Data Engineering Project 4:

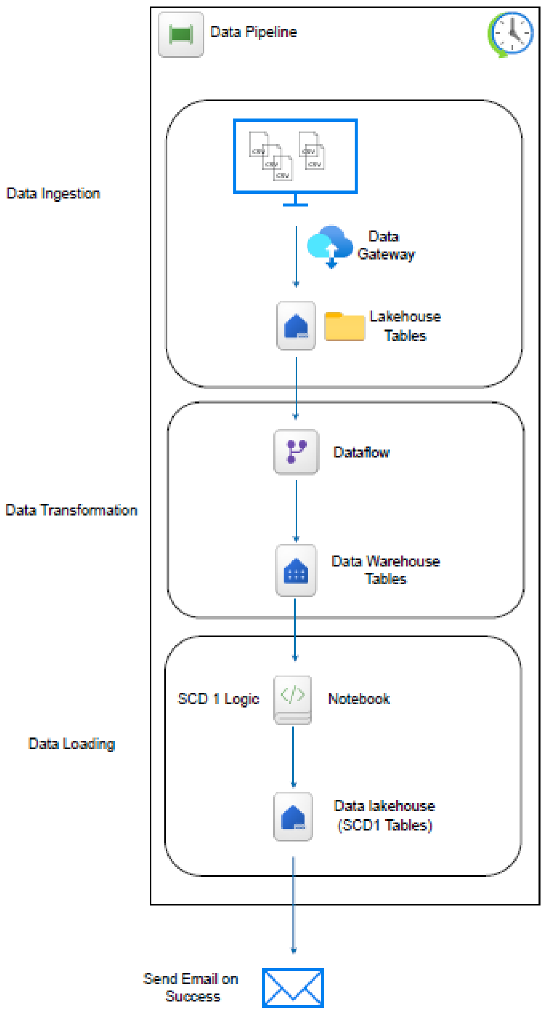
Incremental Data Loading and Automated Notifications using Microsoft Fabric

Table of Contents

Objective:

This project aims to design and implement an end-to-end data pipeline using Microsoft Fabric that incrementally ingests structured data from on-premises sources into a Lakehouse. It performs data transformation using Dataflows and Notebooks, applies SCD Type 1 logic, and automates notifications upon successful execution.

Task 1 - Architecture Diagram:

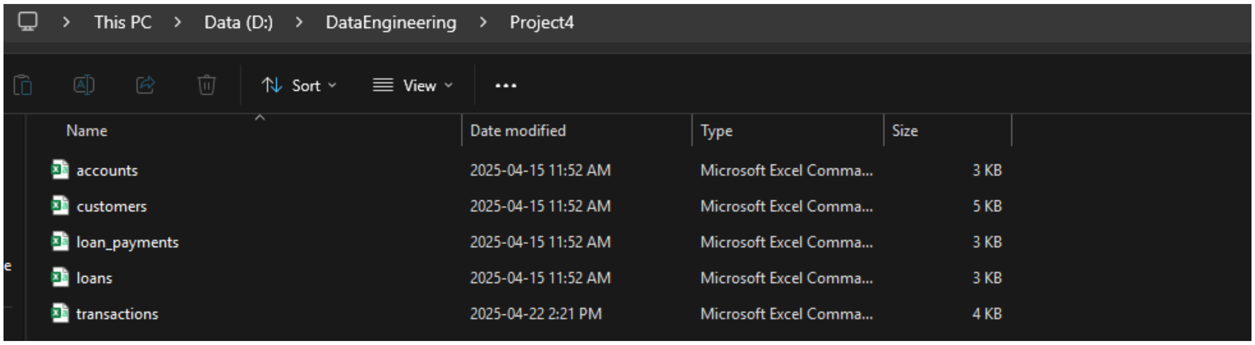


Task 1: Incremental Data Loading, SCD Type 1 and Notification

Step 1: On-Prem Setup

Dataset URL: [AI Bank Dataset]

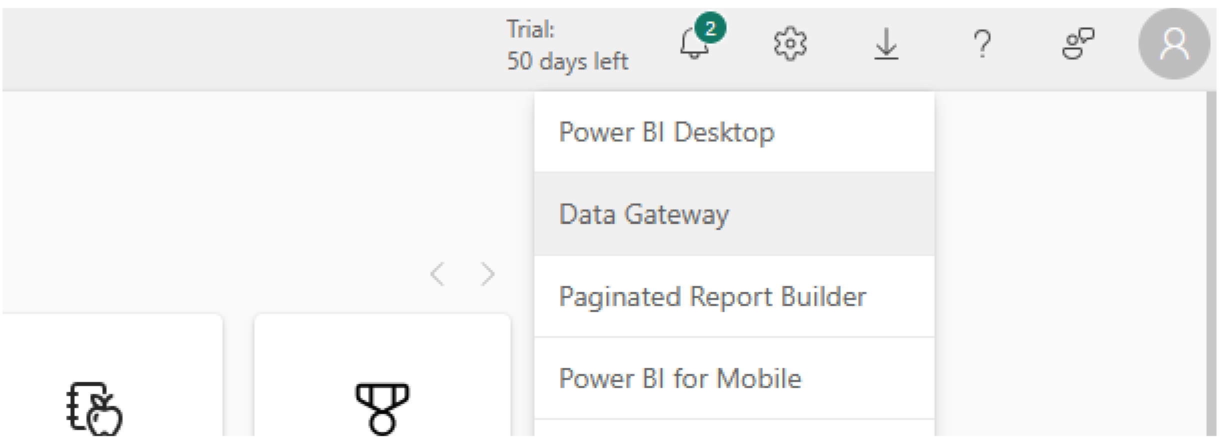
Step 1.1: Download the dataset files into the local folder on the system

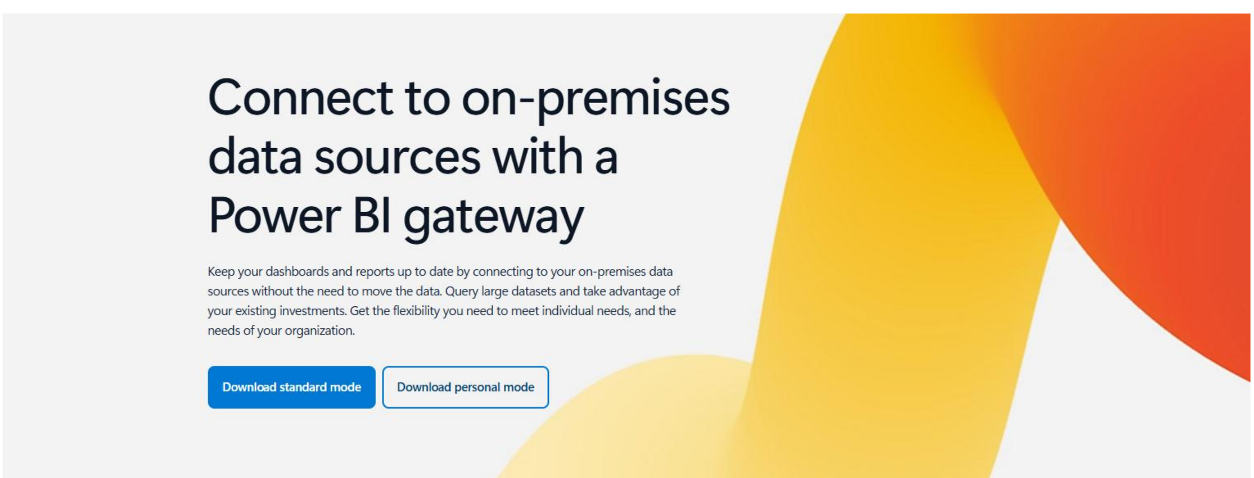


Step 1.2: Download Data Gateway from the Fabric workspace option as shown below

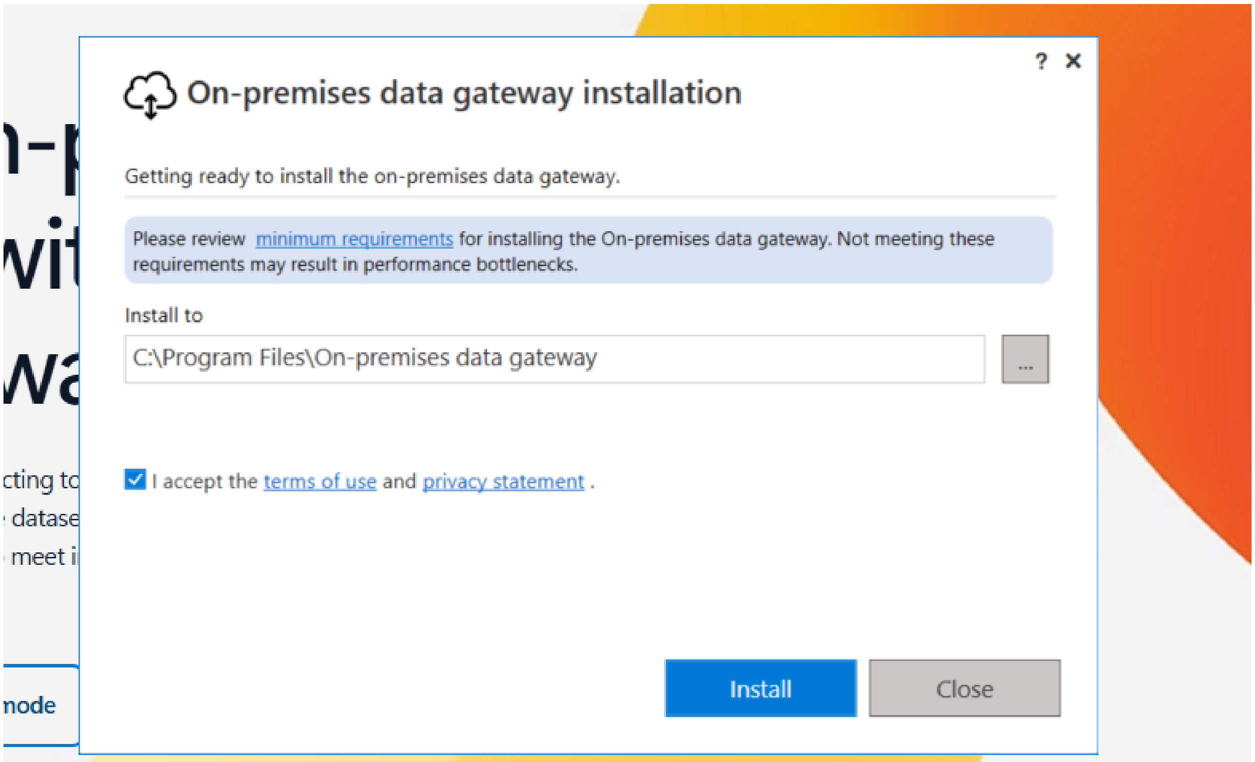
Settings icon -> Data Gateway

Download from this page: [Power BI Gateway | Microsoft Power BI]

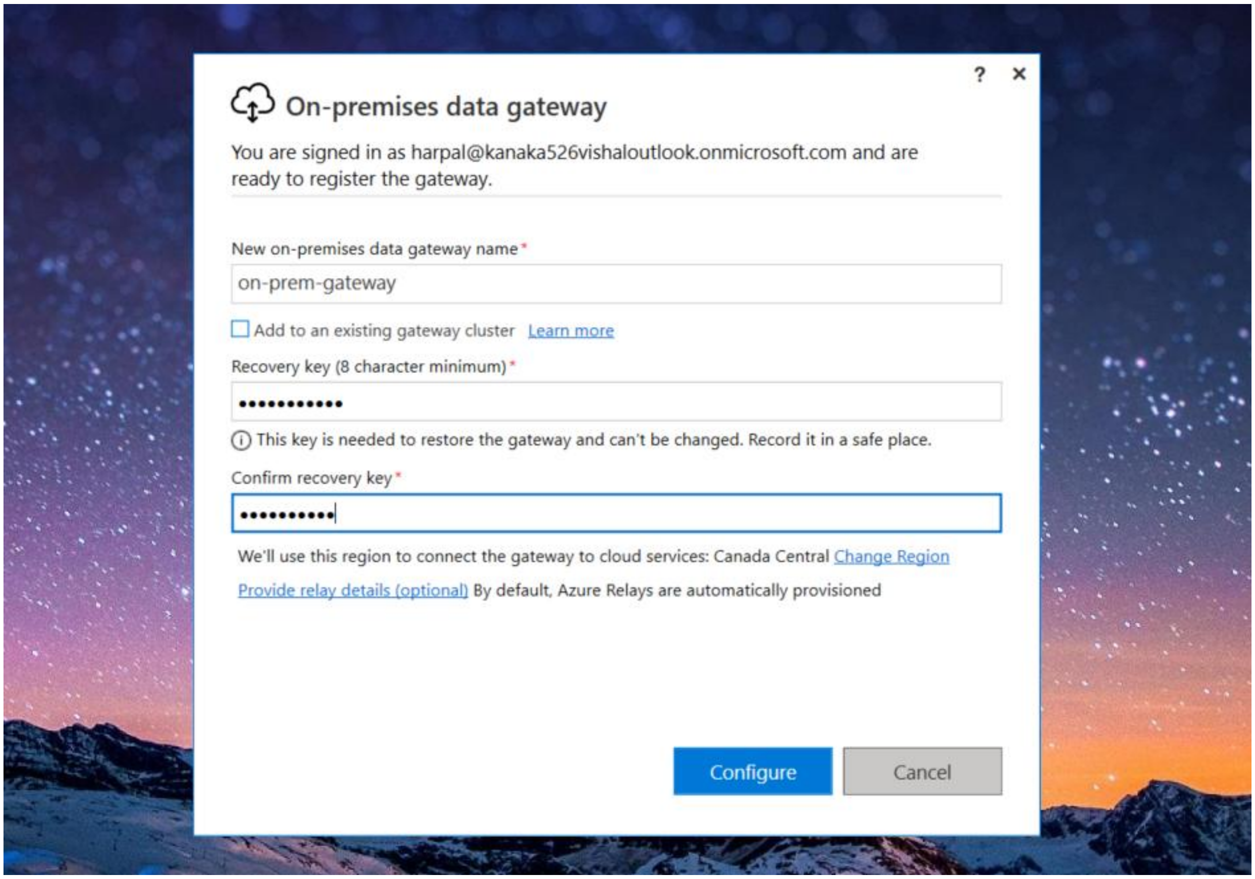


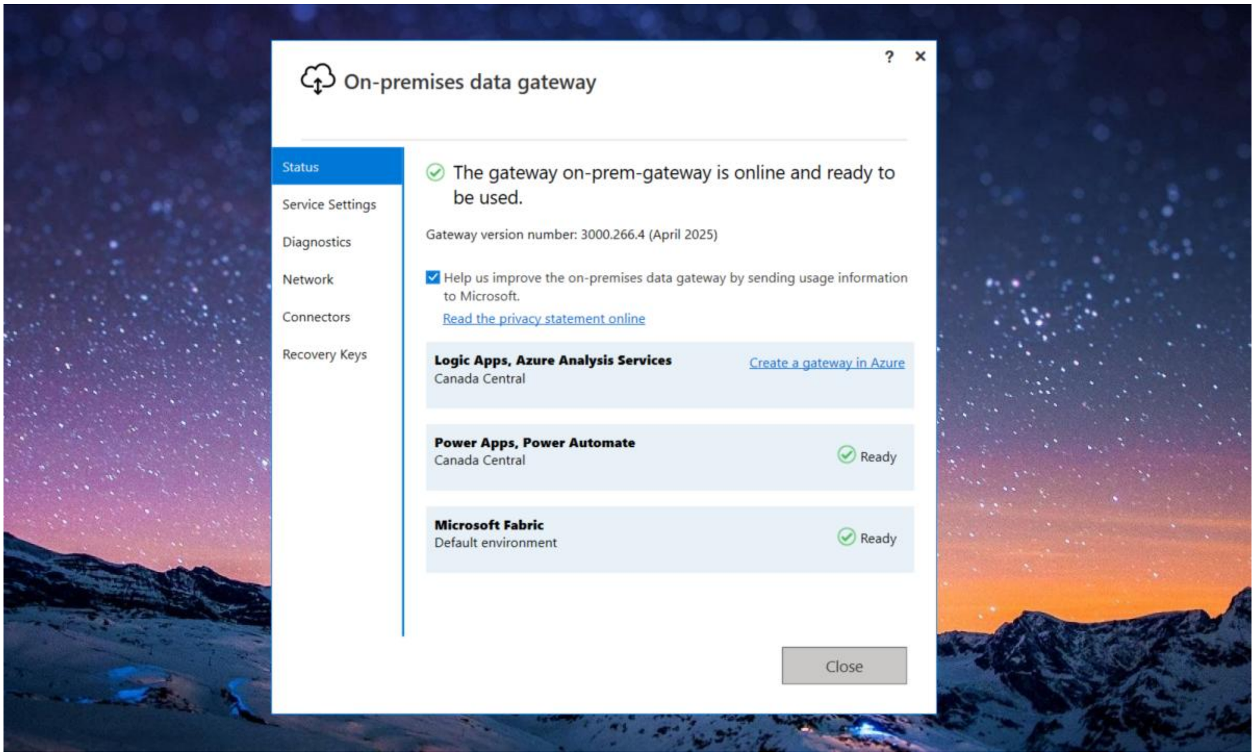


Step 1.3: Installation steps

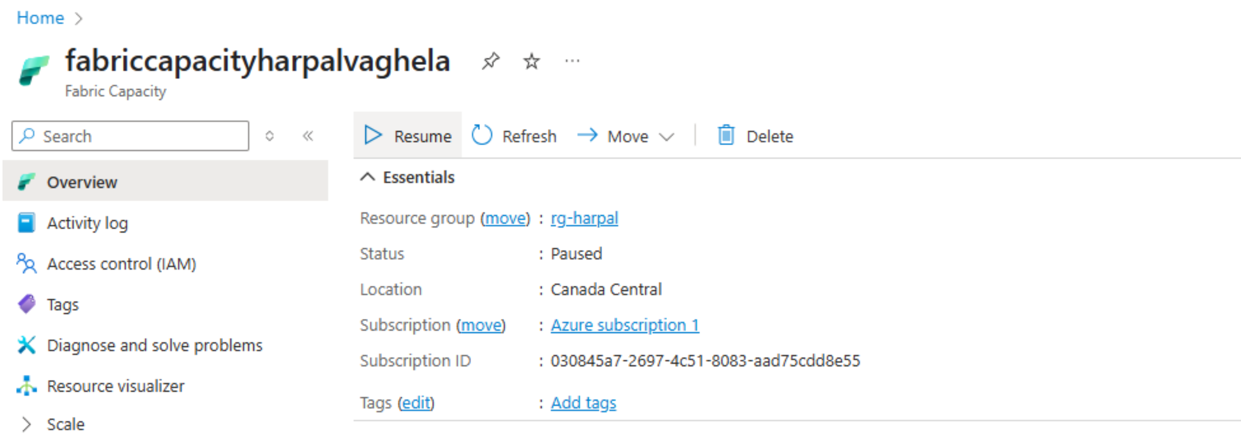


And configure it.

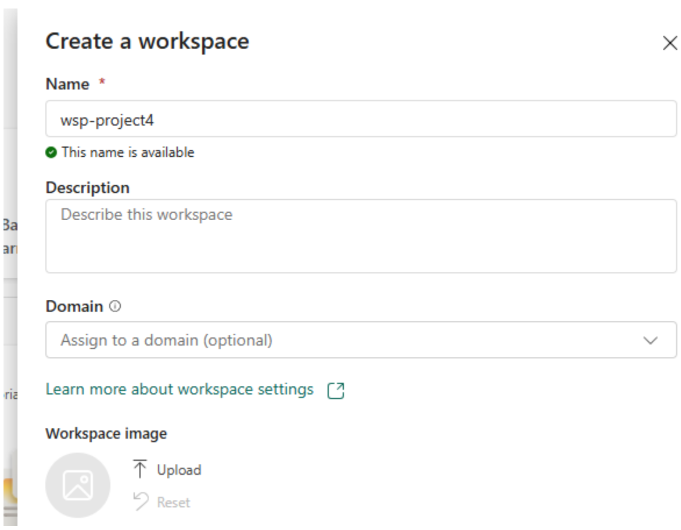

On-premises data gateway
You are signed in as harpal@kanaka526vishaloutlook.onmicrosoft.com and are ready to register the gateway.
New on-premises data gateway name *
on-prem-gateway
Add to an existing gateway cluster Learn more
Recovery key (8 character minimum) *
This key is needed to restore the gateway and can't be changed. Record it in a safe place.
Confirm recovery key *
We'll use this region to connect the gateway to cloud services: Canada Central Change Region
Provide relay details (optional) By default, Azure Relays are automatically provisioned
Configure
Cancel


On-premises data gateway
Status
Service Settings
Diagnostics
Network
Connectors
Recovery Keys
The gateway on-prem-gateway is online and ready to be used.
Gateway version number: 3000.266.4 (April 2025)
Help us improve the on-premises data gateway by sending usage information to Microsoft.
Read the privacy statement online
Logic Apps, Azure Analysis Services
Canada Central
Create a gateway in Azure
Power Apps, Power Automate
Canada Central
Ready
Microsoft Fabric
Default environment
Ready
Close

Step 1.4: Turn on Fabric Capacity from the Azure Account

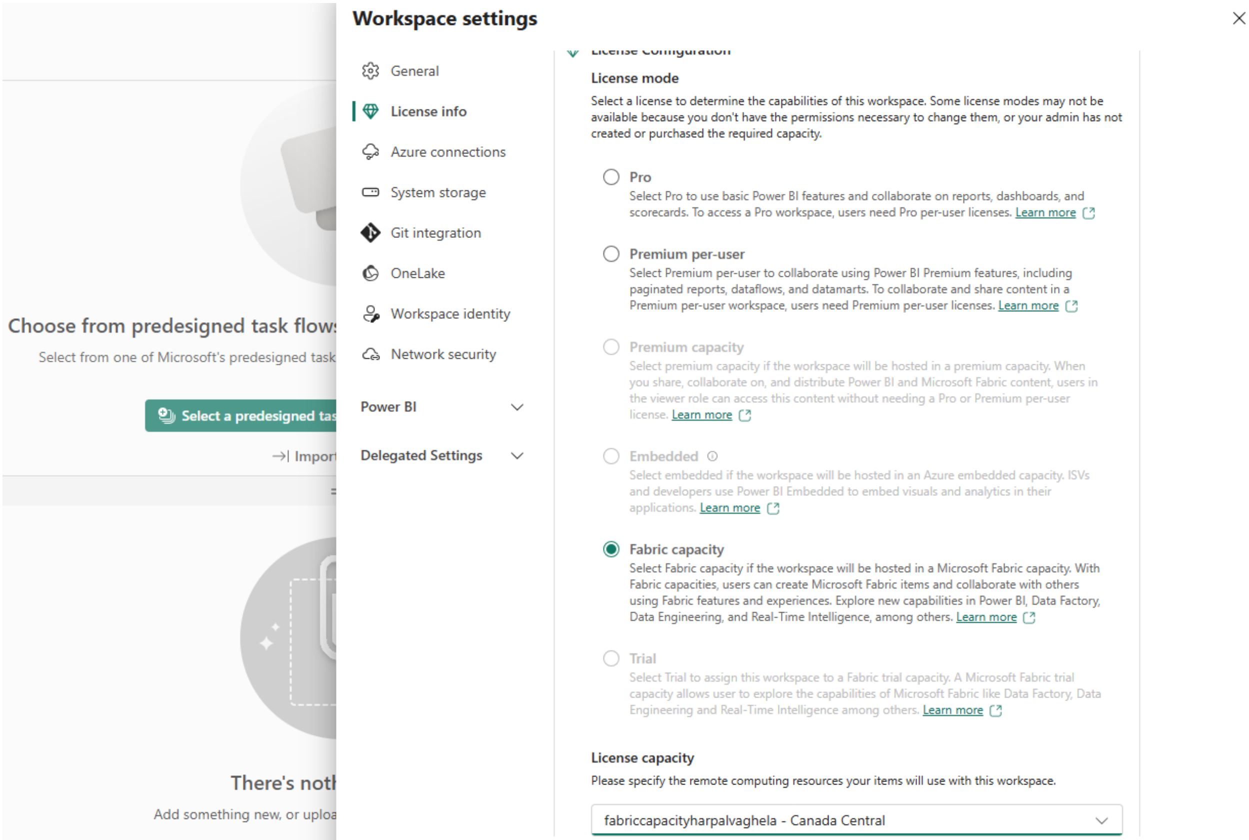


Step 1.5: Go to the Fabric Home Page, create a new workspace



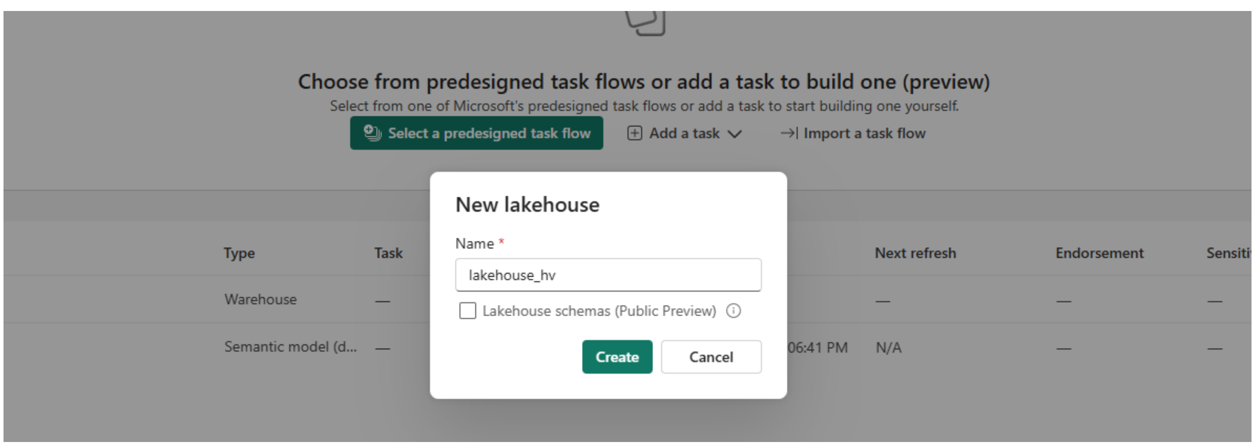
Go back to the Fabric Home page -> Workspace (that we just created) -> Workspace settings -> License Info -> License Configuration

Select License Mode: Fabric Capacity and License capacity as shown below:

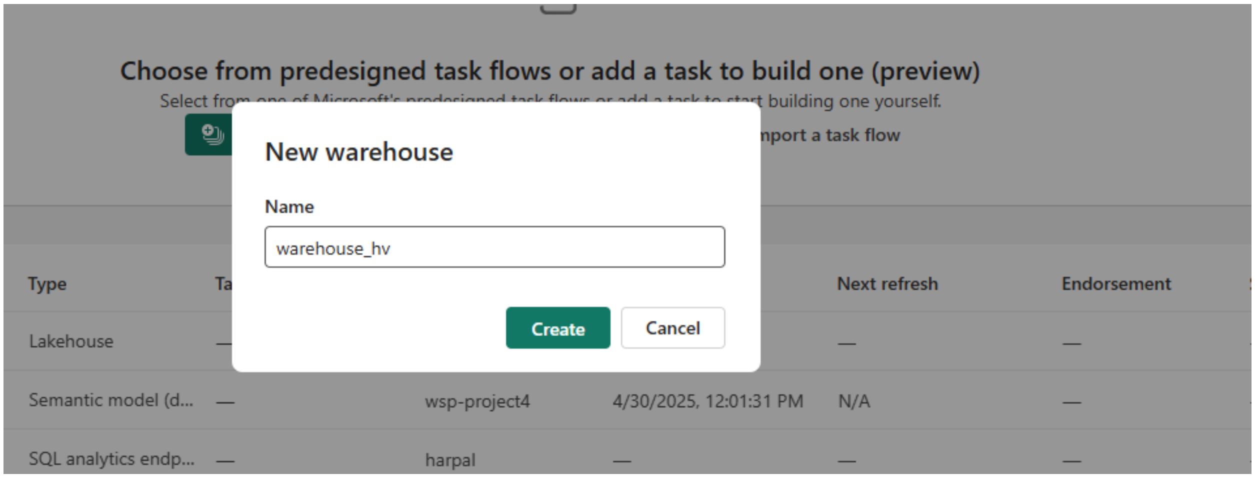


Step 2: Resources in the Fabric Workspace

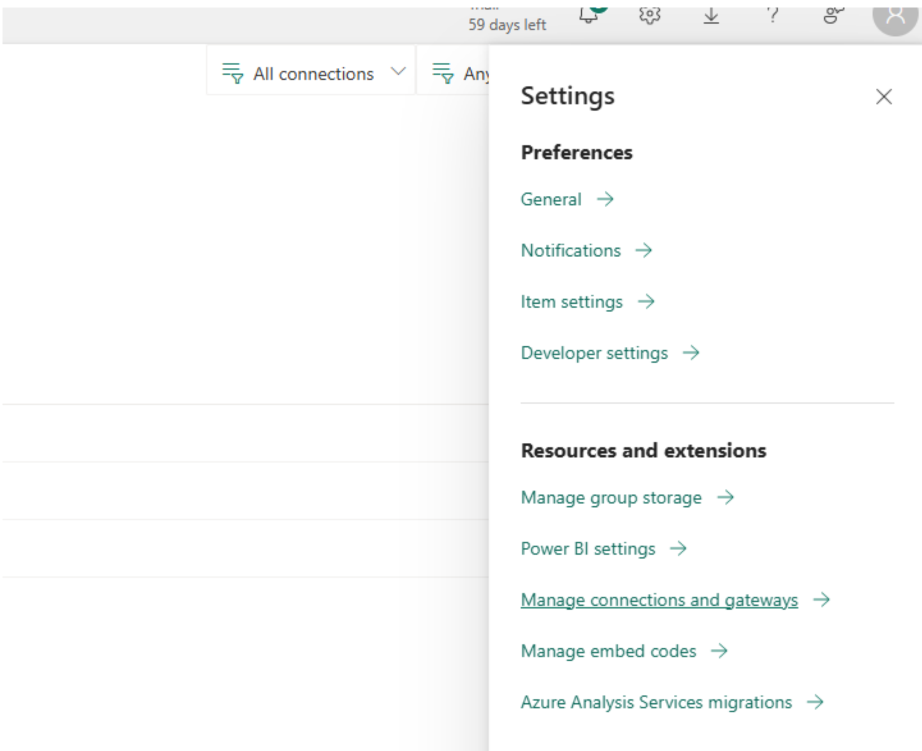
Step 2.1: Click on New Item -> Lakehouse



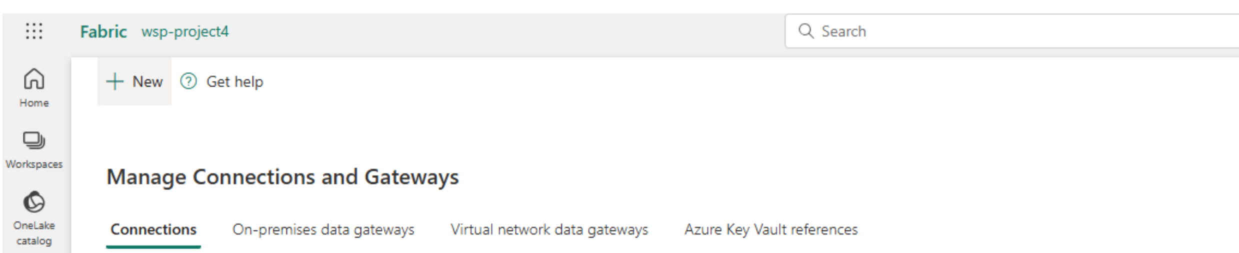
Step 2.2: Click on New Item -> Warehouse



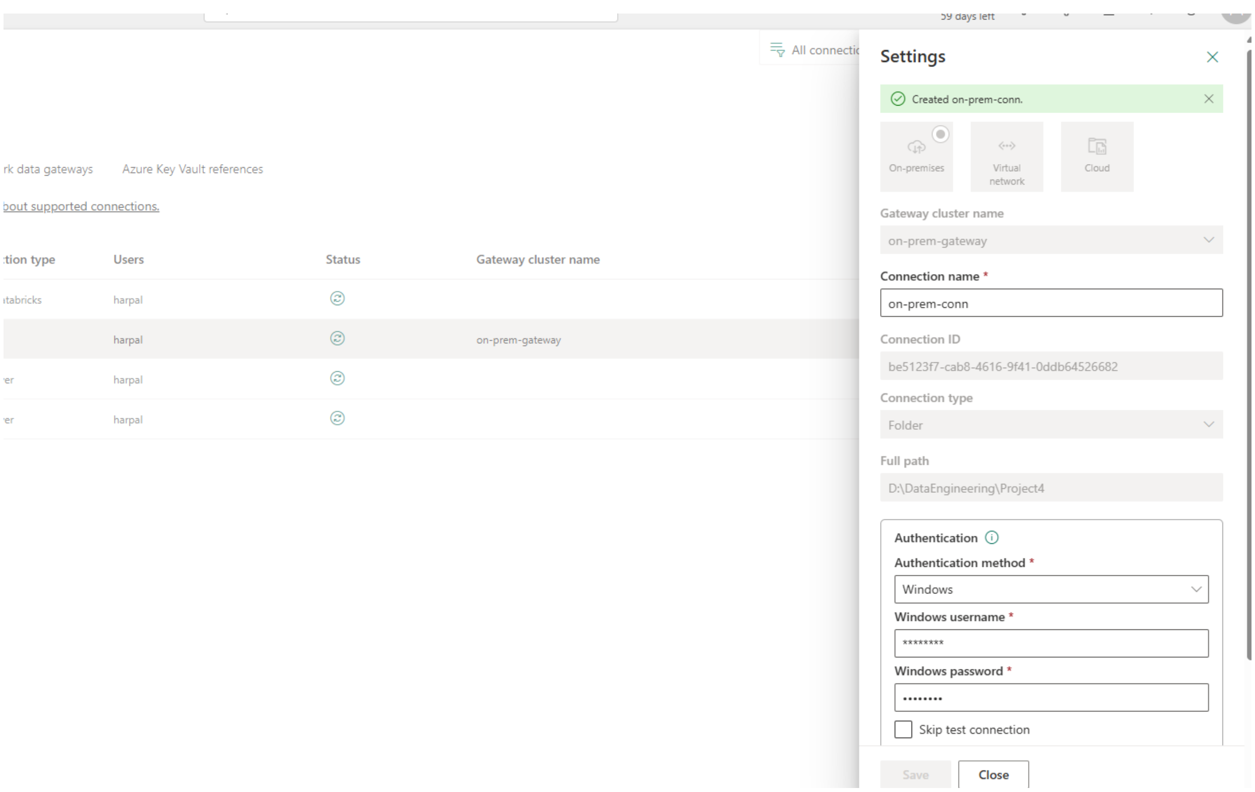
Step 2.3: Go to the Settings icon on the top right -> Manage connections and gateways



Click on **New**

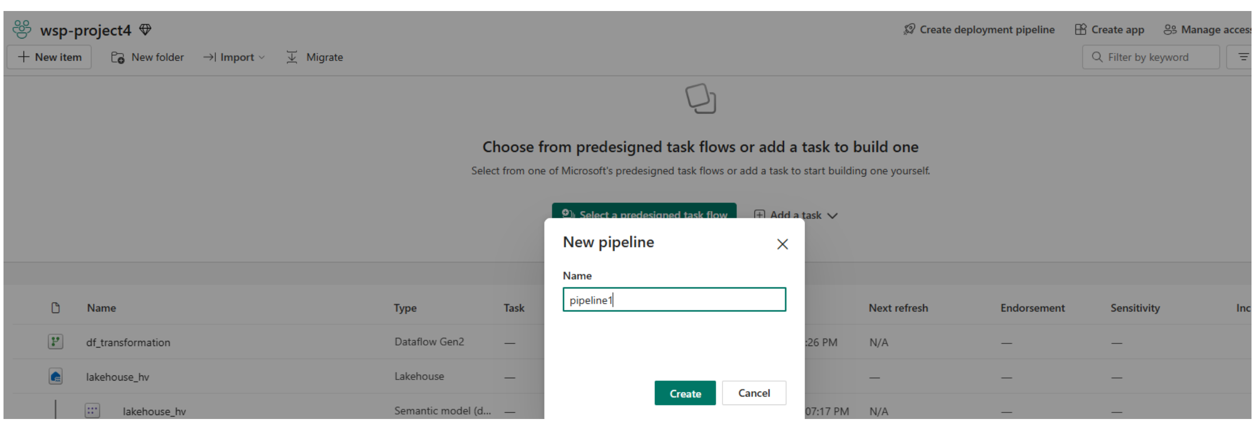


Enter these details as shown below



Step 2.4: Create a Data Pipeline

Workspace -> New Item -> Data Pipeline



Step 3: Data pipeline Design

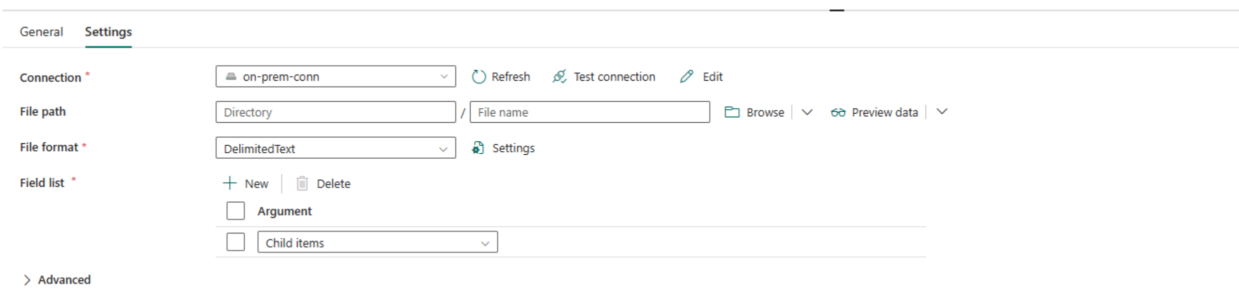
Step 3.1: Take the Get Metadata Activity

Settings tab:

Connection: On-Prem

File format: Delimited Text

Field List: Child Items

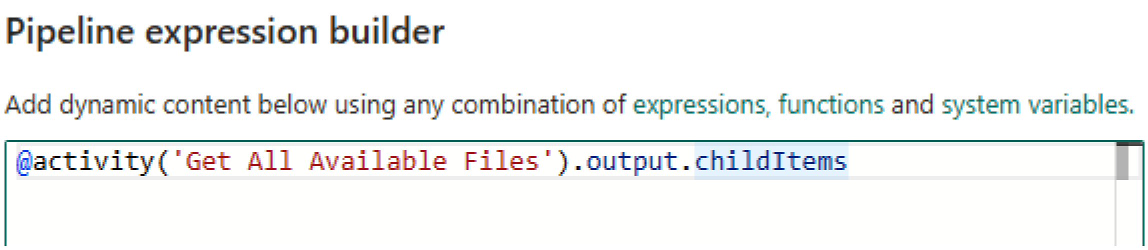


Step 3.2: Take ForEach Activity

Settings tab:

Batch count: 1

Items: @activity('Get All Available Files').output.childItems



Click on the Edit icon on the ForEach Activity

Step 3.3: Take a **Copy Data Activity**

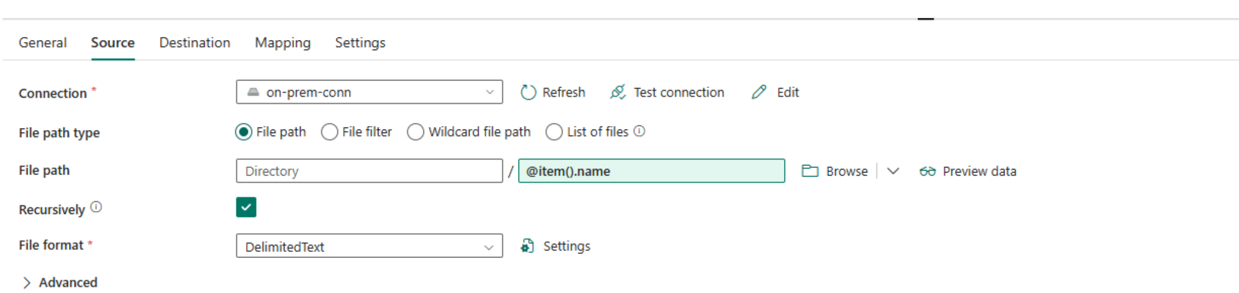
<u>Source Tab:</u>

Connection: On-Prem

File path type: File Path

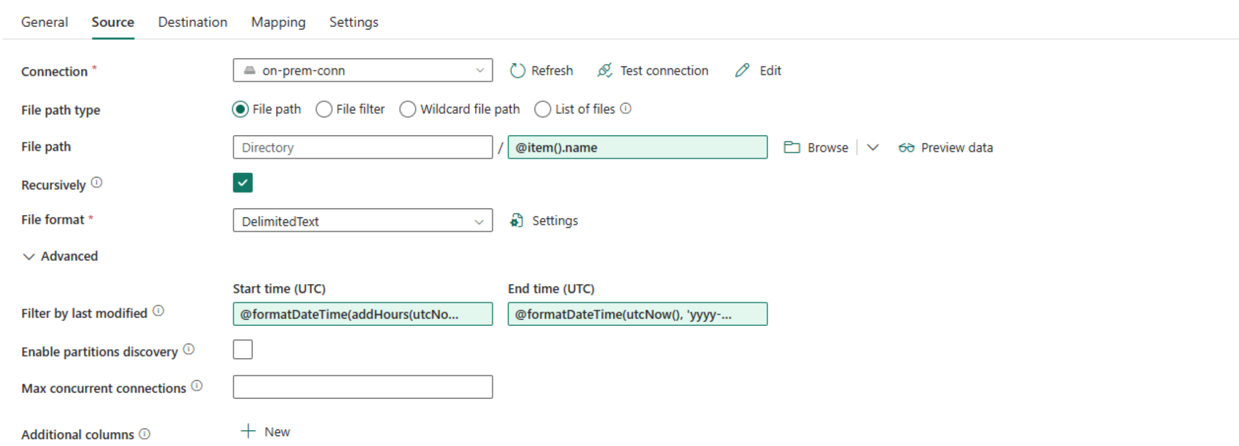
File path: In the field of file name write this **@item().name**

File format: Delimited Text



Start Time Expression: @formatDateTime(addHours(utcNow(), -24), 'yyyy-MM-ddTHH:mm:ssZ')

End Time Expression: @formatDateTime(utcNow(), 'yyyy-MM-ddTHH:mm:ssZ')



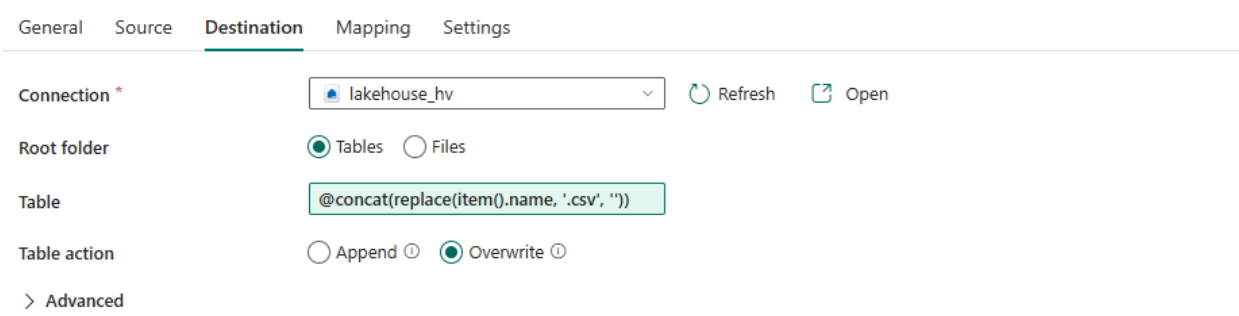
<u>Sink Tab:</u>

Connection: Lakehouse

Root Folder: Tables

Table: **@concat(replace(item().name, '.csv', ''))**

Table action: Overwrite

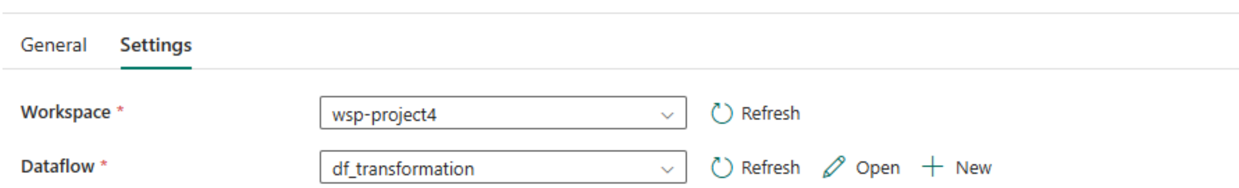


Step 3.4: Run the pipeline, so we will get the on-premise file converted into tables in Lakehouse

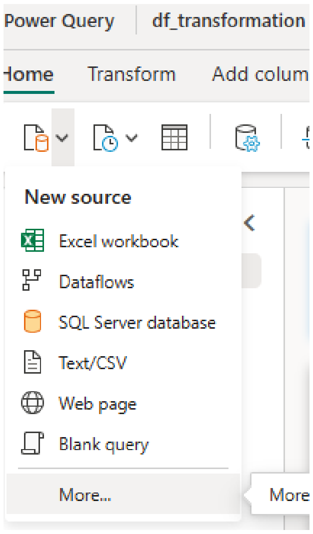
Step 3.5: Take a **Dataflow activity**:

Settings tab:

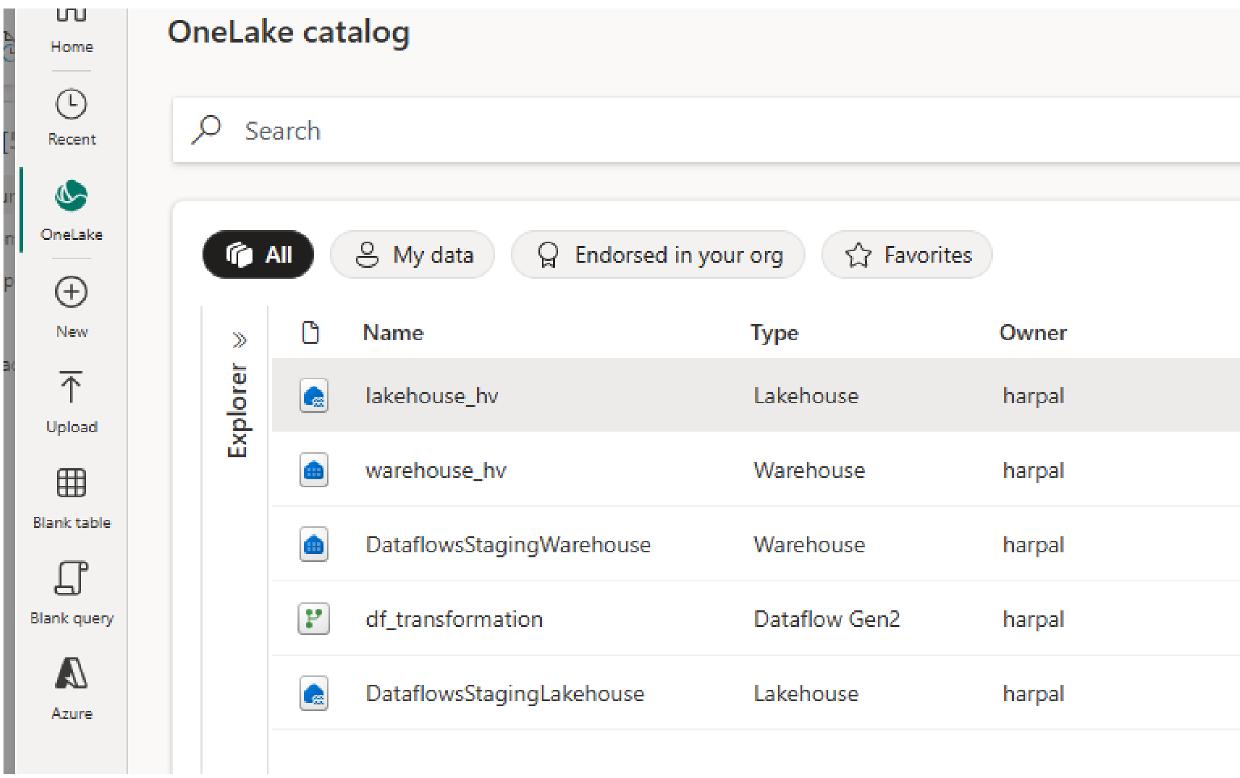
Dataflow: Click on New



Step 3.6: Click on Home -> More...



Step 3.7: Select Lakehouse



Step 3.8: Select the table from Lakehouse, click on create

Similarly, select all 5 tables from the Lakehouse

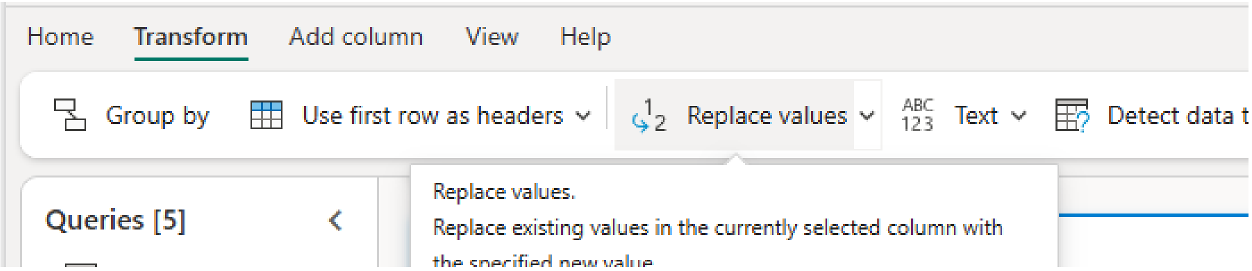
Now, we will do the transformation on each table

Select table -> Select all the columns from left to right using the Shift key on the keyboard

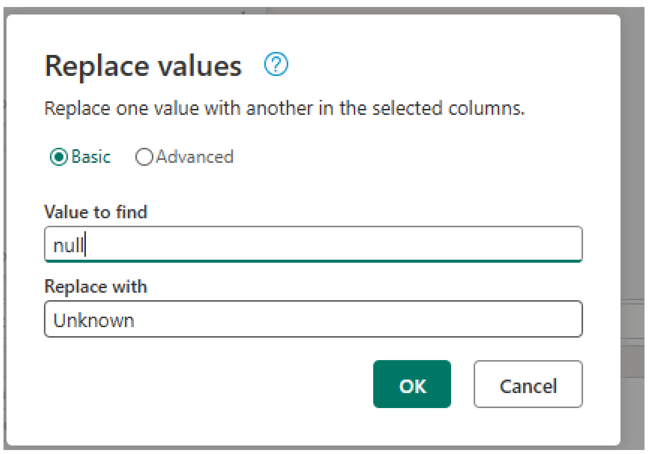
Go to the Transform tab and click on **Detect data type**

Select the column in which we want to replace the values

On the Transform tab -> Click on **Replace Values**

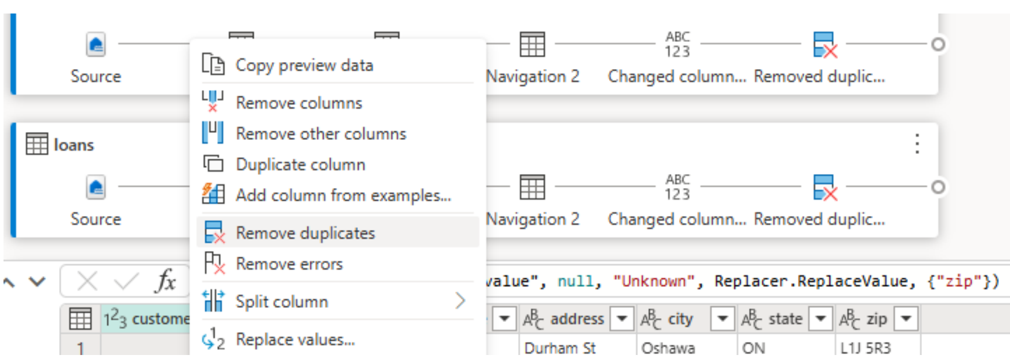


Replace null with “Unknown”

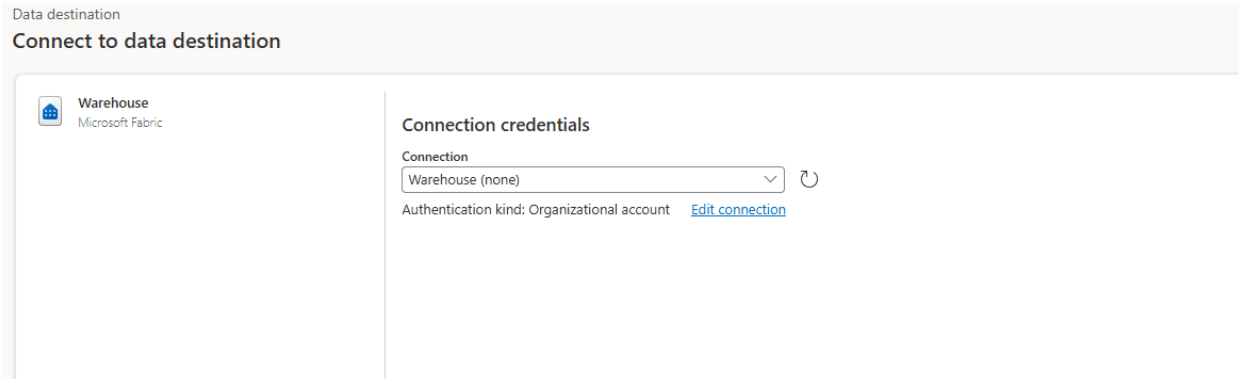


Do this step for all columns where null is present.

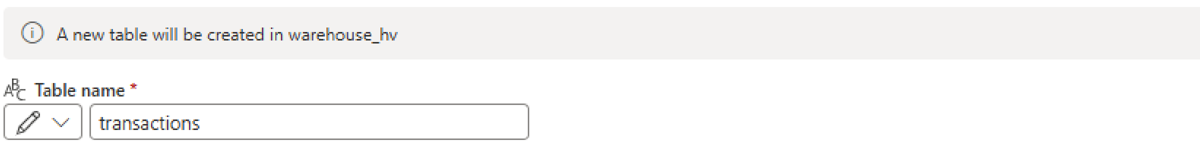
To remove duplicates, right-click on the column name and select **Remove duplicates**



Step 3.9: In each table, add sink as Warehouse

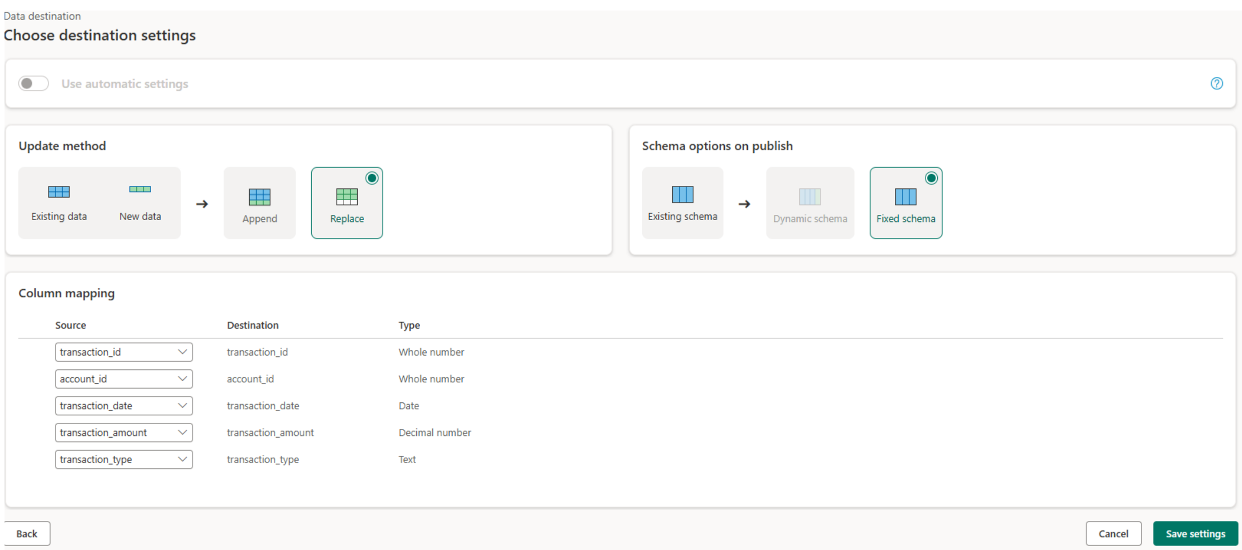


Write the table name if we want to create a new one at runtime



Click on Next.

Destination settings dialog:



Click on Save settings.

At the end of the Dataflow design screen, click on **publish**

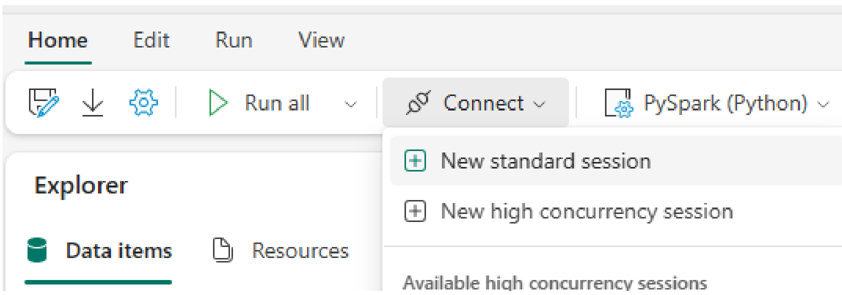
Step 4: Notebook Activity

After that, in the pipeline canvas area, take the **Notebook activity**

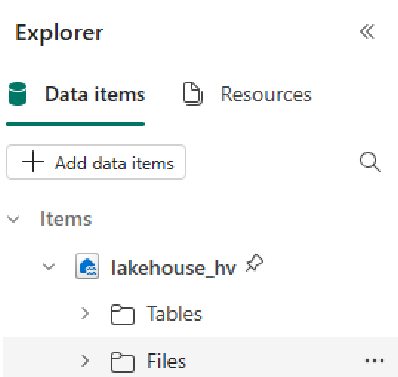
Step 4.1: Create a new Notebook

Here, we will do the SCD 1 transformation steps for all 5 tables

Step 4.2: Connect the Spark session:



Step 4.3 : On the left side, connect the existing **Lakehouse**



Step 4.4: SCD 1 Logic code

Let's code for SCD 1 now,

Write all the important libraries' code that we will need in this notebook

```
#Imports Libraries

from delta.tables import DeltaTable
from pyspark.sql.functions import col, crc32, concat_ws, current_timestamp, lit
import com.microsoft.spark.fabric
```

39] <1 sec - Command executed in 342 ms by harpal on 12:56:29 PM, 5/04/25

SCD 1 Target Tables:

```sql
%%sql

Create table if not exists accounts_scd1
(
  account_id int,
  customer_id int,
  account_type string,
  balance double,
  hashkey bigint,
  createdby string,
  createddate timestamp,
  updatedby string,
  updateddate timestamp
);


Create table if not exists customers_scd1
(
  customer_id integer,
  first_name string,
  last_name string,
  address string,
  city string,
  state string,
  zip string,
  hashkey long,
  createdby string,
  createddate timestamp,
  updatedby string,
  updateddate timestamp
);
```

```sql

Create table if not exists loan_payments_scd1
(
  payment_id int,
  loan_id int,
  payment_date date,
  payment_amount double,
  hashkey long,
  createdby string,
  createddate timestamp,
  updatedby string,
  updateddate timestamp
);

Create table if not exists loans_scd1
(
  loan_id int,
  customer_id int,
  loan_amount double,
  interest_rate double,
  loan_term int,
  hashkey long,
  createdby string,
  createddate timestamp,
  updatedby string,
  updateddate timestamp
);

Create table if not exists transactions_scd1
(
  transaction_id int,
  account_id int,
  transaction_date date,
  transaction_amount double,
  transaction_type string,
  hashkey long,
  createdby string,
  createddate timestamp,
  updatedby string,
  updateddate timestamp
);
```

Created a function called scd_type1_logic

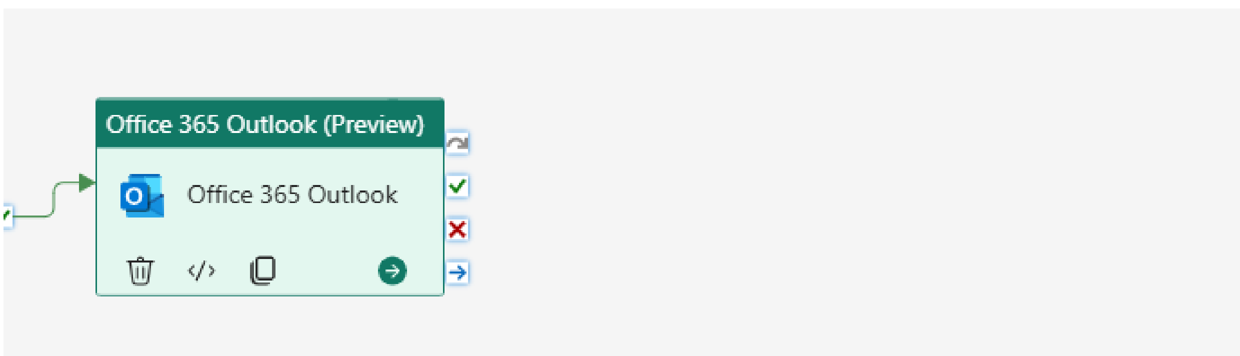
```python
def scd_type1_logic(
    warehouse_table: str,
    lakehouse_table_name: str,
    join_key: str,
    update_columns: list,
    insert_columns: list,
    created_by: str = "fabric",
    updated_by: str = "fabric-updated"
):
    try:
        # Add hash to warehouse table
        df_warehouse = spark.read.synapsesql(warehouse_table)

        df_hashed = (
            df_warehouse
            .withColumn("HashKey", crc32(concat_ws("|", *df_warehouse.columns)))
            .withColumn("CreatedDate", current_timestamp())
            .withColumn("UpdatedDate", current_timestamp())
            .withColumn("CreatedBy", lit(created_by))
            .withColumn("UpdatedBy", lit(created_by))
        )

        # Load existing SCD1 table
        target_table = DeltaTable.forName(spark, lakehouse_table_name)

        # Anti-join to find new/changed rows
        df_src = (
        df_hashed.alias("src")
        .join(
            target_table.toDF().alias("tgt"),
            col(f"src.{join_key}") == col(f"tgt.{join_key}"),
            "left"
        )
        .filter(
            (col(f"tgt.{join_key}").isNull()) |  # New records
            (col("src.HashKey") != col("tgt.HashKey"))  # Changed records
        )
        .select("src.*")
        )
```

```python
        if df_src.count() > 0:
            # Update dictionary
            update_set = {colname: f"source.{colname}" for colname in update_columns}
            update_set["UpdatedDate"] = current_timestamp()
            update_set["UpdatedBy"] = lit(updated_by)
            update_set["HashKey"] = "source.HashKey"

            # Insert dictionary
            insert_values = {colname: f"source.{colname}" for colname in insert_columns}
            insert_values.update({
                "CreatedDate": "source.CreatedDate",
                "UpdatedDate": current_timestamp(),
                "CreatedBy": "source.CreatedBy",
                "UpdatedBy": lit(created_by),
                "HashKey": "source.HashKey"
            })

            # Perform merge
            target_table.alias("target").merge(
                df_src.alias("source"),
                f"target.{join_key} = source.{join_key}"
            ).whenMatchedUpdate(
                set = update_set
            ).whenNotMatchedInsert(
                values = insert_values
            ).execute()

            print(f" --> Merge completed for: {lakehouse_table_name}")
        else:
            print(f"No changes found for: {lakehouse_table_name}")

    except Exception as e:
        print(f"Error in SCD Type 1 process: {str(e)}")

```

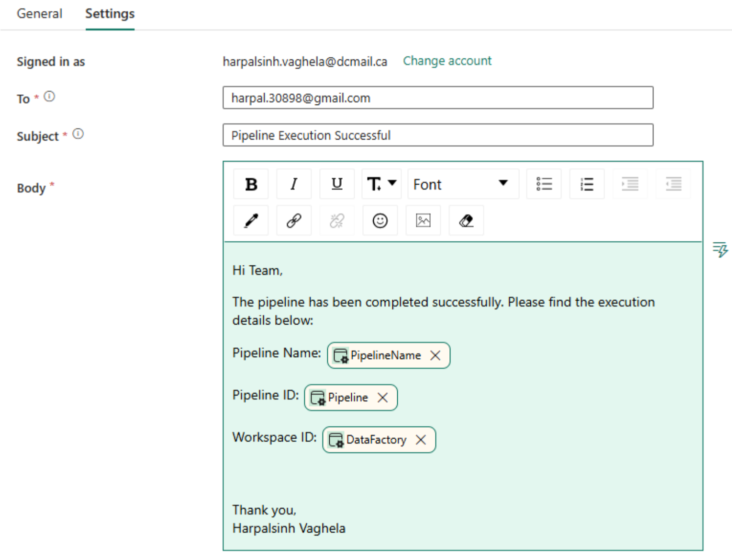
Calling that function:

```python
warehouse_schema = "warehouse_hv.dbo."
#lakehouse_base_path = "lakehouse_hv/Tables/"

table_names = ["accounts", "customers", "loans", "loan_payments", "transactions"]

for name in table_names:

    # Dynamically define join key and columns per file
    if name == "accounts":
        join_key = "account_id"
        update_cols = insert_cols = ["account_id", "customer_id", "account_type", "balance"]

    elif name == "customers":
        join_key = "customer_id"
        update_cols = insert_cols = ["customer_id", "first_name","last_name",  "address", "city", "zip"]

    elif name == "loans":
        join_key = "loan_id"
        update_cols = insert_cols = ["loan_id", "customer_id", "loan_amount", "interest_rate", "loan_term"]

    elif name == "loan_payments":
        join_key = "payment_id"
        update_cols = insert_cols = ["payment_id", "loan_id", "payment_date", "payment_amount"]

    elif name == "transactions":
        join_key = "transaction_id"
        update_cols = insert_cols = ["transaction_id", "account_id", "transaction_date", "transaction_amount", "transaction_type"]

    else:
        print(f"Unknown table: {name}")
        continue

    warehouse_table = f"{warehouse_schema}{name}"
    lakehouse_table_name = f"{name}_scd1"


    print(f"Processing: {name}")
    scd_type1_logic(
        warehouse_table=warehouse_table,
        lakehouse_table_name=lakehouse_table_name,
        join_key=join_key,
        update_columns=update_cols,
        insert_columns=insert_cols
    )
```

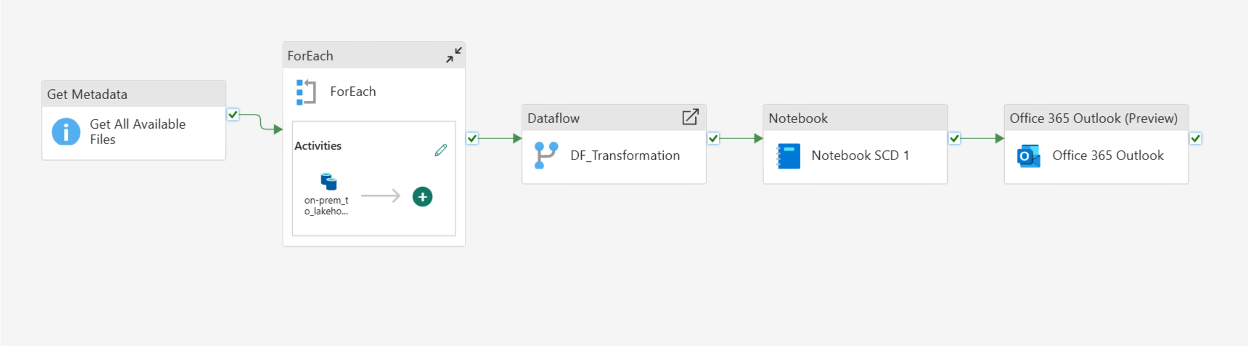
Go to Pipeline Design and take **Outlook email Activity**



Configure it as shown below:



Entire Pipeline Design:



Run the pipeline:

Checking the data in the target tables

Accounts

```python
df = spark.sql("SELECT * FROM lakehouse_hv.accounts_scd1 LIMIT 100")
print("--------------------------------accounts scd1 Data--------------------------------")
display(df)
```

✓ 1 sec - Command executed in 1 sec 619 ms by harpal on 1:02:49 PM, 5/04/25

Spark jobs (2 of 2 succeeded) Resources

--------------------------------accounts scd1 Data--------------------------------

Table + New chart

Table view

	account_id	customer_id	account_type	balance	hashkey	createdby	createddate	updatedby	updateddate
1	1	45	Savings	1000.5	3907550370	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
2	2	12	Checking	2500.75	3017445821	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
3	3	78	Savings	1500.0	348866053	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
4	4	34	Checking	3000.25	1015890270	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
5	5	56	Savings	500.0	2226578082	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
6	6	23	Checking	1200.5	2125750389	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
7	7	89	Savings	800.75	4287677219	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
8	8	67	Checking	2200.0	4121986909	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
9	9	14	Savings	900.25	51075731	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
10	10	92	Checking	1800.5	2452363199	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
11	11	3	Savings	1100.75	511600700	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
12	12	81	Checking	2700.0	2105206115	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
13	13	29	Savings	1300.25	407888565	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
14	14	64	Checking	3200.5	1787786631	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
15	15	47	Savings	700.75	4055528753	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
16	16	18	Checking	1400.0	2174730368	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
17	17	99	Savings	600.25	2802941046	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...

Customers

```python
df = spark.sql("SELECT * FROM lakehouse_hv.customers_scd1 LIMIT 100")
print("--------------------------------customers scd1 Data--------------------------------")
display(df)
```

✓ 1 sec - Command executed in 1 sec 615 ms by harpal on 1:02:46 PM, 5/04/25

Spark jobs (2 of 2 succeeded) Resources

--------------------------------customers scd1 Data--------------------------------

Table + New chart

Table view

	customer_id	first_name	last_name	address	city	state	zip	hashkey	createdby	createddate	updatedby	updateddate
1	1	John	Doe	123 Elm St	Toronto	NULL	M4B1B3	3643162105	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
2	2	Jane	Smith	456 Maple A...	Ottawa	NULL	K1A0B1	3443782088	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
3	3	Michael	Johnson	789 Oak Dr	Montreal	NULL	H1A1A1	2706971633	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
4	4	Emily	Davis	101 Pine Rd	Calgary	NULL	T2A0A1	3528459452	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
5	5	David	Wilson	202 Birch Blvd	Vancouver	NULL	V5K0A1	1632120413	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
6	6	Emma	Clark	505 Cedar St	Halifax	NULL	B3H0A1	2294857924	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
7	7	James	Martinez	606 Spruce Ln	Winnipeg	NULL	R3C0A1	2767791804	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
8	8	Olivia	Garcia	707 Fir St	Edmonton	NULL	T5A0A1	555405233	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
9	9	William	Lopez	808 Redwoo...	Victoria	NULL	V8W0A1	3138603071	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
10	10	Ava	Anderson	909 Cypress ...	Quebec City	NULL	G1A0A1	236675979	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
11	11	Alexander	Thomas	1010 Willow...	St. John's	NULL	A1A0A1	2178920886	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
12	12	Isabella	Lee	1111 Poplar St	Fredericton	NULL	E3B0A1	3540855579	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
13	13	Daniel	Harris	1212 Ash Blvd	Charlotteto...	NULL	C1A0A1	3884251585	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
14	14	Sophia	Young	1313 Beech Dr	Yellowknife	NULL	X1A0A1	1480597731	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
15	15	Matthew	King	1414 Cedar Ln	Whitehorse	NULL	Y1A0A1	2530189205	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
16	16	Charlotte	Scott	1515 Elm St	Iqaluit	NULL	X0A0A1	394384885	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
17	17	Joseph	Green	1616 Maple ...	Regina	NULL	S4P0A1	2521270410	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...

Loans

```python
df = spark.sql("SELECT * FROM lakehouse_hv.loans_scd1 LIMIT 100")
print("--------------------------------loans scd1 Data--------------------------------")
display(df)
```

✓ 1 sec - Command executed in 1 sec 615 ms by harpal on 1:02:44 PM, 5/04/25

Spark jobs (2 of 2 succeeded) Resources

--------------------------------loans scd1 Data--------------------------------

Table + New chart

Table view

	loan_id	customer_id	loan_amount	interest_rate	loan_term	hashkey	createdby	createddate	updatedby	updateddate
1	1	45	10000.5	5.5	36	3926240992	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
2	2	12	20000.75	4.5	48	2129036693	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
3	3	78	15000.0	6.0	60	3887926970	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
4	4	34	30000.25	3.5	24	1883514680	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
5	5	56	25000.0	5.0	36	2321061703	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
6	6	23	17500.5	4.0	48	3578659751	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
7	7	89	22500.75	6.5	60	4273669208	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
8	8	67	27500.0	3.0	24	3761941096	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
9	9	14	32500.25	5.5	36	723673829	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
10	10	92	37500.5	4.5	48	2491094359	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
11	11	3	10000.75	6.0	60	3904343371	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
12	12	81	20000.0	3.5	24	1809043703	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
13	13	29	15000.25	5.0	36	363073924	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
14	14	64	30000.5	4.0	48	3727902380	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
15	15	47	25000.75	6.5	60	3891200344	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
16	16	18	17500.0	3.0	24	2605204923	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
17	17	99	22500.25	5.5	36	7870638	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...

Loan Payments

```
df = spark.sql("SELECT * FROM lakehouse_hv.loan_payments_scd1 LIMIT 100")
print("--------------------------------loan payments scd1 Data--------------------------------")
display(df)
```

✓ <1 sec - Command executed in 877 ms by harpal on 1:02:52 PM, 5/04/25

Spark jobs (1 of 1 succeeded) Resources

--------------------------------loan payments scd1 Data--------------------------------

Table | New chart

Table view

	payment_id	loan_id	payment_date	payment_amount	hashkey	createdby	createddate	updatedby	updateddate
1	1	45	2024-01-01	100.0	4140586677	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...
2	2	23	2024-01-02	150.0	2899073596	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...
3	3	67	2024-01-03	200.0	2599495225	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...
4	4	89	2024-01-04	250.0	1808870290	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...
5	5	12	2024-01-05	300.0	892704064	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...
6	6	34	2024-01-06	350.0	190405837	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...
7	7	56	2024-01-07	400.0	2107680105	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...
8	8	78	2024-01-08	450.0	4201767422	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...
9	9	90	2024-01-09	500.0	1927125073	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...
10	10	11	2024-01-10	550.0	2568269184	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...
11	11	22	2024-01-11	600.0	2114769633	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...
12	12	33	2024-01-12	650.0	4113161484	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...
13	13	44	2024-01-13	700.0	1022886523	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...
14	14	55	2024-01-14	750.0	3674148494	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...
15	15	66	2024-01-15	800.0	1243984276	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...
16	16	77	2024-01-16	850.0	3238051449	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...
17	17	88	2024-01-17	900.0	3469959988	fabric	2025-05-04 17:0...	fabric	2025-05-04 17:0...

Transactions

```
df = spark.sql("SELECT * FROM lakehouse_hv.transactions_scd1 LIMIT 100")
print("--------------------------------transactions scd1 Data--------------------------------")
display(df)
```

✓ 1 sec - Command executed in 1 sec 662 ms by harpal on 1:02:57 PM, 5/04/25

Spark jobs (2 of 2 succeeded) Resources

--------------------------------transactions scd1 Data--------------------------------

Table | New chart

Table view

	transaction_id	account_id	transaction_date	transaction_amount	transaction_type	hashkey	createdby	createddate	updatedby	updateddate
1	1	45	2024-01-01	100.5	Deposit	76340005	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
2	2	12	2024-01-02	200.75	Withdrawal	4106264961	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
3	3	78	2024-01-03	150.0	Deposit	1408726936	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
4	4	34	2024-01-04	300.25	Withdrawal	306793635	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
5	5	56	2024-01-05	250.0	Deposit	2501074206	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
6	6	23	2024-01-06	175.0	Withdrawal	950469601	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
7	7	89	2024-01-07	225.5	Deposit	3620667809	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
8	8	67	2024-01-08	275.75	Withdrawal	2035157507	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
9	9	14	2024-01-09	325.0	Deposit	3275121915	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
10	10	92	2024-01-10	375.25	Withdrawal	2577213654	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
11	11	3	2024-01-11	100.5	Deposit	2264570702	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
12	12	81	2024-01-12	200.75	Withdrawal	2126541435	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
13	13	29	2024-01-13	150.0	Deposit	840590155	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
14	14	64	2024-01-14	300.25	Withdrawal	3193770397	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
15	15	47	2024-01-15	250.0	Deposit	128844546	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
16	16	18	2024-01-16	175.0	Withdrawal	3355376620	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...
17	17	99	2024-01-17	225.5	Deposit	1298795972	fabric	2025-05-04 16:5...	fabric	2025-05-04 16:5...

Email Notification

Pipeline Execution Successful Inbox ×

Harpalsinh Vaghela <harpalsinh.vaghela@dcmail.ca>
to me

Hi Team,

The pipeline has been completed successfully. Please find the execution details below:

Pipeline Name: pl_on-prem_to_lakehouse_tables

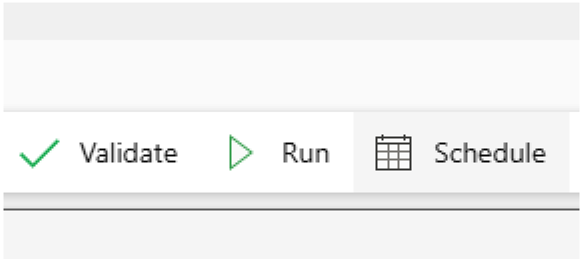
Pipeline ID: f4f0728e-04d9-4313-84f2-be94075313ba

Workspace ID: da4fe365-8cc9-4c2c-9002-0e665f345cb6

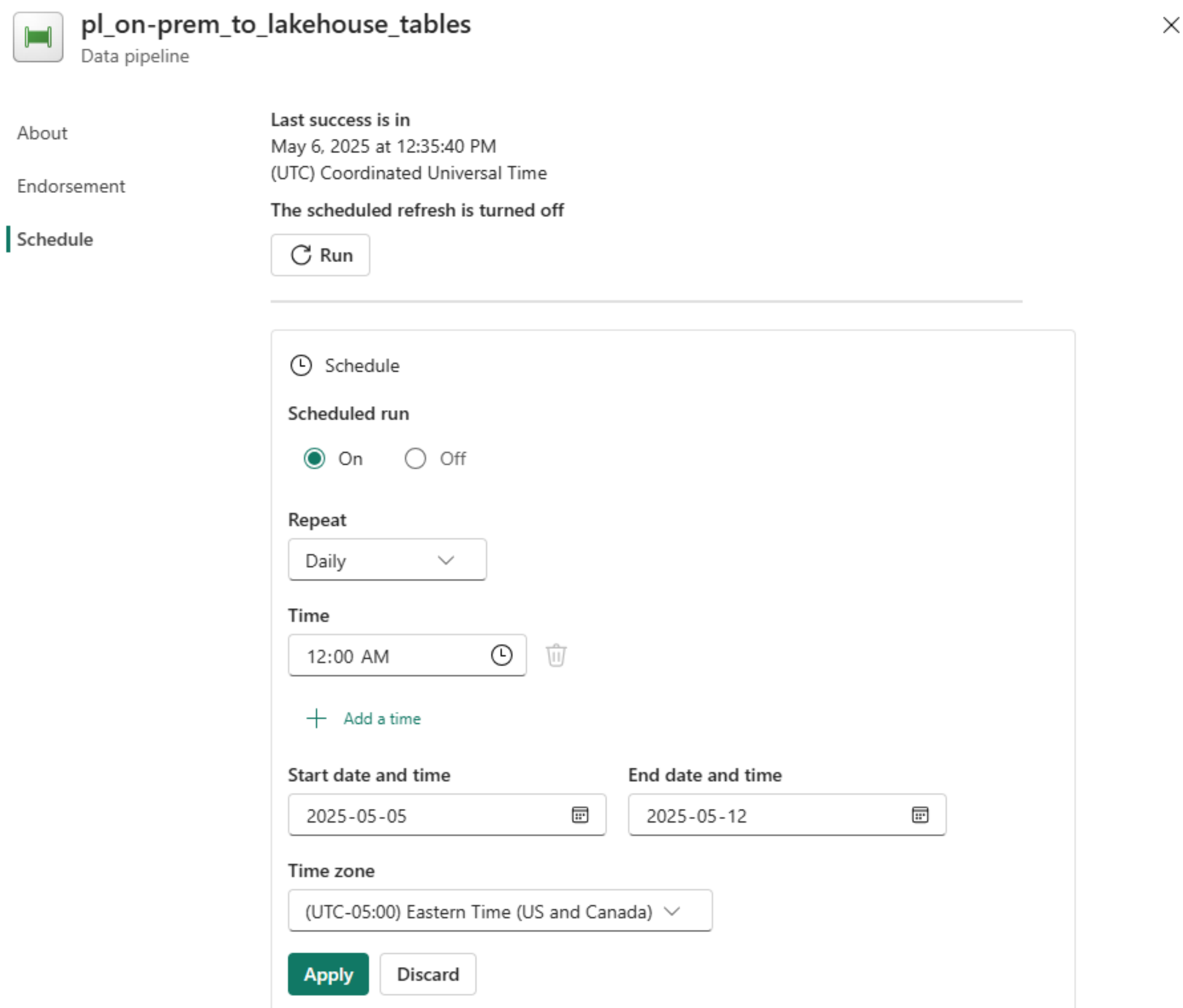
Thank you,
Harpalsinh Vaghela

Step 5: Schedule the Pipeline:

Pipeline -> Click on Schedule



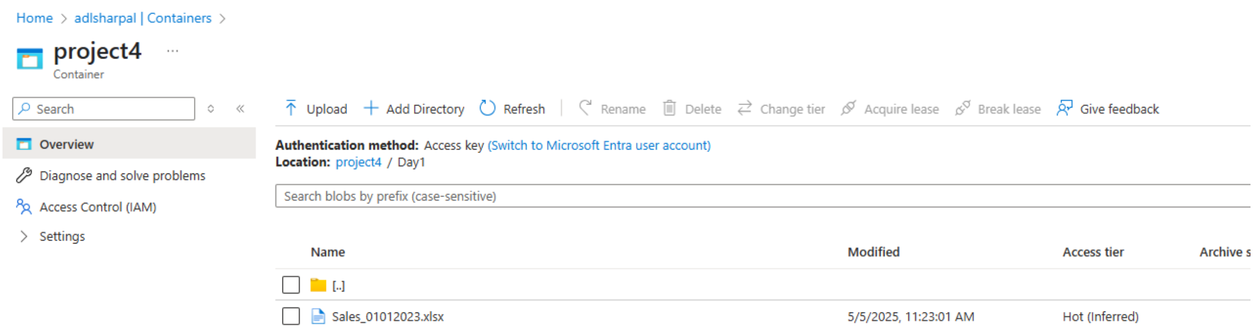
Configure according to need:



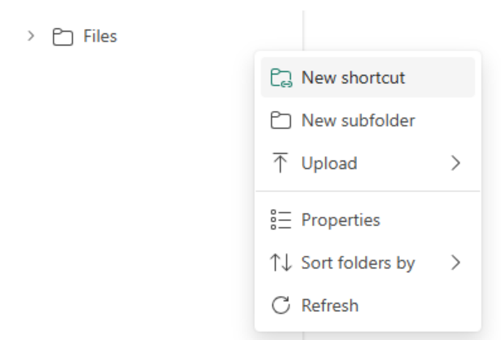
Task 2: Sales Data Modeling and Reporting Pipeline using Microsoft Fabric

Azure Home -> ADLS Gen 2 storage account

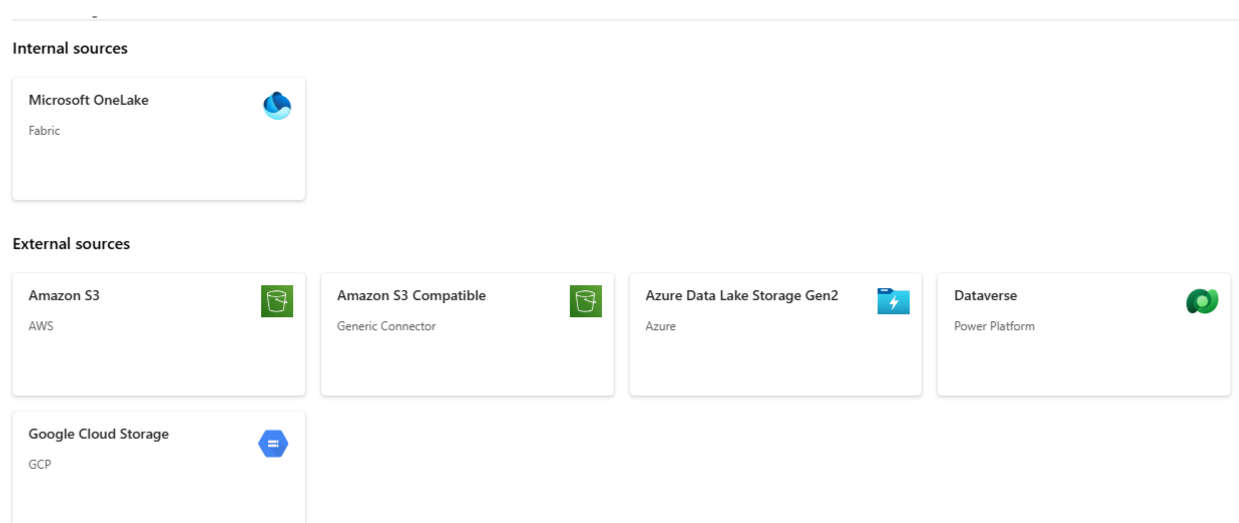
Upload the Sales Excel file, which has two sheets: “Sales” and “Returns”



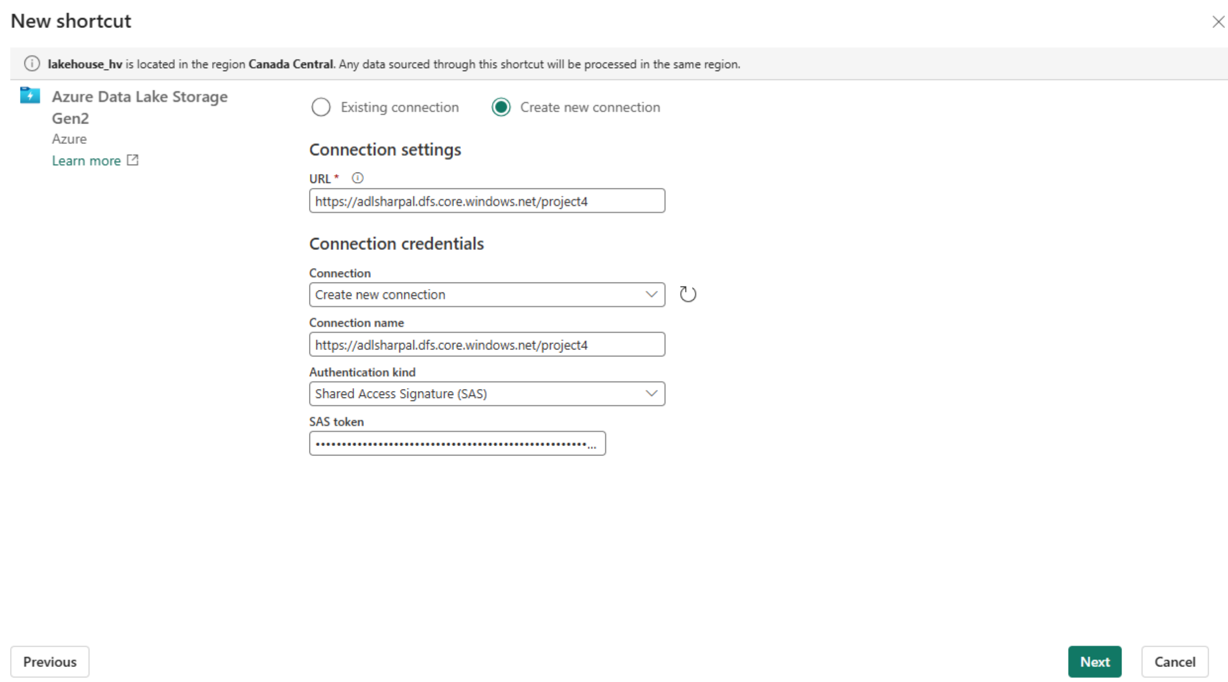
Go to Microsoft Fabric -> Lakehouse -> Files -> Create a new shortcut



Select ADLS Gen 2 Storage Account

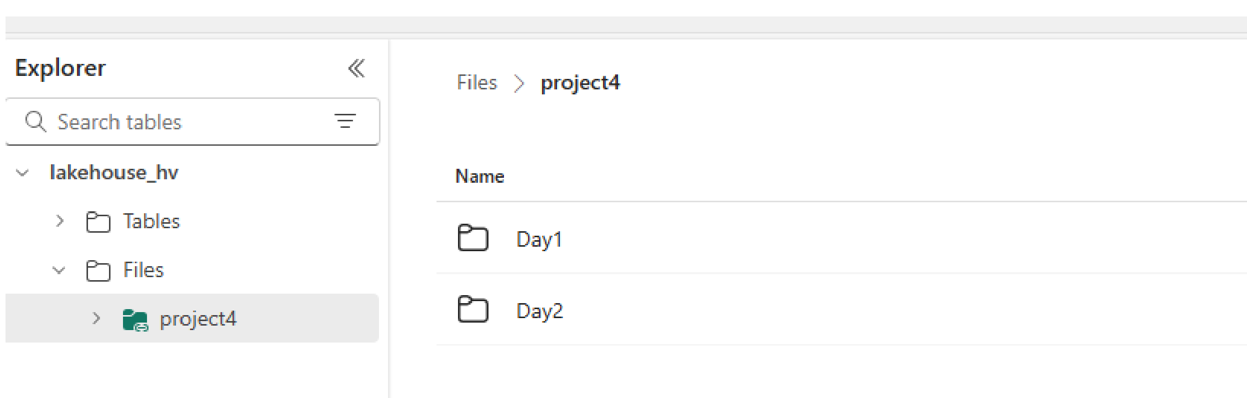


Copy ADLS Gen 2 Storage Account End point URL and SAS token in this dialog



Click on Next

Shortcut will be created into Files section of the lakehouse:



Fabric Workspace -> Create New Item -> Notebook

Here, we will read the Excel file.

All the required imports and libraries:

```
import com.microsoft.spark.fabric
from pyspark.sql.functions import col, round, regexp_replace, trim, when, year, month, quarter, weekofyear, to_date, expr, row_number, lit
from pyspark.sql.types import *
from datetime import datetime, timedelta
from pyspark.sql.window import Window
import pandas as pd
```

Read the data from the Sales Excel file "Sales" sheet

```
# Define the path to the Excel file inside Lakehouse
src_path = '/lakehouse/default/Files/project4/Day1/Sales_01012023.xlsx'

# Read the entire "Sales" sheet (all columns)
df = pd.read_excel(src_path, sheet_name='Sales')

# Convert to Spark DataFrame
spark_df = spark.createDataFrame(df)

# Display the Spark DataFrame
display(spark_df.limit(5))
```

- Command executed in 6 sec 280 ms by Fabric on 10:45:15 PM, 5/05/25 PySpark (Python)

Table | New chart | 20 columns, 5 rows

Table view | Download | Search

	Order_ID	Order_Date	Shipping_Date	Aging	Ship_Mode	Product_Category	Product	Sales	Quantity	Discount	Profit	Shipping_Cost	O
1	AU-2015-30	2023-01-23 00:0...	2023-01-27 00:00:00	4	First Class	Auto & Accessories	Car Body Co...	117	1	0.02	34.66	3.4659999999999997	High
2	AU-2015-50	2023-01-25 00:0...	2023-01-30 00:00:00	5	First Class	Auto & Accessories	Tyre	250	2	0.01	165.0	16.5	Medi
3	AU-2015-59	2023-01-06 00:0...	2023-01-12 00:00:00	6	First Class	Auto & Accessories	Tyre	250	1	0.04	160.0	16.0	Medi
4	AU-2015-81	2023-01-19 00:0...	2023-01-27 00:00:00	8	First Class	Auto & Accessories	Car Pillow & ...	231	1	0.03	144.07	14.407	High
5	AU-2015-97	2023-01-24 00:0...	2023-01-31 00:00:00	7	First Class	Auto & Accessories	Car Mat	54	5	0.02	10.8	1.08	High

Data Transformation Steps

- Converted into appropriate data types
- Filter out the null records
- Rounding values to two decimal points

```
# Remove duplicates
df_dedup = spark_df.dropDuplicates()

# Cast to appropriate data types
df_casted = df_dedup \
    .withColumn("Order_Date", col("Order_Date").cast(DateType())) \
    .withColumn("Shipping_Date", col("Shipping_Date").cast(DateType())) \
    .withColumn("Aging", col("Aging").cast(IntegerType())) \
    .withColumn("Sales", col("Sales").cast(DoubleType())) \
    .withColumn("Quantity", col("Quantity").cast(IntegerType())) \
    .withColumn("Discount", col("Discount").cast(DoubleType())) \
    .withColumn("Profit", col("Profit").cast(DoubleType())) \
    .withColumn("Shipping_Cost", col("Shipping_Cost").cast(DoubleType()))

# Filter out rows where critical columns are null
df_clean = df_casted.filter(
    col("Order_ID").isNotNull() &
    col("Order_Date").isNotNull() &
    col("Sales").isNotNull() &
    col("Customer_ID").isNotNull()
)

df_clean_rounded = df_clean \
    .withColumn("Shipping_Cost", round(col("Shipping_Cost"), 2)) \
    .withColumn("Sales", round(col("Sales"), 2)) \
    .withColumn("Profit", round(col("Profit"), 2)) \
    .withColumn("Discount", round(col("Discount"), 2))

# Display the cleaned DataFrame
display(df_clean_rounded.limit(5))
```

Output after Transformation:

[4] ✓ - Command executed in 3 sec 475 ms by Fabric on 10:45:19 PM, 5/05/25 PySpark (Python)

Table | New chart | 20 columns, 5 rows

Table view | Download | Search

	Order_ID	Order_Date	Shipping_Date	Aging	Ship_Mode	Product_Category	Product	Sales	Quantity	Discount	Profit	Shipping_Cost	Or
1	AU-2015-6087	2023-01-21	2023-01-28	7	First Class	Auto & Accessories	Car Speakers	211.0	4	0.02	114.12	11.41	Medi
2	AU-2015-4543	2023-01-19	2023-01-29	10	First Class	Auto & Accessories	Car Mat	54.0	2	0.03	27.0	2.7	High
3	AU-2015-216	2023-01-24	2023-01-29	5	First Class	Auto & Accessories	Car Pillow & ...	231.0	5	0.05	93.25	9.33	High
4	AU-2015-1670	2023-01-04	2023-01-13	9	First Class	Auto & Accessories	Tyre	250.0	3	0.02	155.0	15.5	Critic
5	AU-2015-1763	2023-01-25	2023-01-27	2	First Class	Auto & Accessories	Car Seat Cov...	114.0	2	0.01	31.72	3.17	Medi

Read the second sheet “Returns” in that Excel file

```
# Define the path to the Excel file inside Lakehouse
src_path = '/lakehouse/default/Files/project4/Day1/Sales_01012023.xlsx'

# Read the entire "Returns" sheet (all columns)
df = pd.read_excel(src_path, sheet_name='Returns')

# Convert to Spark DataFrame
spark_df = spark.createDataFrame(df)

# Display the Spark DataFrame
display(spark_df.limit(5))
```

✓ - Command executed in 806 ms by Fabric on 10:45:20 PM, 5/05/25 PySp

Table | New chart | 4 columns, 5

Table view | Download | Search

	Order_ID	Customer_Name	Return	Sales_Amount
1	AU-2015-30	Poole Lucas	Yes	117
2	AU-2015-50	Hernandez Badders	Yes	250
3	AU-2015-59	Kelly Braden	Yes	250
4	AU-2015-97	Hatfield Trafton	Yes	54
5	AU-2015-124	Bullock Prost	Yes	54

Transformation steps on “Returns” Data

- Convert 'Sales_Amount' from string ($250.00) to float
- Filter out the null records

```python
# Remove duplicates (if any)
df_return_dedup = spark_df.dropDuplicates()

# Clean up and convert 'Sales_Amount' from string ($250.00) to float
df_return_cleaned = df_return_dedup \
    .withColumn("Sales_Amount", regexp_replace(col("Sales_Amount"), "[$]", "").cast("double")) \
    .withColumn("Order_ID", trim(col("Order_ID"))) \
    .withColumn("Customer_Name", trim(col("Customer_Name"))) \
    .withColumn("Return", trim(col("Return")))

# Filter rows where key fields are not null
df_return_final = df_return_cleaned.filter(
    col("Order_ID").isNotNull() & col("Customer_Name").isNotNull() & col("Sales_Amount").isNotNull()
)

# Display result
display(df_return_final.limit(5))
```

- Command executed in 1 sec 485 ms by Fabric on 10:45:21 PM, 5/05/25 PyS

Table + New chart 4 columns, 5

Table view Download Search

	Order_ID	Customer_Name	Return	Sales_Amount
1	AU-2015-30	Poole Lucas	Yes	117.0
2	AU-2015-50	Hernandez Badders	Yes	250.0
3	AU-2015-97	Hatfield Trafton	Yes	54.0
4	AU-2015-59	Kelly Braden	Yes	250.0
5	AU-2015-124	Bullock Prost	Yes	54.0

Create **Date** Dimension Table

- Generates a 10-year date range from 2020 to 2030.
- Creates a DataFrame with business calendar columns like year, month, quarter, and week.
- Adds a surrogate key Date_ID using row_number based on the business date.
- Writes the final date dimension table to the Fabric Warehouse.

```python
# -----------------------------------------Dimension - Date Table---------------------------------------------------

# Generate a 10-year date range
start_date = datetime(2020, 1, 1)
end_date = datetime(2030, 12, 31)
date_list = [start_date + timedelta(days=x) for x in range((end_date - start_date).days + 1)]

# Convert to Spark DataFrame with a single column
df_date = spark.createDataFrame([(d,) for d in date_list], ["BusinessDate"])

# Add business calendar columns
df_date = df_date \
    .withColumn("Business_Year", year("BusinessDate")) \
    .withColumn("Business_Month", month("BusinessDate")) \
    .withColumn("Business_Quarter", quarter("BusinessDate")) \
    .withColumn("Business_Week", weekofyear("BusinessDate"))

# Add Date_ID as surrogate key
df_date_with_id = df_date.withColumn(
    "Date_ID", row_number().over(Window.orderBy("BusinessDate"))
)

# Reorder columns
df_date_with_id = df_date_with_id.select(
    "Date_ID", "BusinessDate", "Business_Year", "Business_Month", "Business_Quarter", "Business_Week"
)

# Write to Warehouse as table
df_date_with_id.write.mode('overwrite').synapsesql('warehouse_hv.sales.dim_date')
```

Create **Segment** Dimension Table

- Extracts unique segment values from the cleaned sales data.
- Generates a surrogate key Segment_ID using row_number.
- Selects Segment_ID and Segment columns for the dimension table.
- Saves the segment dimension table to the Fabric Warehouse.

```
# -------------------------------------------Dimension - Segment-----------------------------------------------------
df_segment = df_clean_rounded.select("Segment").dropDuplicates()

df_segment_dim = df_segment.withColumn(
    "Segment_ID", row_number().over(Window.orderBy("Segment"))
)

df_segment_dim = df_segment_dim.select("Segment_ID", "Segment")

# Write to Warehouse as table
df_segment_dim.write.mode('overwrite').synapsesql('warehouse_hv.sales.dim_segment')
```

Create **Customer** Segment Table

- Selects customer-related columns and removes duplicates from the cleaned dataset.
- Joins with dim_segment to map each customer to a Segment_ID.
- Normalizes by dropping the original Segment column and keeping Segment_ID.
- Writes the final customer dimension table to the Fabric Warehouse.

```
# ----------------------------------------Dimension - Customer--------------------------------------------------
dim_customer = df_clean_rounded.select(
    "Customer_ID", "Customer_Name", "Segment", "City", "State", "Country", "Region"
).dropDuplicates()

# Join with dim_segment to add Segment_ID
dim_customer_with_segment_id = dim_customer.join(
    df_segment_dim, on="Segment", how="left"
)

# Optional: drop Segment column to normalize
dim_customer_final = dim_customer_with_segment_id.select(
    "Customer_ID", "Customer_Name", "Segment_ID", "City", "State", "Country", "Region"
)

# Write to Warehouse as table
dim_customer_final.write.mode('overwrite').synapsesql('warehouse_hv.sales.dim_customer')
```

Create **Product** Dimension Table

- Removes duplicate entries based on the Product column from the cleaned dataset.
- Adds a Product_ID as a surrogate key using row_number() ordered by product name.
- Selects relevant columns for the product dimension table.
- Saves the dimension table to the Fabric Warehouse.

```
# -------------------------------------------Dimension - Product----------------------------------------------------
df_product = df_clean_rounded.dropDuplicates(["Product"])

#Add ProductID Column
df_product_dim = df_product.withColumn(
    "Product_ID", row_number().over(Window.orderBy("Product"))
)

df_product_dim = df_product_dim.select("Product_ID", "Product", "Product_Category")

# Write to Warehouse as table
df_product_dim.write.mode('overwrite').synapsesql('warehouse_hv.sales.dim_product')
```

Create **Order** Dimension Table

- Extracts unique order records including dates, shipping info, and aging.
- Joins with the date dimension twice to replace Order_Date and Shipping_Date with surrogate keys.
- Selects normalized columns including Order_Date_ID and Shipping_Date_ID.
- Saves the finalized order dimension table to the Fabric Warehouse.

```
# -------------------------------------------Dimension - Order---------------------------------------------------

# Extract order table
dim_order_raw = df_clean_rounded.select(
    "Order_ID", "Order_Date", "Shipping_Date", "Ship_Mode", "Order_Priority", "Aging"
).dropDuplicates()

# Join with dim_date twice (for Order_Date and Shipping_Date)
dim_order_with_order_date_id = dim_order_raw \
    .join(df_date_with_id.withColumnRenamed("BusinessDate", "Order_Date"), on="Order_Date", how="left") \
    .withColumnRenamed("Date_ID", "Order_Date_ID")

dim_order_with_both_dates = dim_order_with_order_date_id \
    .join(df_date_with_id.withColumnRenamed("BusinessDate", "Shipping_Date"), on="Shipping_Date", how="left") \
    .withColumnRenamed("Date_ID", "Shipping_Date_ID")

# Select final normalized columns
dim_order_final = dim_order_with_both_dates.select(
    "Order_ID",
    "Order_Date_ID",
    "Shipping_Date_ID",
    "Ship_Mode",
    "Order_Priority",
    "Aging"
)

dim_order_final.write.mode('overwrite').synapsesql('warehouse_hv.sales.dim_order')
```

Create **Sales_Return** Dimension Table (Cleaning and Mapping on "Returns" Data)

- Removes duplicates and trims whitespaces from important columns.
- Cleans Sales_Amount by removing currency symbols and casting to numeric type.
- Maps Return values to Return_ID using binary logic (Yes → 1, No → 0).
- Writes the cleaned and normalized sales return dimension table to the warehouse.

```
# Clean and map Return column
sales_return_cleaned = df_return_final.dropDuplicates() \
    .withColumn("Order_ID", trim(col("Order_ID"))) \
    .withColumn("Customer_Name", trim(col("Customer_Name"))) \
    .withColumn("Return_ID", when(trim(col("Return")) == "Yes", 1).otherwise(0)) \
    .withColumn("Sales_Amount", regexp_replace(col("Sales_Amount"), "[$]", "").cast("double"))

# Filter out rows with missing critical fields
sales_return_filtered = sales_return_cleaned.filter(
    col("Order_ID").isNotNull() &
    col("Customer_Name").isNotNull() &
    col("Sales_Amount").isNotNull()
)

# Map Return to 1/0
sales_return_mapped = sales_return_cleaned.withColumn(
    "Return_ID", when(col("Return") == "Yes", 1).otherwise(0)
)

dim_sales_return_final = sales_return_mapped.select(
    "Order_ID", "Customer_Name", "Sales_Amount", "Return_ID"
)

# Write to Warehouse table
dim_sales_return_final.write.mode('overwrite').synapsesql('warehouse_hv.sales.dim_sales_return')
display(dim_sales_return_final.limit(5))
```

✓ - Command executed in 4 sec 766 ms by Fabric on 10:46:12 PM, 5/05/25

Table + New chart

Table view Download

	Order_ID	Customer_Name	Sales_Amount	Return_ID
1	AU-2015-97	Hatfield Trafton	54.0	1
2	AU-2015-50	Hernandez Badders	250.0	1
3	AU-2015-124	Bullock Prost	54.0	1
4	AU-2015-59	Kelly Braden	250.0	1
5	AU-2015-152	Andrews Daniels	114.0	1

Create a **Fact** Table (fact_sales)

- Loads all required dimension tables from the Fabric Warehouse.
- Joins the cleaned sales data with product, customer, order, and return dimension tables.
- Ensures Return_ID is not null by assigning 0 for missing values.
- Selects key foreign IDs and measures to form the final fact table and writes it to the warehouse.

```python
# Read dimension tables from Fabric Warehouse
dim_customer = spark.read.synapsesql("warehouse_hv.sales.dim_customer")
dim_product = spark.read.synapsesql("warehouse_hv.sales.dim_product")
dim_order = spark.read.synapsesql("warehouse_hv.sales.dim_order")
dim_sales_return = spark.read.synapsesql("warehouse_hv.sales.dim_sales_return")
dim_segment = spark.read.synapsesql("warehouse_hv.sales.dim_segment")
dim_date = spark.read.synapsesql("warehouse_hv.sales.dim_date")


# Add Product_ID
fact_joined = df_clean_rounded \
    .join(dim_product.select("Product", "Product_ID"), on="Product", how="left") \
    .join(dim_customer.select("Customer_ID", "Segment_ID"), on="Customer_ID", how="left") \
    .join(dim_order.select("Order_ID", "Order_Date_ID", "Shipping_Date_ID"), on="Order_ID", how="left") \
    .join(dim_sales_return.select("Order_ID", "Return_ID"), on="Order_ID", how="left")

fact_joined = fact_joined.withColumn("Return_ID", when(col("Return_ID").isNull(), 0).otherwise(col("Return_ID")))

# -------------------------------- Final Fact Table --------------------------------
fact_sales_final = fact_joined.select(
    "Order_ID",
    "Customer_ID",
    "Segment_ID",
    "Product_ID",
    "Order_Date_ID",
    "Shipping_Date_ID",
    "Sales",
    "Profit",
    "Quantity",
    "Discount",
    "Shipping_Cost",
    "Return_ID"
)

# Save fact table to Warehouse
fact_sales_final.write.mode('overwrite').synapsesql('warehouse_hv.sales.fact_sales')
```

Check few records from Fact Table

```python
display(fact_sales_final.limit(5))
```

✓ - Command executed in 2 sec 434 ms by Fabric on 10:48:29 PM, 5/05/25 PySpark (Py

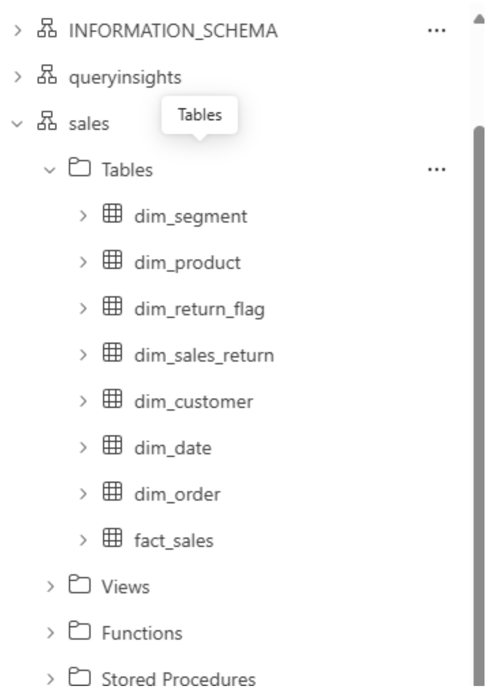
Table + New chart 12 columns, 5 rows

Table view Download Search

	Order_ID	Customer_ID	Segment_ID	Product_ID	Order_Date_ID	Shipping_Date_ID	Sales	Profit	Quantity	Discount	Shipping_Cost	Return_ID
1	AU-2015-1763	ER-001763	2	10	1121	1123	114.0	31.72	2	0.01	3.17	1
2	AU-2015-216	AN-00216	3	9	1120	1125	231.0	93.25	5	0.05	9.33	0
3	AU-2015-4543	RK-004543	2	7	1115	1125	54.0	27.0	2	0.03	2.7	0
4	AU-2015-1670	BS-001670	1	40	1100	1109	250.0	155.0	3	0.02	15.5	0
5	AU-2015-6087	ER-006087	3	11	1117	1124	211.0	114.12	4	0.02	11.41	0

Go to **Warehouse**

We can see all the tables loaded here:

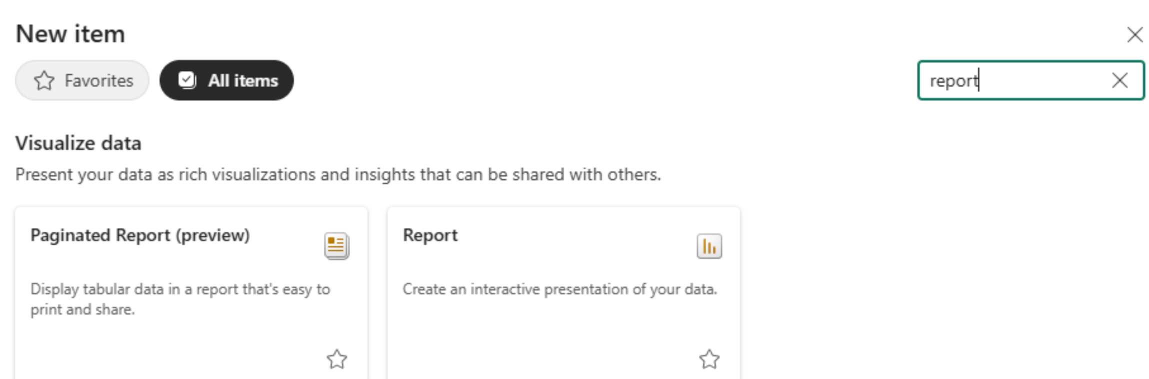


Go to Queries -> My queries -> New SQL query

Here, we will create a view that can be used to create visuals in Power BI.

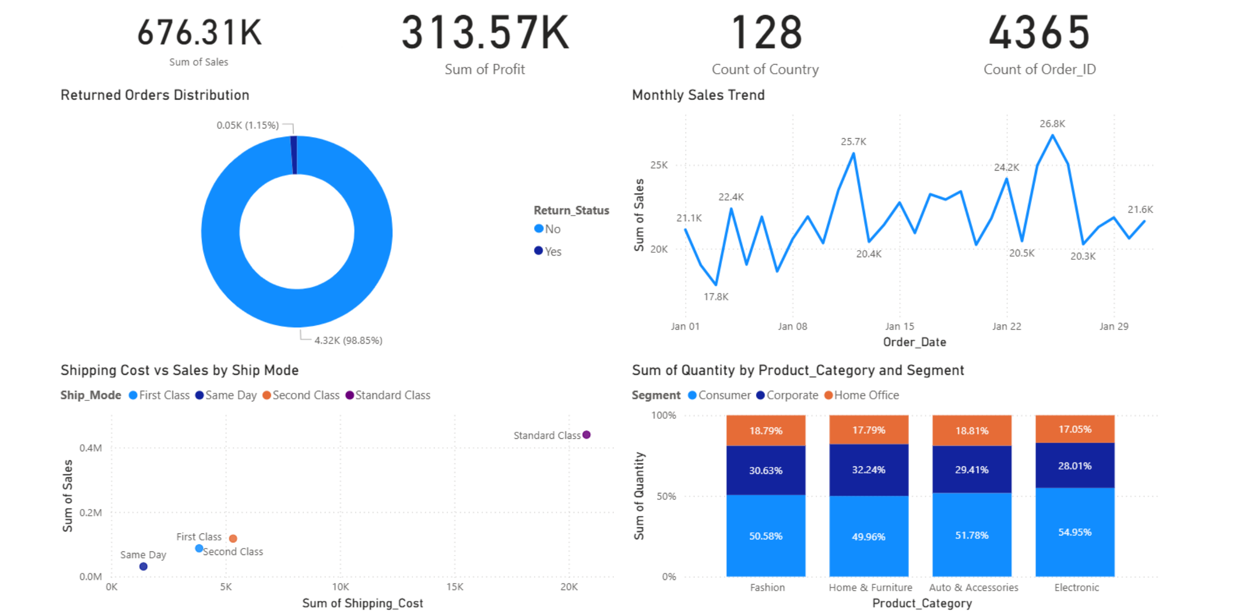
```sql
CREATE OR ALTER VIEW sales.view_sales_details AS
SELECT
    fs.Order_ID,
    fs.Customer_ID,
    dc.Customer_Name,
    ds.Segment,
    dc.City,
    dc.[State],
    dc.Country,
    dc.Region,
    dp.Product,
    dp.Product_Category,
    fs.Sales,
    fs.Profit,
    fs.Quantity,
    fs.Discount,
    fs.Shipping_Cost,
    CASE dsr.Return_ID
        WHEN 1 THEN 'Yes'
        ELSE 'No'
    END AS Return_Status,
    dord.BusinessDate AS Order_Date,
    dship.BusinessDate AS Shipping_Date,
    do.Order_Priority,
    do.Ship_Mode

FROM
    sales.fact_sales fs
LEFT JOIN sales.dim_customer dc ON fs.Customer_ID = dc.Customer_ID
LEFT JOIN sales.dim_segment ds ON dc.Segment_ID = ds.Segment_ID
LEFT JOIN sales.dim_product dp ON fs.Product_ID = dp.Product_ID
LEFT JOIN sales.dim_order do ON fs.Order_ID = do.Order_ID
LEFT JOIN sales.dim_date dord ON fs.Order_Date_ID = dord.Date_ID
LEFT JOIN sales.dim_date dship ON fs.Shipping_Date_ID = dship.Date_ID
LEFT JOIN sales.dim_sales_return dsr ON fs.Order_ID = dsr.Order_ID;
```

Go to Workspace -> New Item -> Search for Report



Click on Create a blank report import view as dataset/table

Created the dashboard as shown below:



Points to remember:

- All Excel and CSV files (Sales and Returns) were read, cleaned, and transformed using Fabric Notebooks before loading into tables.
- Turn on the fabric capacity before doing any work on fabric workspace.
- Dimension Modeling: Multiple dimension tables were created—dim_customer, dim_product, dim_order, dim_date, dim_segment, and dim_sales_return—following star schema best practices.
- Surrogate keys like Date_ID, Segment_ID, and Product_ID were generated using row_number() to ensure normalization and consistent joins.
- Cleaning steps included trimming strings, removing duplicates, handling nulls, and casting fields (e.g., converting currency fields to double).
- The fact table includes only numeric measures and foreign keys from all relevant dimension tables (not raw text fields).
- dim_date provides a full calendar reference, enabling reporting by year, month, quarter, and week using Order_Date_ID and Shipping_Date_ID.
- A view (view_sales_details) was created in the warehouse to simplify Power BI queries and unify all descriptive fields via joins.
- All cleaned dimensions and the final fact table were written to the Fabric Warehouse using .synapsesql() for downstream reporting.