Taksh Shah

Project – 3: Customer 360 Data Integration

Overview:

A retail business wants to build a unified Customer 360 view by integrating data from multiple sources, including online transactions, in-store purchases, customer service interactions, and loyalty programs. This project uses a mix of fact and dimension tables to ensure a clean, scalable structure.

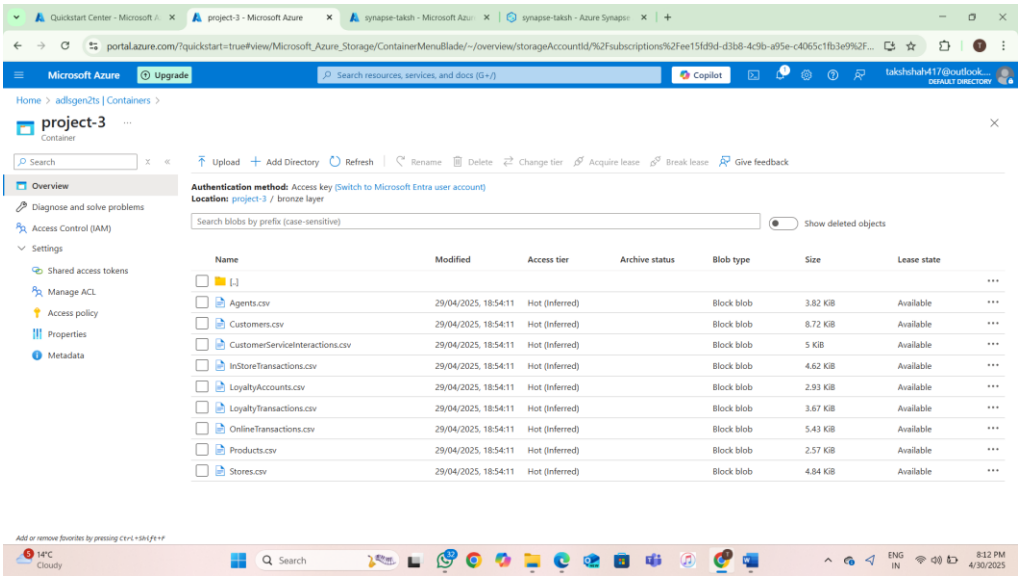
Dataset used: https://www.kaggle.com/datasets/varunkumari/customer-360-data

Tools used:

- Azure Synapse Analytics
- Azure Data Lake Gen2 Storage
- Azure SQL Database
- Azure Keyvault
- Power BI
- Microsoft Fabric

Bronze Layer:

In this layer, we are simply uploading all the files of the dataset from the local system into a folder called bronze layer into the adlsgen2 storage account container.



Silver Layer:

In this layer, we'll be cleaning the data (remove null and duplicate values) using dataflows by fetching data from the bronze layer folder in adlsgen2 storage account and eventually storing it in our azure sql database.

First, let's create the tables in which we'll store our data in the sql database. The queries for the tables were already provided to us as follows:

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100),
    Address VARCHAR(255)
);

CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    Name VARCHAR(100),
    Category VARCHAR(50),
    Price DECIMAL(10, 2)
);

CREATE TABLE OnlineTransactions (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    ProductID INT,
    DateTime DATETIME,
    PaymentMethod VARCHAR(50),
    Amount DECIMAL(10, 2),
    Status VARCHAR(20),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);

CREATE TABLE Stores (
    StoreID INT PRIMARY KEY,
    Location VARCHAR(100),
    Manager VARCHAR(100),
    OpenHours VARCHAR(50)
);
```

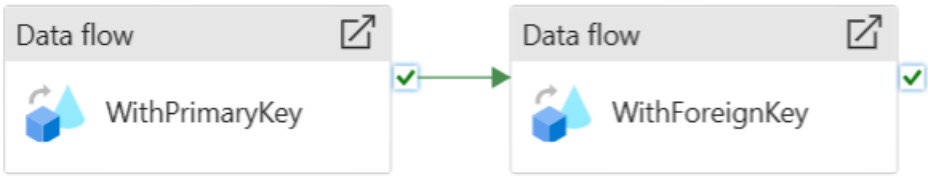
```sql
CREATE TABLE InStoreTransactions (
    TransactionID INT PRIMARY KEY,
    CustomerID INT,
    StoreID INT,
    DateTime DATETIME,
    Amount DECIMAL(10, 2),
    PaymentMethod VARCHAR(50),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (StoreID) REFERENCES Stores(StoreID)
);

CREATE TABLE Agents (
    AgentID INT PRIMARY KEY,
    Name VARCHAR(100),
    Department VARCHAR(50),
    Shift VARCHAR(50)
);

CREATE TABLE CustomerServiceInteractions (
    InteractionID INT PRIMARY KEY,
    CustomerID INT,
    DateTime DATETIME,
    AgentID INT,
    IssueType VARCHAR(50),
    ResolutionStatus VARCHAR(50),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (AgentID) REFERENCES Agents(AgentID)
);

CREATE TABLE LoyaltyAccounts (
    LoyaltyID INT PRIMARY KEY,
    CustomerID INT,
    PointsEarned INT,
    TierLevel VARCHAR(20),
    JoinDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

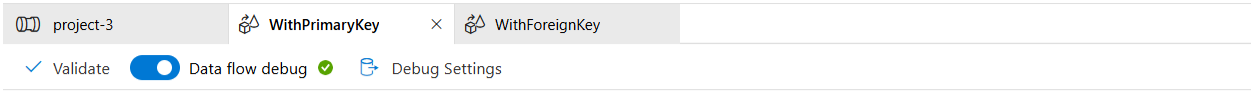
```
CREATE TABLE LoyaltyTransactions (
    LoyaltyID INT,
    DateTime DATETIME,
    PointsChange INT,
    Reason VARCHAR(100),
    PRIMARY KEY (LoyaltyID, DateTime),
    FOREIGN KEY (LoyaltyID) REFERENCES LoyaltyAccounts(LoyaltyID)
);
```

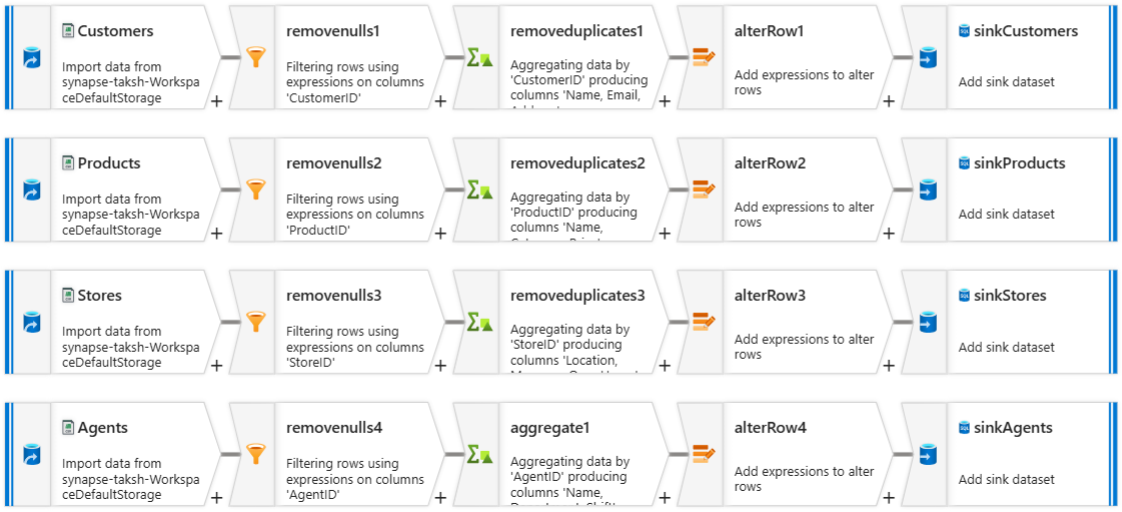
We'll create the tables in the same order as given here as there are dependencies between some of them.

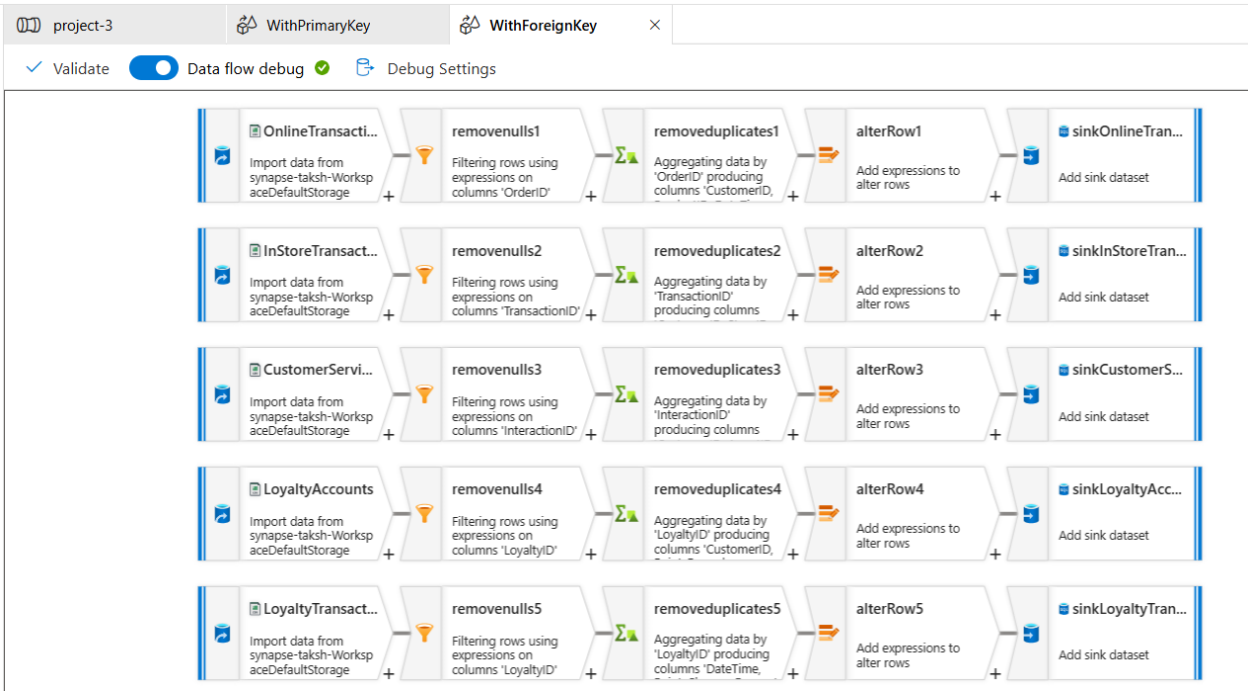
We'll create 2 dataflows – one for all tables that have primary key and one for all the tables that have foreign keys in their schema and connect both the dataflows.

The data flow transformations for all source csv files will be the same so we'll discuss 1 of them below for our understanding.

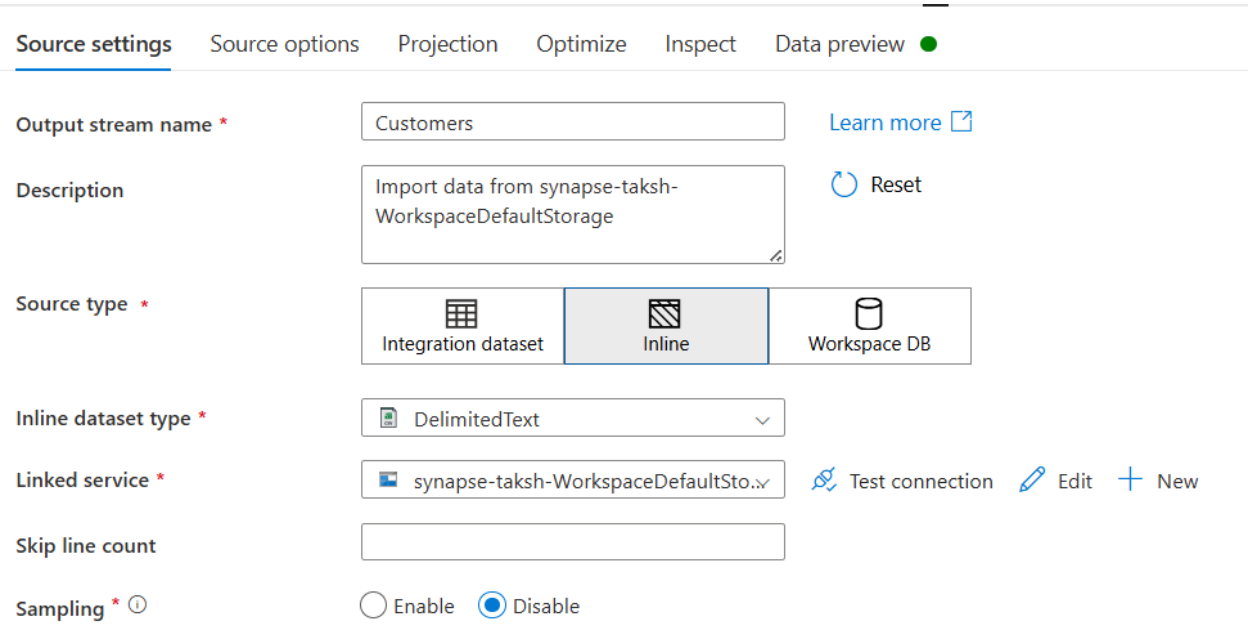




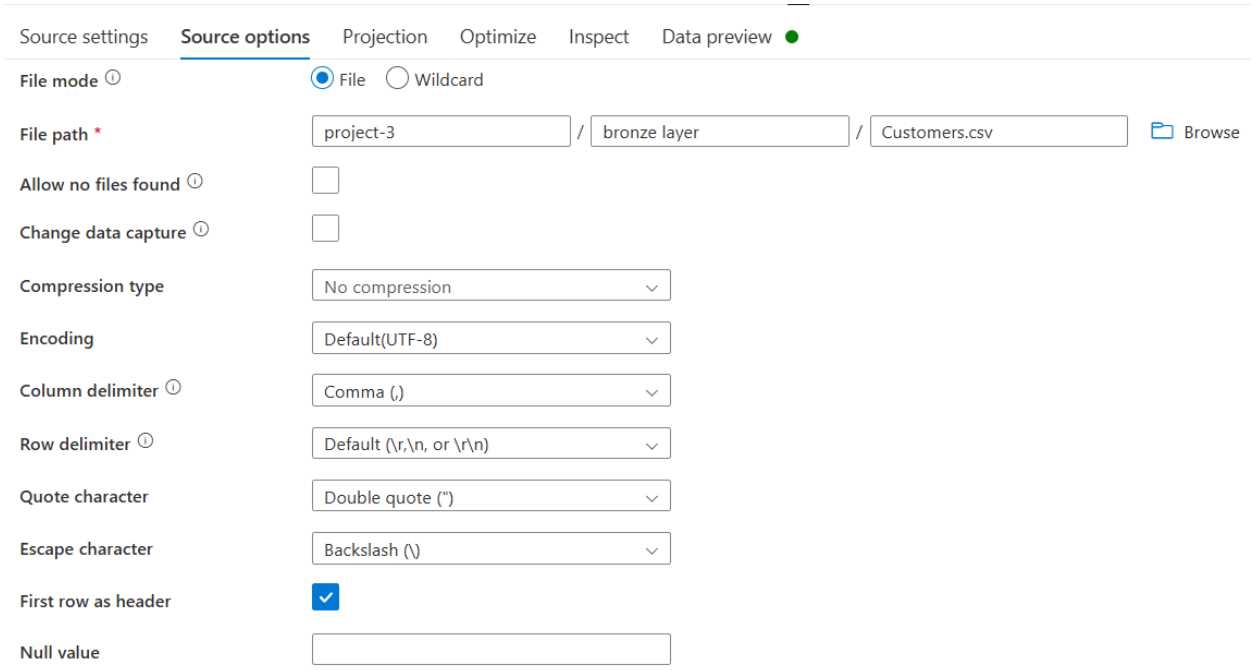




Let's dicuss the 1st table of withprimarykey dataflow, ie, customers table.

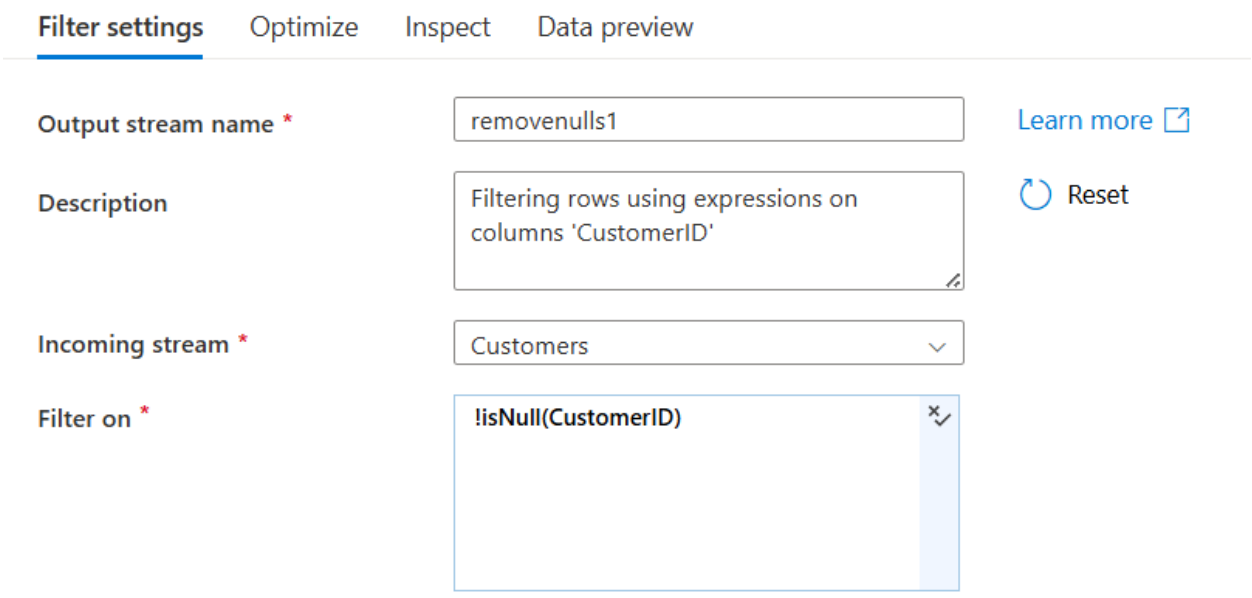


First, we configure the source by setting source type as inline and dataset type as delimited text as it's a csv file. Additionally, we'll select the linked service for connecting adlsgen2 storage account and synapse workspace.



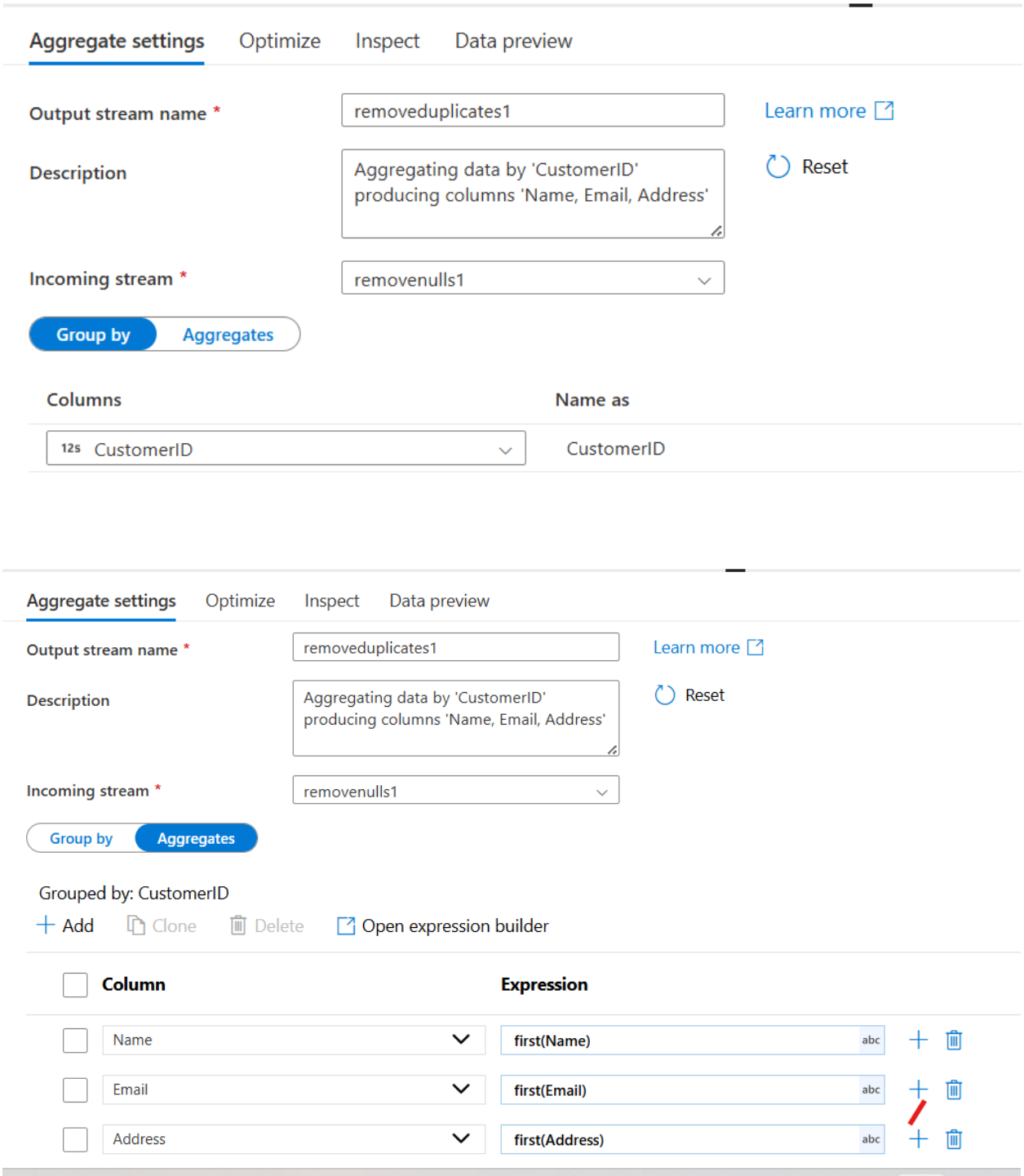
Under source options, we'll provide the path where our source csv file is located (adlsgen2 bronze layer folder) and make sure to select first row as header option.

Next, we connect the source with a filter transformation to remove null values from the source file.



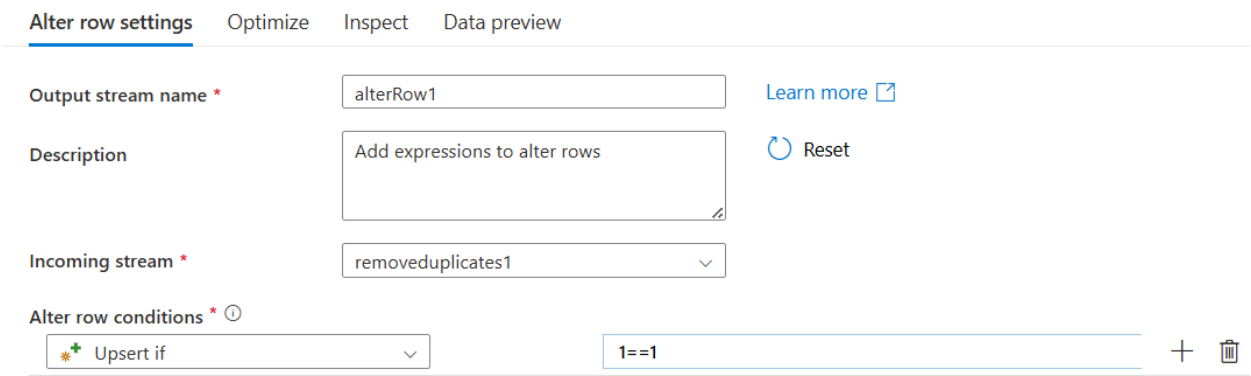
Here, we're checking to see if there are null values in primary key column.

Now, we are connecting the filter transformation to an aggregate transformation.

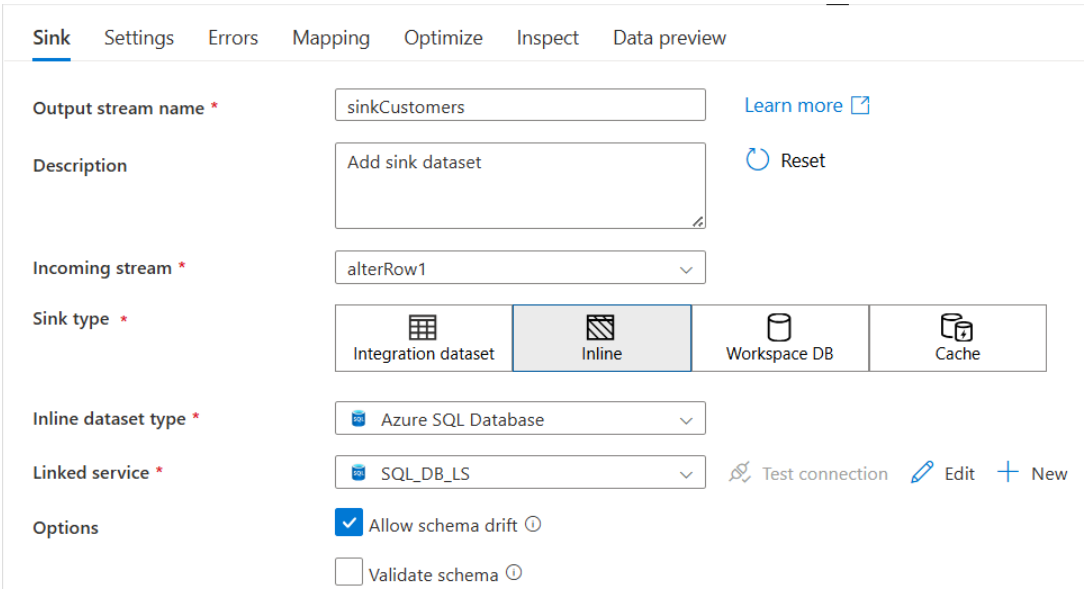


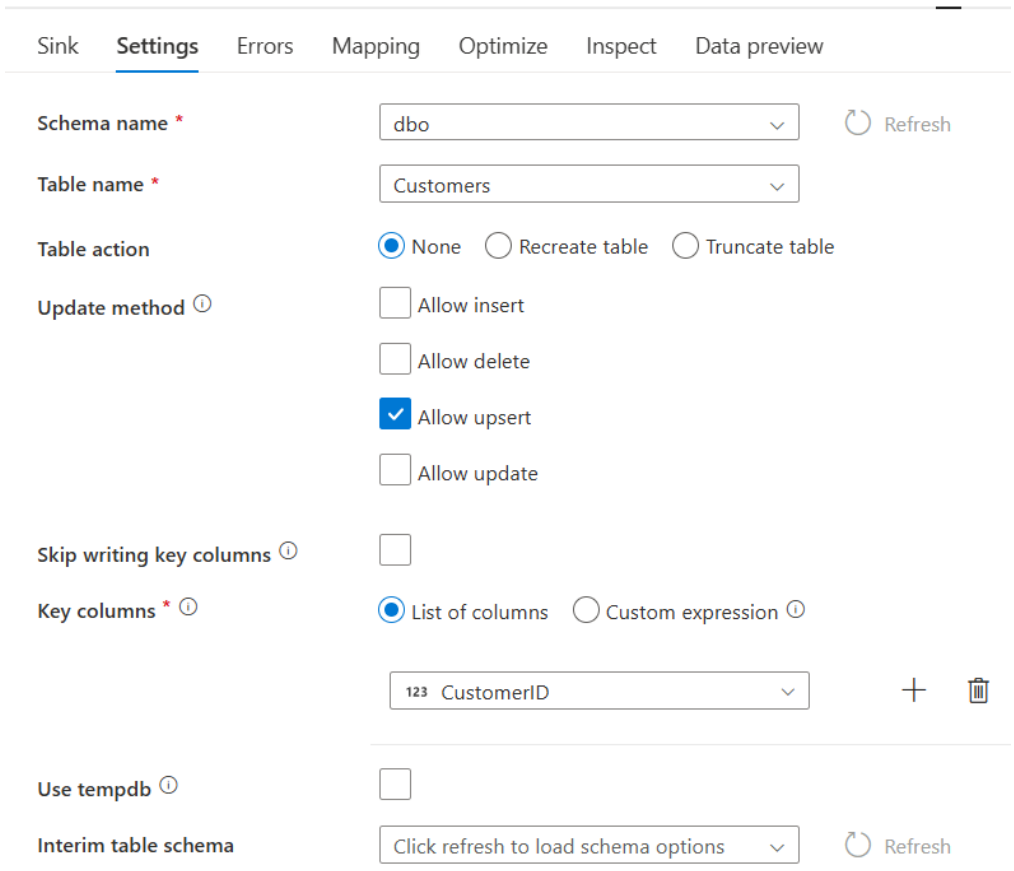
Here, “aggregates” section must have at least 1 column as that’s a mandatory requirement for the aggregate transformation. What we’ll do is provide the key column for “group by” section, meaning we want to group the data in the source file based on the key column, so in our case, since CustomerID is the key column, this means that we’ll group (or check) the data for each customer. Subsequently, we’ll provide all the other columns in the “aggregates” section. This essentially means that we’ll group the data to check some aggregates. In the expression field under “aggregates” section, we utilize the first() function which returns the first non-null value in a column.

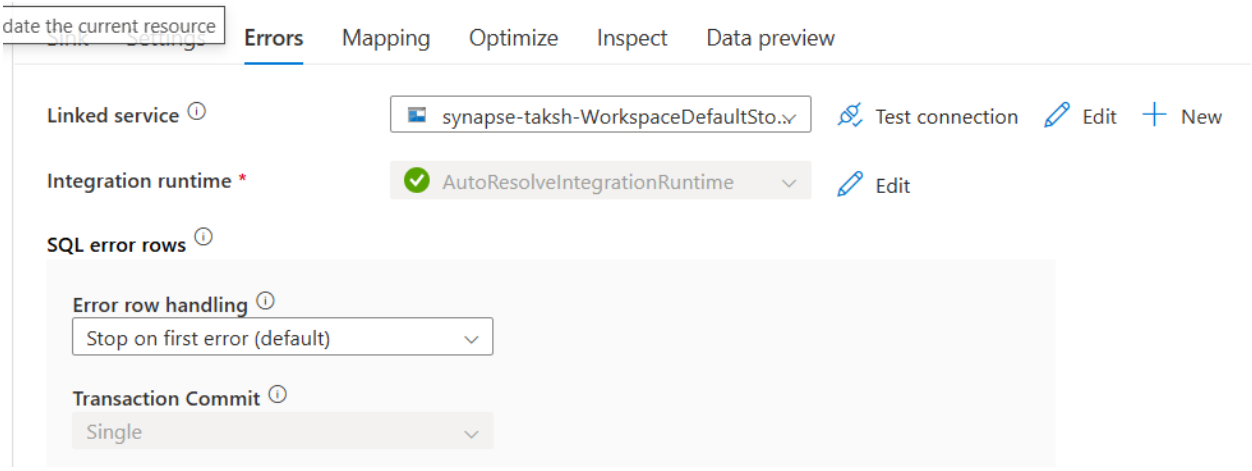
Next, we connect it to an alter row transformation, where we basically provide permission to the dataflow to access and make changes to our sink, which will be azure sql database. We provide “upsert if 1==1” condition.

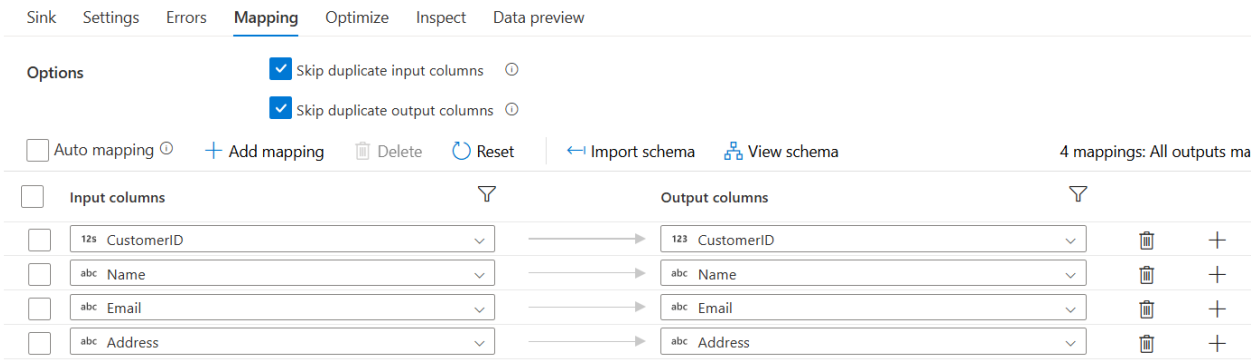


Finally, we connect it to a sink and configure it to point to our azure sql database as per the images provided below:

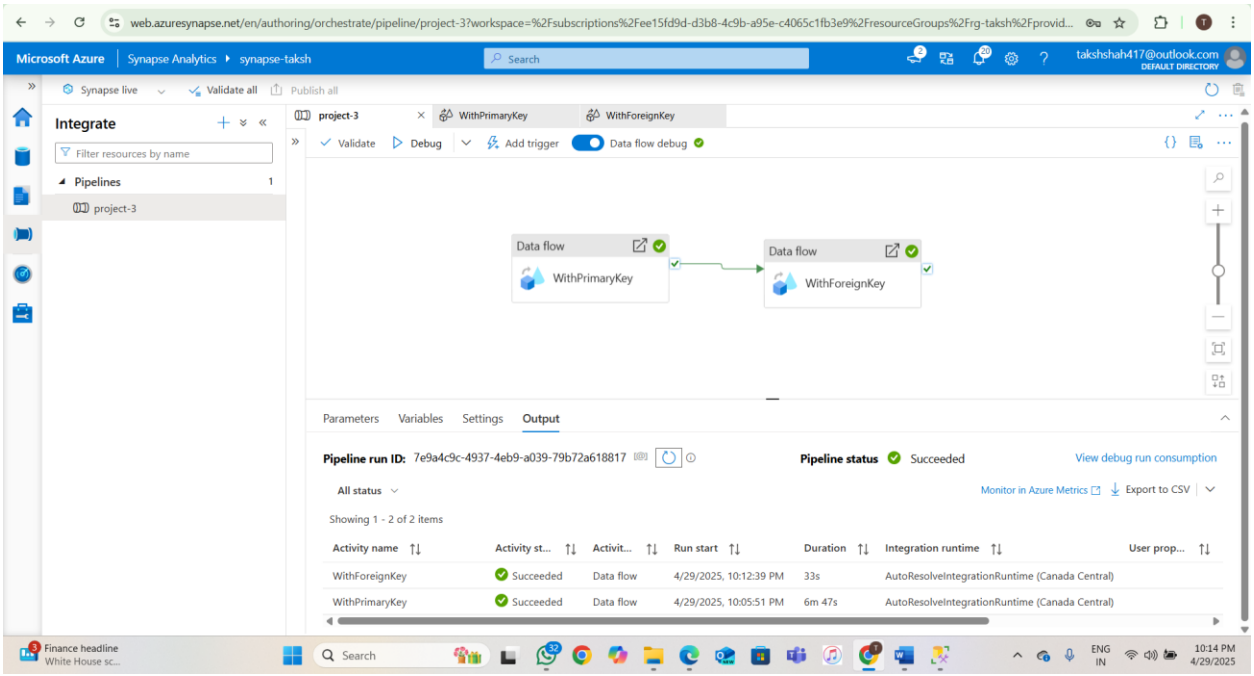



Sink
Settings
Errors
Mapping
Optimize
Inspect
Data preview
Schema name *
dbo
Refresh
Table name *
Customers
Table action
None
Recreate table
Truncate table
Update method
Allow insert
Allow delete
Allow upsert
Allow update
Skip writing key columns
Key columns *
List of columns
Custom expression
CustomerID
Use tempdb
Interim table schema
Click refresh to load schema options
Refresh


date the current resource
Errors
Mapping
Optimize
Inspect
Data preview
Linked service
synapse-taksh-WorkspaceDefaultSto...
Test connection
Edit
New
Integration runtime *
AutoResolveIntegrationRuntime
Edit
SQL error rows
Error row handling
Stop on first error (default)
Transaction Commit
Single



Similarly, we build the data flow for all the other source files. The image provided below shows the pipeline's successful run after completing the data flows.



Gold Layer:

We are required to create some views on the data in sql and consequently generate reports on the data in Power BI as per the following requirements:

- **View 1** - for **Average Order Value (AOV)**
  - SUM(Amount) / COUNT(OrderID) per product, category, and location.
- **View 2** - for Segment customers based on total spend, purchase frequency, and loyalty tier (**`LoyaltyAccounts.TierLevel`**).
  - Example: "High-Value Customers" (Top 10% spenders), "One-Time Buyers," "Loyalty Champions."
- **View 3** - for Analyze **`DateTime`** to find peak days and times in-store vs. online.
- **View 4** - for Number of interactions and resolution success rates per agent (**`ResolutionStatus`**).

Let's use SQL Server Management Studio (SSMS) to create and execute these queries.

View-1:

```
CREATE VIEW AverageOrderValue AS
SELECT
    P.ProductID,
    P.Name AS ProductName,
    P.Category,
    S.StoreID,
    S.Location,
    SUM(OT.Amount)/COUNT(OT.OrderID) AS AverageOrderValue
FROM
    OnlineTransactions OT
JOIN
    Products P ON OT.ProductID = P.ProductID
JOIN
    Customers C ON OT.CustomerID = C.CustomerID
JOIN
    InStoreTransactions IST ON C.CustomerID = IST.CustomerID
JOIN
    Stores S ON IST.StoreID = S.StoreID
GROUP BY
    P.ProductID, P.Name, P.Category, S.StoreID, S.Location;
```

Output-1:

```
-- View the results
SELECT * FROM AverageOrderValue;
```

109 %

Results | Messages

	ProductID	ProductName	Category	StoreID	Location	AverageOrderValue
1	49	Already	Electronics	3	Vanessachester	78.530000
2	52	Final	Home Goods	3	Vanessachester	182.260000
3	69	Almost	Home Goods	4	Maytown	110.290000
4	25	Young	Books	7	Rebeccastad	78.110000
5	47	Contain	Toys	7	Rebeccastad	109.860000
6	43	Pick	Clothing	8	North Mariaburgh	164.040000
7	87	Star	Clothing	8	North Mariaburgh	131.080000
8	1	Consider	Electronics	10	Thompsonshire	49.780000
9	7	Happy	Toys	10	Thompsonshire	174.510000
10	23	Ahead	Home Goods	10	Thompsonshire	159.270000
11	7	Happy	Toys	11	East Justin	173.830000
12	8	Effect	Electronics	11	East Justin	166.460000
13	55	Sign	Books	11	East Justin	25.960000
14	13	Play	Home Goods	12	Jeffreyport	27.550000
15	78	Position	Clothing	12	Jeffreyport	162.580000
16	46	Citizen	Books	14	Josephfort	195.760000
17	53	Daughter	Electronics	14	Josephfort	87.740000
18	73	Political	Clothing	15	New Kimberly	103.300000
19	86	Alone	Books	15	New Kimberly	187.640000

View-2:

```
CREATE VIEW Customer_Segmentation AS
SELECT
    C.CustomerID,
    COALESCE(SUM(OT.Amount), 0) + COALESCE(SUM(IST.Amount), 0) AS TotalSpend,
    COUNT(DISTINCT OT.OrderID) + COUNT(DISTINCT IST.TransactionID) AS PurchaseFrequency,
    LA.TierLevel,
    CASE
        WHEN (COALESCE(SUM(OT.Amount), 0) + COALESCE(SUM(IST.Amount), 0)) >= 1000 THEN 'High-Value Customer'
        WHEN (COUNT(DISTINCT OT.OrderID) + COUNT(DISTINCT IST.TransactionID)) = 1 THEN 'One-Time Buyer'
        WHEN LA.TierLevel IN ('Gold', 'Platinum') THEN 'Loyalty Champion'
        ELSE 'Regular Customer'
    END AS Segment
FROM
    Customers C
JOIN
    OnlineTransactions OT ON C.CustomerID = OT.CustomerID
JOIN
    InStoreTransactions IST ON C.CustomerID = IST.CustomerID
JOIN
    LoyaltyAccounts LA ON C.CustomerID = LA.CustomerID
GROUP BY
    C.CustomerID, LA.TierLevel;
```

Output-2:

```
SELECT * FROM Customer_Segmentation;
```

109 %

Results | Messages

	CustomerID	TotalSpend	PurchaseFrequency	TierLevel	Segment
1	1	299.56	3	Bronze	Regular Customer
2	12	402.86	3	Bronze	Regular Customer
3	17	292.74	2	Bronze	Regular Customer
4	41	377.14	4	Bronze	Regular Customer
5	43	249.90	3	Bronze	Regular Customer
6	49	409.79	4	Bronze	Regular Customer
7	51	669.78	2	Bronze	Regular Customer
8	73	1727.86	5	Bronze	High-Value Customer
9	90	1142.60	3	Bronze	High-Value Customer
10	5	175.38	2	Gold	Loyalty Champion
11	13	1386.48	3	Gold	High-Value Customer
12	17	292.74	2	Gold	Loyalty Champion
13	20	897.08	4	Gold	Loyalty Champion
14	26	1825.12	5	Gold	High-Value Customer
15	31	463.71	4	Gold	Loyalty Champion

View-3:

```
CREATE VIEW Peak_Times AS
SELECT
    Channel,
    DATENAME(WEEKDAY, DateTime) AS DayOfWeek
    DATEPART(HOUR, DateTime) AS HourOfDay,
    COUNT(*) AS TransactionCount
FROM (
    SELECT
        DateTime,
        'Online' AS Channel
    FROM OnlineTransactions
    UNION ALL
    SELECT
        DateTime,
        'In-Store' AS Channel
    FROM InStoreTransactions
) AS Combined
GROUP BY
    Channel,
    DATENAME(WEEKDAY, DateTime),
    DATEPART(HOUR, DateTime);
```

Output-3:

```sql
-- View the results
SELECT * FROM Peak_Times;
```

109 %

Results | Messages

	Channel	DayOfWeek	HourOfDay	TransactionCount
1	In-Store	Friday	0	1
2	In-Store	Friday	1	1
3	In-Store	Friday	6	1
4	In-Store	Friday	8	1
5	In-Store	Friday	10	1
6	In-Store	Friday	13	2
7	In-Store	Friday	14	1
8	In-Store	Friday	21	3
9	In-Store	Friday	23	1
10	In-Store	Monday	6	1
11	In-Store	Monday	7	1
12	In-Store	Monday	8	1
13	In-Store	Monday	9	1
14	In-Store	Monday	12	1
15	In-Store	Monday	18	2

View-4:

```sql
CREATE VIEW Resolution_Stats AS
SELECT
    A.AgentID,
    A.Name AS AgentName,
    COUNT(CSI.InteractionID) AS TotalInteractions,
    SUM(CASE WHEN CSI.ResolutionStatus = 'Resolved' THEN 1 ELSE 0 END) AS ResolvedInteractions,
    ROUND(
        100.0 * SUM(CASE WHEN CSI.ResolutionStatus = 'Resolved' THEN 1 ELSE 0 END) / NULLIF(COUNT(CSI.InteractionID), 0),
        2
    ) AS ResolutionRatePercent
FROM
    Agents A
JOIN
    CustomerServiceInteractions CSI ON A.AgentID = CSI.AgentID
GROUP BY
    A.AgentID, A.Name;
```

Output-4:

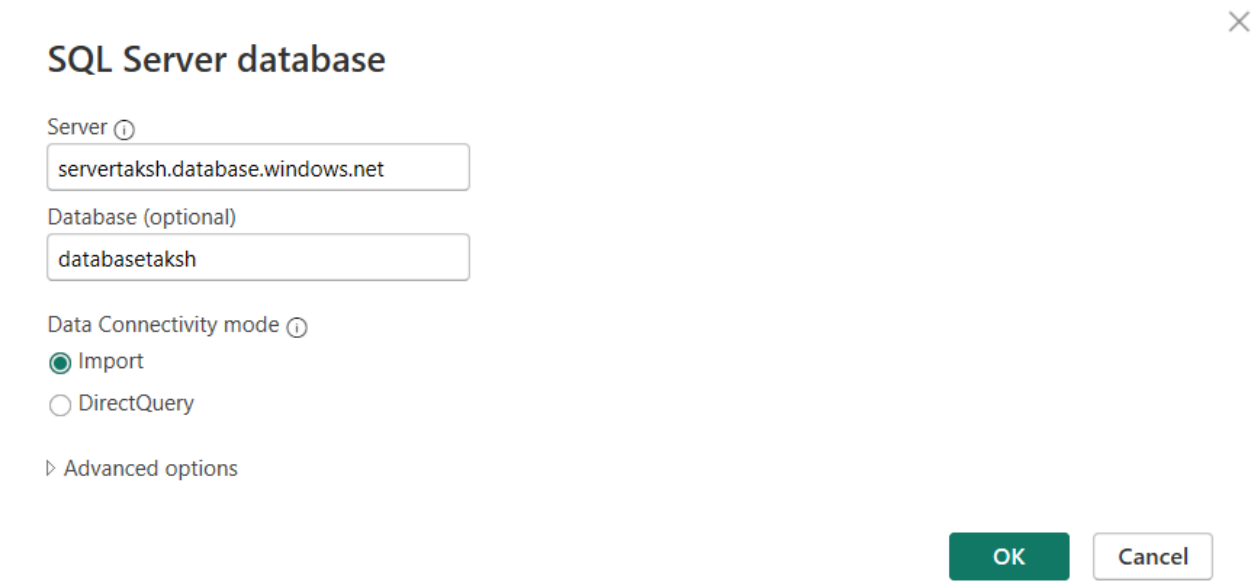
```
-- View the results
SELECT * FROM Resolution_Stats;
```

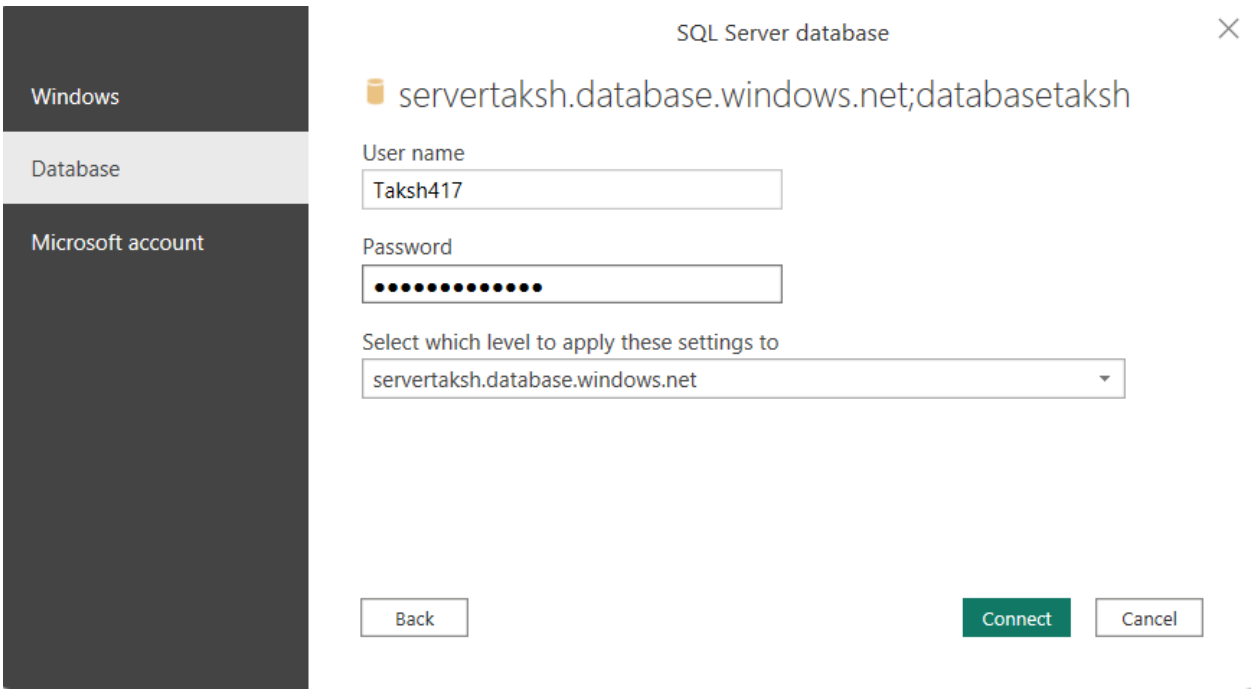
109 %

Results | Messages

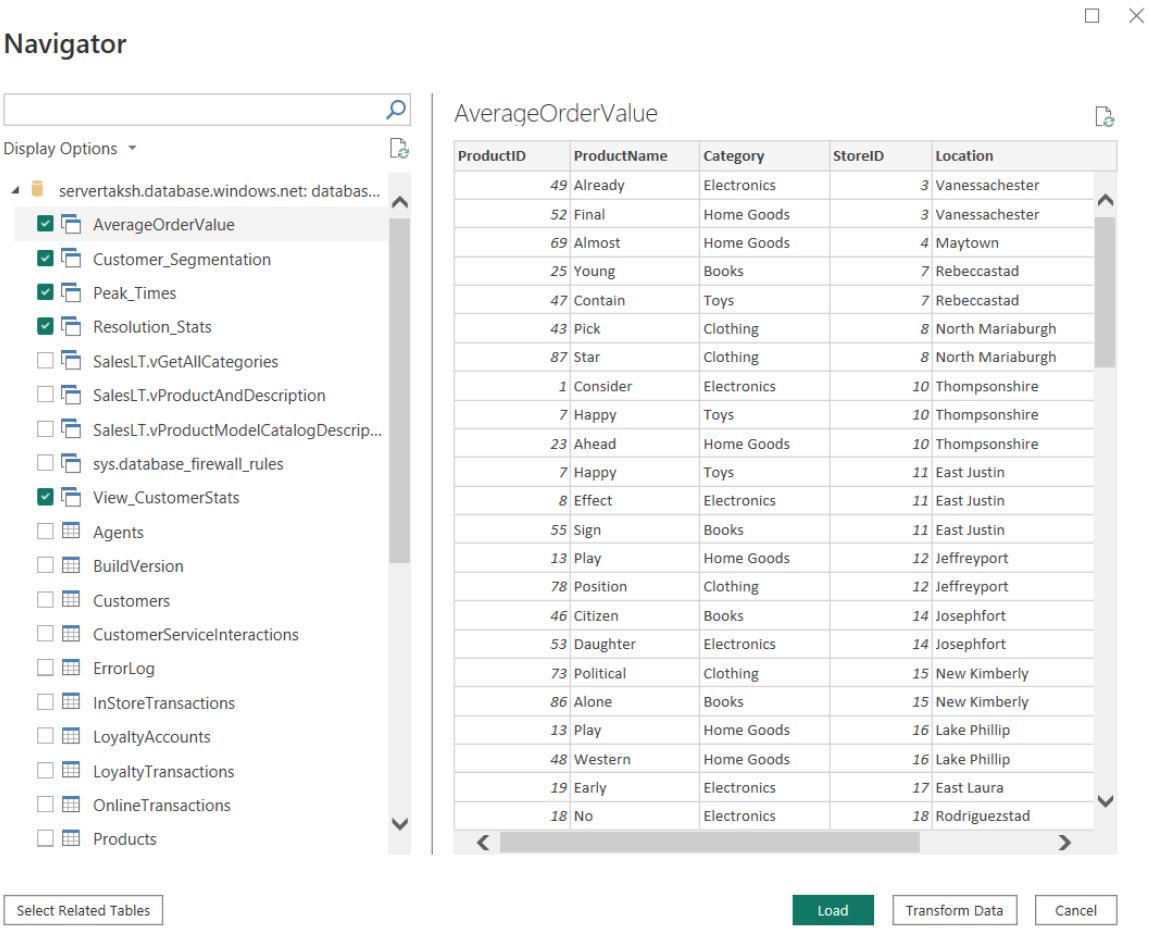
	AgentID	AgentName	TotalInteractions	ResolvedInteractions	ResolutionRatePercent
1	1	Jonathan Williams	1	0	0.000000000000
2	2	Terry Edwards	1	0	0.000000000000
3	4	Daryl Benjamin	2	0	0.000000000000
4	5	Matthew Long	1	0	0.000000000000
5	7	Elizabeth James	2	2	100.000000000000
6	8	Teresa Bennett	1	0	0.000000000000
7	9	Amber Ross	2	1	50.000000000000
8	10	Tonya Jones	1	0	0.000000000000
9	11	Curtis Mcbride	1	1	100.000000000000
10	13	Scott Flowers	1	0	0.000000000000
11	15	Kristen Crawford	1	1	100.000000000000
12	17	Brandon Jimenez	1	0	0.000000000000
13	18	Melissa White	2	0	0.000000000000
14	19	Eddie Pierce	1	0	0.000000000000
15	20	Julia Owens	1	0	0.000000000000
16	21	Cindy Gomez	3	2	66.670000000000

Finally, we connect our azure sql database to Power BI to generate reports. In Power BI, select azure sql database for connection and follow the images below to establish the connection.





Load only the views that we created into Power BI.



Images shown below are just a couple of reports generated on the views. To conclude, publish these reports to your fabric workspace.