.724
7	P0007	1954.02	1758.618
8	P0008	1251.26	1126.134
9	P0009	1247.15	1122.4350000000002

▶ ▾ ✓ Apr 27, 2025 (1s) 10

```
df = df.withColumn("discounted_price",expr("databricks_cata.bronze.discount_func(price)"))
df.display()
```

▶ 📄 df: pyspark.sql.dataframe.DataFrame = [product_id: string, product_name: string ... 4 more fields]

Table ▾ +

	A _C product_id	A _C product_name	A _C category	A _C brand	1.2 price	1.2 discounted_price
1	P0001	Clearly Its	Beauty	Nike	1868.54	1681.686
2	P0002	Production Clear	Beauty	Apple	587.13	528.417
3	P0003	Culture Coach	Home	Revlon	1599.24	1439.316
4	P0004	Movement Part	Sports	LG	651.71	586.5390000000001
5	P0005	Fact Name	Clothing	Samsung	1861.78	1675.602
6	P0006	Usually Stop	Toys	Adidas	936.36	842.724
7	P0007	Reveal Current	Sports	Adidas	1954.02	1758.618
8	P0008	Force Language	Beauty	Puma	1251.26	1126.134
9	P0009	Stage Leg	Clothing	Samsung	1247.15	1122.4350000000002

expr is used whenever we want to use sql functions

Python way

▶ ✓ Apr 27, 2025 (1s)

```
%sql
CREATE OR REPLACE FUNCTION databricks_cata.bronze.upper_func(p_brand STRING)
RETURNS STRING
LANGUAGE PYTHON
AS
$$
|   return p_brand.upper()
$$
```

OK

▶ ✓ Apr 27, 2025 (14s)

```
%sql
SELECT product_id, brand, databricks_cata.bronze.upper_func(brand) as brand_upper
FROM products
```

▶ _sqldf: pyspark.sql.dataframe.DataFrame = [product_id: string, brand: string ... 1 more field]

Table ▼ +

	product_id	brand	brand_upper
17	P0017	Revlon	REVLON
18	P0018	Puma	PUMA
19	P0019	Samsung	SAMSUNG
20	P0020	Nike	NIKE
21	P0021	Nike	NIKE
22	P0022	Puma	PUMA
23	P0023	Adidas	ADIDAS
24	P0024	Samsung	SAMSUNG
25	P0025	Nike	NIKE
--	-----	-	-----

▶ ✓ Apr 27, 2025 (12s)

```
df.write.format("delta")\
  .mode("overwrite")\
  .option("path", "abfss://silver@databricksete.dfs.core.windows.net/products")\
  .save()
```

▶ ✓ Apr 27, 2025 (2s)

```
%sql
CREATE TABLE IF NOT EXISTS databricks_cata.silver.products_silver
USING DELTA
LOCATION 'abfss://silver@databricksete.dfs.core.windows.net/products'
```

OK

▶ ✓ 2 days ago (8s)

```
%sql
CREATE OR REPLACE FUNCTION databricks_cata.bronze.tablefunc(p_year DOUBLE)
RETURNS TABLE(order_id STRING, year DOUBLE)
LANGUAGE SQL
RETURN
( SELECT order_id, year FROM databricks_cata.gold.factorders
  WHERE year = p_year )
```

▶

✓

2 days ago (9s)

```
%sql
SELECT * FROM databricks_cata.bronze.tablefunc(2024)
```

▶

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [order_id: string, year: double]

Table

▼

+

	order_id	year
28	O00045	2024
29	O00049	2024
30	O00050	2024
31	O00051	2024
32	O00052	2024
33	O00055	2024
34	O00059	2024
35	O00066	2024

Silver_Regions

Python

Tabs: OFF

☆

File

Edit

View

Run

Help

Workspace

← Databricks ETE Project

Bronze_Layer

Gold Orders

Gold Products

Gold_Customers

parameters

Silver_Customers

Silver_Orders

Silver_Products

Silver_Regions

▶

✓

Apr 27, 2025 (1s)

1

```
df = spark.read.table("databricks_cata.bronze.regions")
```

▶

df: pyspark.sql.dataframe.DataFrame = [region_id: string, region: string ... 1 more field]

▶

✓

Apr 27, 2025 (1s)

2

```
df.display()
```

Table

▼

+

	product_id	product_name	category	brand	price	discounted_price
1	P0001	Clearly Its	Beauty	Nike	1868.54	1681.686
2	P0002	Production Clear	Beauty	Apple	587.13	528.417
3	P0003	Culture Coach	Home	Revlon	1599.24	1439.316
4	P0004	Movement Part	Sports	LG	651.71	586.5390000000001
5	P0005	Fact Name	Clothing	Samsung	1861.78	1675.602
6	P0006	Usually Stop	Toys	Adidas	936.36	842.724
7	P0007	Reveal Current	Sports	Adidas	1954.02	1758.618
8	P0008	Force Language	Beauty	Puma	1251.26	1126.134
9	P0009	Stage Leg	Clothing	Samsung	1247.15	1122.4350000000002
10	P0010	Leader Then	Sports	Sony	975.53	877.977
11	P0011	Term Rest	Electronics	Adidas	1475.31	1327.779
12	P0012	Theory Wrong	Electronics	Apple	624.61	562.149
13	P0013	Democrat Book	Sports	Puma	384.99	346.49100000000004
14	P0014	Follow Brother	Clothing	Dell	32.2	28.980000000000004
15	P0015	Pattern Story	Beauty	LG	1459.2	1313.28

500 rows | 1.12s runtime


```
▶ ✓ Apr 27, 2025 (<1s)

df = df.drop("_rescued_data")

▶ df: pyspark.sql.dataframe.DataFrame = [region_id: string, region: string]
```

```
▶ ✓ Apr 27, 2025 (4s)

df.write.format("delta")\
  .mode("overwrite")\
  .save("abfss://silver@databricksete.dfs.core.windows.net/regions")
```

```
▶ ✓ Apr 27, 2025 (1s)

%sql
CREATE TABLE IF NOT EXISTS databricks_cata.silver.regions_silver
USING DELTA
LOCATION 'abfss://silver@databricksete.dfs.core.windows.net/regions'

OK
```

Gold Layer

Create a gold schema from UI (go to Catalogs→ Create schema)

We will prepare SCD Type 1 in such a way that it will take care of historical/initial/Full Load as well as Incremental Load

init_load_flag =1 for initial load, for rest loads keep init_load_flag =0 (init_load_flag == 0 means incremental load)

Gold_Customers Python Tabs: OFF ☆
File Edit View Run Help

init_load_flag ⚙️
0

1

▶ ✓ Apr 27, 2025 (<1s)

```
from pyspark.sql.functions import *
from pyspark.sql.types import *
```

2

▶

```
dbutils.widgets.text("init_load_flag", "0")
```

3

▶ ✓ Apr 27, 2025 (<1s)

```
init_load_flag = int(dbutils.widgets.get("init_load_flag"))
```

Data Reading From Source

```
df = spark.sql("select * from databricks_cata.silver.customers_silver")
```

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

Removing Duplicates

```
df = df.dropDuplicates(subset=['customer_id'])
```

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

	A ^B _C customer_id	A ^B _C email	A ^B _C city	A ^B _C state	A ^B _C domains	A ^B _C full_name
1	C01220	matthew01@yahoo.com	New Andrewhav...	WA	yahoo.com	Thomas Fitzgerald
2	C01579	nathancastro@gmail.com	South Amanda	FL	gmail.com	Brittany Schmidt
3	C01155	cynthia51@lewis-dixon.com	East Theresa	FL	lewis-dixon.com	Latasha Phelps
4	C01943	rodriguezzachary@hotmail.co...	East Timothy	PA	hotmail.com	David Cooper
5	C01554	sthompson@harding.com	New Leefurt	ME	harding.com	Gavin Lindsey
6	C00767	rsmith@pruitt-hodges.net	Heatherton	IL	pruitt-hodges.net	Pam Watts
7	C01097	joel22@stone-holmes.com	Davidhaven	LA	stone-holmes.com	James Martinez
8	C01674	crystalraymond@keller.com	Hoborough	ME	keller.com	Nicole Johnston
9	C00215	petersonthomas@yahoo.com	South Andrea	OK	yahoo.com	Danielle Huerta
10	C00587	phillipsstephanie@gmail.com	New Ronaldmouth	OR	gmail.com	Michael Bailey

We need a Surrogate key column (pseudo key to easily apply joins) → Dimension Surrogate key

Dividing New vs Old Records

```

10
if init_load_flag == 0:
    df_old = spark.sql('''select DimCustomerKey, customer_id, create_date, update_date
                        from databricks_cata.gold.DimCustomers''')
else:
    df_old = spark.sql('''select 0 DimCustomerKey, 0 customer_id, 0 create_date, 0 update_date
                        FROM databricks_cata.silver.customers_silver where 1=0''')
df_old: pyspark.sql.connect.dataframe.DataFrame = [DimCustomerKey: long, customer_id: string ... 2 more fields]

```

where 1=0 means it will only return the columns, not data; 0 is required since we are creating pseudo columns

Just now (<1s) 10

df_old.display()

Table +

1 ² DimCustomersKey	1 ² customer_id	1 ² create_date	1 ² update_date
No rows returned			

First time

Renaming Columns of df_old

Apr 27, 2025 (<1s) 13

```
df_old = df_old.withColumnRenamed("DimCustomerKey", "old_DimCustomerKey")\
                .withColumnRenamed("customer_id", "old_customer_id")\
                .withColumnRenamed("create_date", "old_create_date")\
                .withColumnRenamed("update_date", "old_update_date")
```

df_old: pyspark.sql.connect.dataframe.DataFrame = [old_DimCustomerKey: long, old_customer_id: string ... 2 more fields]

Applying Join with the Old Records

Apr 27, 2025 (1s) 15

```
df_join = df.join(df_old, df['customer_id'] == df_old['old_customer_id'], 'left')
```

df_join: pyspark.sql.connect.dataframe.DataFrame = [customer_id: string, email: string ... 8 more fields]

Table +

	_name	1 ² DimCustomerKey	1 ² DimCustomersKey	1 ² customer_id	1 ² create_date	1 ² update_date
1	fooney	1	null	null	null	null
2	Sellers	2	null	null	null	null
3	ayes	3	null	null	null	null
4	cott	4	null	null	null	null
5	asquez	5	null	null	null	null
6	ccarthy	6	null	null	null	null
7	a Doyle	7	null	null	null	null
8	mpos	8	null	null	null	null
9	reen	9	null	null	null	null

null means new records

This is before renaming

▶ (2) Spark Jobs

	name	¹ ₃ DimCustomerKey	¹ ₃ old_DimCustomerKey	¹ ₃ old_customer_id	¹ ₃ old_create_date	¹ ₃ old_update_date
1	Mooney	1	null	null	null	
2	Sellers	2	null	null	null	
3	Ayes	3	null	null	null	
4	Cott	4	null	null	null	
5	Esqueez	5	null	null	null	
6	McCarthy	6	null	null	null	
7	McDoyle	7	null	null	null	

Separating New vs Old Records

▶ ✓ Apr 27, 2025 (<1s) 18

```
df_new = df_join.filter(df_join['old_DimCustomerKey'].isNull())
```

df_new: pyspark.sql.connect.dataframe.DataFrame = [customer_id: string, email: string ... 8 more fields]

▶ ✓ Apr 27, 2025 (<1s) 19

```
df_old = df_join.filter(df_join['old_DimCustomerKey'].isNotNull())
```

df_old: pyspark.sql.connect.dataframe.DataFrame = [customer_id: string, email: string ... 8 more fields]

Preparing df_old

▶ ✓ Apr 27, 2025 (1s) 21

```
# Dropping all the columns which are not required

df_old = df_old.drop('old_customer_id', 'old_update_date')

# Renaming "old_DimCustomerKey" to "DimCustomerKey"

df_old = df_old.withColumnRenamed("old_DimCustomerKey", "DimCustomerKey")

# Renaming "old_create_date" column to "create_date"

df_old = df_old.withColumnRenamed("old_create_date", "create_date")
df_old = df_old.withColumn("create_date", to_timestamp(col("create_date")))

# Recreating "update_date" column with current timestamp

df_old = df_old.withColumn("update_date", current_timestamp())
```

df_old: pyspark.sql.connect.dataframe.DataFrame = [customer_id: string, email: string ... 7 more fields]

Preparing df_new

```
▶ ✓ Apr 27, 2025 (<1s) 24

# Dropping all the columns which are not required

df_new = df_new.drop('old_DimCustomerKey', 'old_customer_id', 'old_update_date', 'old_create_date')

# Recreating "update_date", "current_date" columns with current timestamp

df_new = df_new.withColumn("update_date", current_timestamp())
df_new = df_new.withColumn("create_date", current_timestamp())

▶ df_new: pyspark.sql.connect.dataframe.DataFrame = [customer_id: string, email: string ... 6 more fields]
```

Surrogate Key - From 1

```
▶ ✓ Apr 27, 2025 (<1s) 27

df_new = df_new.withColumn("DimCustomerKey",monotonically_increasing_id()+lit(1))

▶ df_new: pyspark.sql.connect.dataframe.DataFrame = [customer_id: string, email: string ... 7 more fields]
```

+lit(1) has been defined cause it starts from 0 by default

Adding Max Surrogate Key

```
▶ ✓ Apr 27, 2025 (1s) 30

if init_load_flag == 1:
    max_surrogate_key = 0
else:
    df_maxsur = spark.sql("select max(DimCustomerKey) as max_surrogate_key from databricks_cata.gold.DimCustomers")

    #Converting df_maxsur to max_surrogate_key
    max_surrogate_key = df_maxsur.collect()[0]['max_surrogate_key']

▶ df_maxsur: pyspark.sql.connect.dataframe.DataFrame = [max_surrogate_key: long]
```

collect will help convert the dataframe into a variable

```
▶ ✓ Apr 27, 2025 (<1s) 31

df_new = df_new.withColumn("DimCustomerKey",lit(max_surrogate_key)+col("DimCustomerKey"))

▶ df_new: pyspark.sql.connect.dataframe.DataFrame = [customer_id: string, email: string ... 7 more fields]
```

Union of df_old and df_new

```
▶ ✓ Apr 27, 2025 (1s) 33  
df_final = df_new.unionByName(df_old)  
▶ df_final: pyspark.sql.connect.dataframe.DataFrame = [customer_id: string, email: string ... 7 more fields]
```

unionByName will do a union by column name

SCD Type - 1

```
▶ ✓ Apr 27, 2025 (<1s)  
from delta.tables import DeltaTable
```

```
▶ ✓ Apr 27, 2025 (5s) 37  
if (spark.catalog.tableExists("databricks_cata.gold.DimCustomers")):  
    dlt_obj = DeltaTable.forPath(spark, "abfss://gold@databricksete.dfs.core.windows.net/DimCustomers")  
    dlt_obj.alias("trg").merge(df_final.alias("src"), "trg.DimCustomerKey = src.DimCustomerKey")\  
        .whenMatchedUpdateAll()\  
        .whenNotMatchedInsertAll()\  
        .execute()  
else:  
    df_final.write.mode("overwrite")\  
        .format("delta")\  
        .option("path", "abfss://gold@databricksete.dfs.core.windows.net/DimCustomers")\  
        .saveAsTable("databricks_cata.gold.DimCustomers")
```


Gold ProductsPython▼Tabs: OFF▼☆

FileEditViewRunHelp

Workspace

←Databricks ETE Project

Bronze_Layer

Gold Orders

Gold Products

Gold_Customers

parameters

Silver_Customers

Silver_Orders

Silver_Products

Silver_Regions

DLT Pipeline

▶✓Apr 27, 2025 (2s)2

import dlt
from pyspark.sql.functions import *

Streaming Table

▶4

Expectations
my_rules = {
 "rule1": "product_id IS NOT NULL",
 "rule2": "product_name IS NOT NULL"
}

▶5

@dlt.table()

@dlt.expect_all_or_drop(my_rules)
def DimProducts_stage():
 df = spark.readStream.option("skipChangeCommits", "true").table("databricks_cata.silver.products_silver")
 return df

Streaming View

```
▶ ✓ Apr 27, 2025 (2s)  
  
@dlt.view  
  
def DimProducts_view():  
    df = spark.readStream.table("DimProducts_stage")  
    return df  
  
at com.databricks.pipelines.util.SparkSessionUtils$.withSparkSession(SparkSes
```

DimProducts

▶ ✓ Apr 27, 2025 (<1s)

```
dlt.create_streaming_table("DimProducts")
```

DimProducts is defined as a **Delta Live Tables** dataset.

To populate your table you must either:

- ▶ Run an existing pipeline using the **Delta Live Tables** menu
- ▶ Create a new pipeline: [Create Pipeline](#)

▶ ✓ Apr 27, 2025 (1s)

```
dlt.apply_changes(  
    target = "DimProducts",  
    source = "DimProducts_view",  
    keys = ["product_id"],  
    sequence_by = "product_id",  
    stored_as_scd_type = 2  
)
```

```
at com.databricks.pipelines.util.SparkSessionUtils$.withSparkSession(SparkS
```

The screenshot displays the Databricks web interface. At the top, the 'Gold Products' pipeline is selected, with a 'Python' language dropdown and a 'Star' icon. The interface includes a 'Catalog' sidebar on the left, a search bar, and buttons for 'Validate', 'Start', and 'Share'. The main area shows the 'DLT graph' tab, which visualizes the data flow. The graph consists of three nodes: 'dimproducts_stage' (a Streaming table, completed in 10s, with 500 rows), 'dimproducts_view' (a View), and 'dimproducts' (a Streaming table, completed in 21s, with 500 rows). Arrows indicate the flow from the stage to the view, and then to the final table. The 'dimproducts' node shows a progress bar with 500 rows completed and 0 rows in progress.

Gold OrdersPython▼Tabs: OFF▼☆

FileEditViewRunHelp

Workspace

←Databricks ETE Project

Bronze_Layer

Gold Orders

Gold_Products

Gold_Customers

parameters

Silver_Customers

Silver_Orders

Silver_Products

Silver_Regions

FACT ORDERS

Data Reading

▶✓Apr 27, 2025 (1s)3

df = spark.sql("select * from databricks_cata.silver.orders_silver")
df.display()

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

Table▼+

	order_id	customer_id	product_id	order_date	quantity	total_amount	year
1	O00001	C00710	P0159	2023-03-22T00:00:00.000+00:...	3	2022.87	2023
2	O00002	C00954	P0036	2023-06-30T00:00:00.000+00:...	2	3560.74	2023
3	O00003	C01578	P0427	2023-11-06T00:00:00.000+00:...	3	5903.52	2023
4	O00004	C00962	P0332	2024-02-27T00:00:00.000+00:...	3	4107.99	2024
5	O00005	C00156	P0038	2024-10-13T00:00:00.000+00:...	5	5784.95	2024
6	O00006	C00521	P0174	2023-05-17T00:00:00.000+00:...	5	407.75	2023
7	O00007	C00982	P0352	2024-01-18T00:00:00.000+00:...	4	4907.64	2024
8	O00008	C00976	P0172	2023-01-10T00:00:00.000+00:...	4	7037.88	2023

▶✓Apr 27, 2025 (1s)4

df_dimcus = spark.sql("select DimCustomerKey, customer_id as dim_customer_id from databricks_cata.gold.dimcustomers")

df_dimpro = spark.sql("select product_id as DimProductKey, product_id as dim_product_id from databricks_cata.gold.dimproducts")

▶df_dimcus: pyspark.sql.connect.dataframe.DataFrame = [DimCustomerKey: long, dim_customer_id: string]

▶df_dimpro: pyspark.sql.connect.dataframe.DataFrame = [DimProductKey: string, dim_product_id: string]

Creating Star Schema

Fact Dataframe

◀▶+ Code + Text

▶✓Apr 27, 2025 (1s)6

df_fact = df.join(df_dimcus, df['customer_id'] == df_dimcus['dim_customer_id'],how='left').join(df_dimpro, df['product_id'] == df_dimpro['dim_product_id'],how='left')

df_fact_new = df_fact.drop('dim_customer_id','dim_product_id','customer_id','product_id')

▶df_fact: pyspark.sql.connect.dataframe.DataFrame = [order_id: string, customer_id: string ... 9 more fields]

▶df_fact_new: pyspark.sql.connect.dataframe.DataFrame = [order_id: string, order_date: timestamp ... 5 more fields]

▶✓Apr 27, 2025 (1s)7

df_fact_new.display()

Table▼+

	order_id	order_date	quantity	total_amount	year	DimCustomerKey	DimProductKey
1	O00001	2023-03-22T00:00:00.000+00:...	3	2022.87	2023	958	P0159
2	O00002	2023-06-30T00:00:00.000+00:...	2	3560.74	2023	1624	P0036
3	O00003	2023-11-06T00:00:00.000+00:...	3	5903.52	2023	1410	P0427
4	O00004	2024-02-27T00:00:00.000+00:...	3	4107.99	2024	1716	P0332
5	O00005	2024-10-13T00:00:00.000+00:...	5	5784.95	2024	1227	P0038
6	O00006	2023-05-17T00:00:00.000+00:...	5	407.75	2023	1761	P0174
7	O00007	2024-01-18T00:00:00.000+00:...	4	4907.64	2024	336	P0352

Upsert on Fact Table

```
from delta.tables import DeltaTable
```

```
if spark.catalog.tableExists("databricks_cata.gold.FactOrders"):

    dlt_obj = DeltaTable.forName(spark, "databricks_cata.gold.FactOrders")

    dlt_obj.alias("trg").merge(df_fact_new.alias("src"), "trg.order_id = src.order_id AND trg.DimCustomerKey = src.DimCustomerKey AND trg.DimProductKey = src.DimProductKey")\
        .whenMatchedUpdateAll()\
        .whenNotMatchedInsertAll()\
        .execute()

else:
    df_fact_new.write.format("delta")\
        .option("path", "abfss://gold@databricksete.dfs.core.windows.net/FactOrders")\
        .saveAsTable("databricks_cata.gold.FactOrders")
```

```
%sql
select * from databricks_cata.gold.FactOrders
```

```
_sqldf: pyspark.sql.connect.dataframe.DataFrame = [order_id: string, order_date: timestamp ... 5 more fields]
```

	order_id	order_date	quantity	total_amount	year	DimCustomerKey	DimProductKey
1	O00001	2023-03-22T00:00:00.000+00:00	3	2022.87	2023	958	P0159
2	O00002	2023-06-30T00:00:00.000+00:00	2	3560.74	2023	1624	P0036
3	O00003	2023-11-06T00:00:00.000+00:00	3	5903.52	2023	1410	P0427
4	O00004	2024-02-27T00:00:00.000+00:00	3	4107.99	2024	1716	P0332
5	O00005	2024-10-13T00:00:00.000+00:00	5	5784.95	2024	1227	P0038
6	O00006	2023-05-17T00:00:00.000+00:00	5	407.75	2023	1761	P0174
7	O00007	2024-01-18T00:00:00.000+00:00	4	4907.64	2024	336	P0352
8	O00008	2023-01-10T00:00:00.000+00:00	4	7037.88	2023	1731	P0172
9	O00009	2023-04-20T00:00:00.000+00:00	3	4076.97	2023	1914	P0238
10	O00010	2023-07-07T00:00:00.000+00:00	4	5695.64	2023	1726	P0258

adb-2972879960737553.13.azuredatabricks.net/jobs/79696709994162/tasks?o=2972879960737553

Microsoft Azure databricks

Search data, notebooks, recents, and more... CTRL

New

- Workspace
- Recents
- Catalog
- Workflows
- Compute
- Marketplace

SQL

- SQL Editor
- Queries
- Dashboards

Workflows > Jobs >

End-To-Pipeline ☆

Runs Tasks

Parameters

- ...m/Databricks ETE Project/parameters
- Ansh Lamba's Cluster

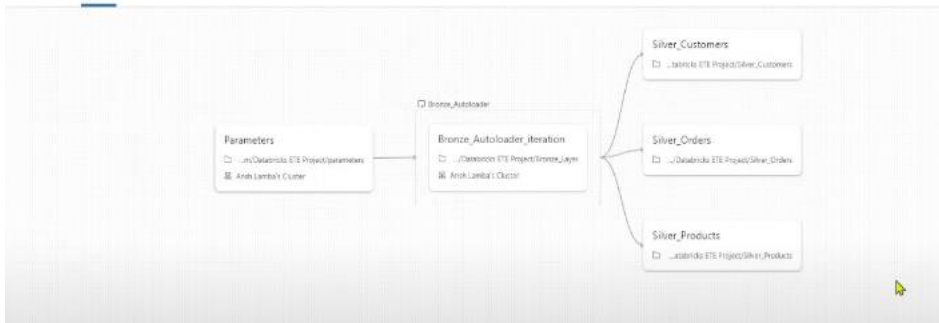
Bronze_Autoloader

Bronze_Autoloader_iteration

- .../Databricks ETE Project/Bronze_Layer
- Ansh Lamba's Cluster

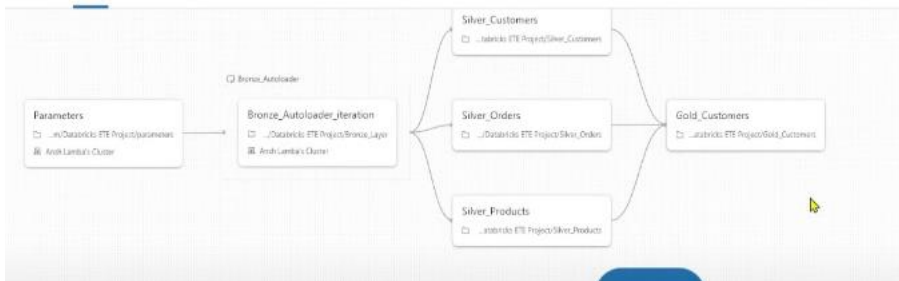
End-To-Pipeline ☆

Runs Tasks



End-To-Pipeline ☆

Runs Tasks



End-To-Pipeline ☆

Send feedback

Runs Tasks

Source* ⓪ Workspace

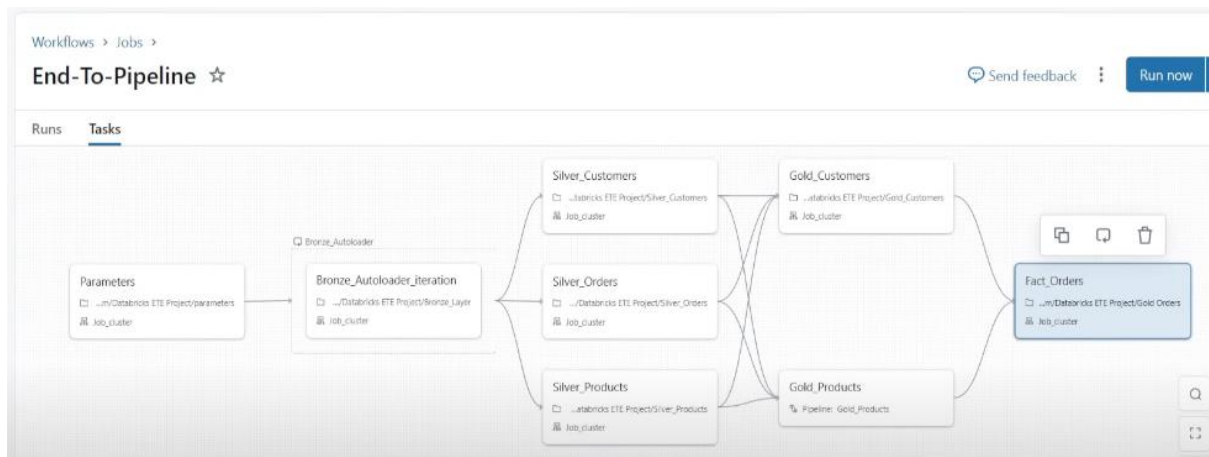
Path* ⓪ ...pace/Users/anshlambaa@gmail.com/Databricks ETE Project/Gold_Customers

Compute* ⓪ Serverless

Depends on Silver_Customers X Silver_Orders X Silver_Products X

Run if dependencies ⓪ Run if dependencies

Dependent libraries ⓪ Bronze_Autoloader



Gold will start once all the silver ingestion are completed, Fact gold will start once the Dimension gold are completed

Microsoft Azure databricks

Search data, notebooks, recent, and more... CTRL + P

Marketplace

SQL Editor

Queries

Dashboards

Genie

Alerts

Query History

SQL Warehouses

Data Engineering

Job Runs

Data Ingestion

Pipelines

Machine Learning

Playground

New SQL warehouse

Name: Ansh SQL Cluster

Cluster size: 2X-Small 4 DBU / h

Auto stop: After 10 minutes of inactivity.

Scaling: Min. 1 Max. 1 clusters (4 DBU)

Type: ☒ Serverless ☐ Pro ☐ Classic

Advanced options

Cancel Run

Microsoft Azure databricks

Search data, notebooks, recent, and more... CTRL + P

Catalog

Type to search...

For you All

- My organization
 - system
 - databricks.internal
 - databricks.cata
 - bronze
 - default
 - gold
 - information_schema
 - silver
 - main
 - Delta Shares Received
 - samples
 - Legacy

New Query 2025-04-27 12:28pm

Run (1000) databricks_ete.Select schema

1 | select * from databricks_cata.gold.factorders

Raw results

	order_id	order_date	quantity	total_amount	year	DimCustomerKey	DimProductKey
1	O00001	2023-03-22T00:00:00.0...	3	2022.87	2023	958	P0159
2	O00002	2023-06-30T00:00:00.0...	2	3560.74	2023	1624	P0036
3	O00003	2023-11-06T00:00:00.0...	3	5903.52	2023	1410	P0427
4	O00004	2024-02-27T00:00:00.0...	3	4107.99	2024	1716	P0332
5	O00005	2024-10-13T00:00:00.0...	5	5784.95	2024	1227	P0038