Data Engineering Project 3:

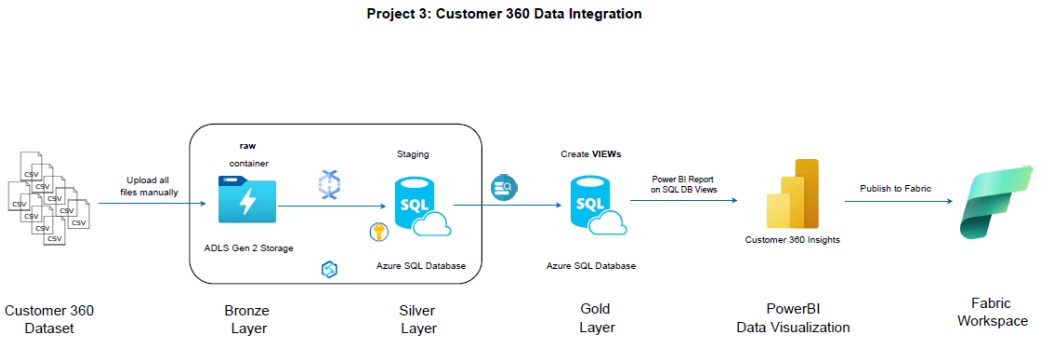
Customer 360 Data Integration

Table of Contents

Objective:

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.

Architecture Diagram:



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

Step 1: Data Ingestion

Use Azure Synapse Analytics pipelines to ingest data from multiple sources (online, in-store, customer service, loyalty programs) into the raw container in ADLS.

Step 1.1: Create a folder called **bronze** in ADLS Gen 2 storage account and upload all the csv files which you have downloaded form source dataset URL.

Authentication method: Access key (Switch to Microsoft Entra user account)
Location: project3 / bronze

Search blobs by prefix (case-sensitive) Show deleted objects

Name	Modified	Access tier	Archive status	Blob type	Size
Agents.csv	4/28/2025, 12:15:27 PM	Hot (Inferred)		Block blob	3.82 KiB
Customers.csv	4/28/2025, 12:15:27 PM	Hot (Inferred)		Block blob	8.72 KiB
CustomerServiceInteractions.csv	4/28/2025, 12:15:27 PM	Hot (Inferred)		Block blob	5 KiB
InStoreTransactions.csv	4/28/2025, 12:15:27 PM	Hot (Inferred)		Block blob	4.62 KiB
LoyaltyAccounts.csv	4/28/2025, 12:15:27 PM	Hot (Inferred)		Block blob	2.93 KiB
LoyaltyTransactions.csv	4/28/2025, 12:15:27 PM	Hot (Inferred)		Block blob	3.67 KiB
OnlineTransactions.csv	4/28/2025, 12:15:27 PM	Hot (Inferred)		Block blob	5.43 KiB
Products.csv	4/28/2025, 12:15:27 PM	Hot (Inferred)		Block blob	2.57 KiB
Stores.csv	4/28/2025, 12:15:27 PM	Hot (Inferred)		Block blob	4.84 KiB

Step 1.2 Open SSMS and Connect the Azure SQL Database

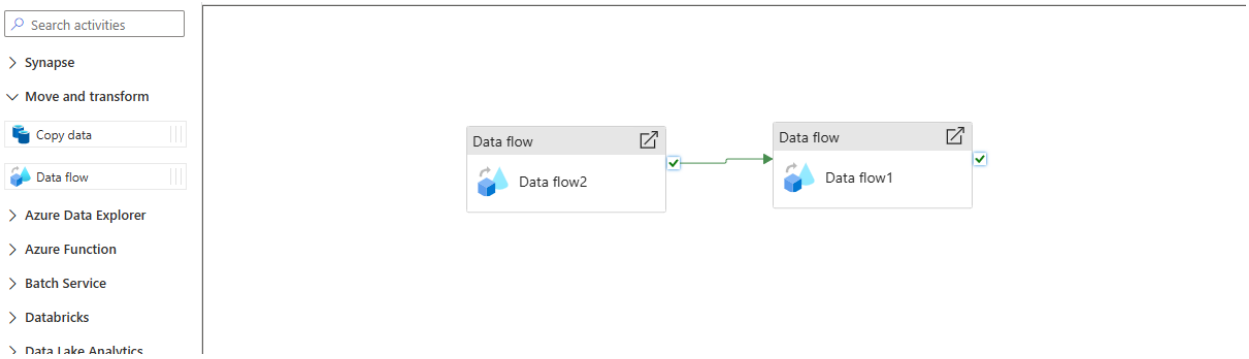
Create these table in this sequence (as they have constraints)

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

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

----Dependent Table------------
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)
);

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

----Dependent Table------------
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)
);
```

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

----Dependent Table------------
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)
);

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

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

Step 1.3: Open Synapse Workspace, Create a new pipeline

Drag and drop two dataflow activities in canvas area.



Click on the first one, and Settings -> Dataflow -> New

Blank dataflow design canvas will be opened

Here, we will design transformation step from bronze to silver (staging) layer

Step 1.4 Overview of the customer records transformation steps:

Let's do step by step.

Select the source

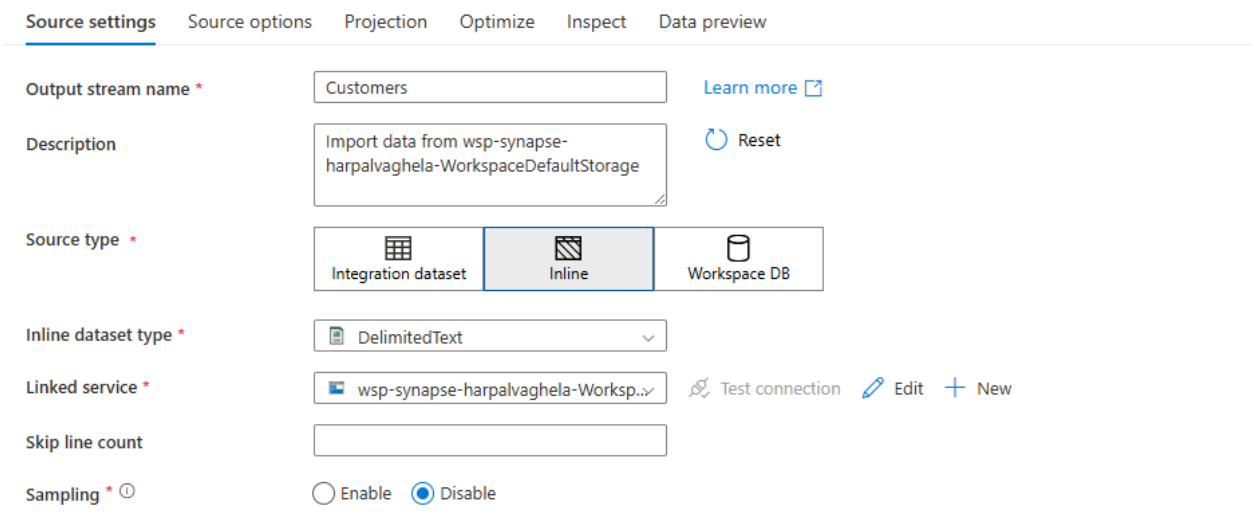
Source Settings Tab:

Write output stream: Customers

Source type: Inline

Inline dataset type: Delimited Text

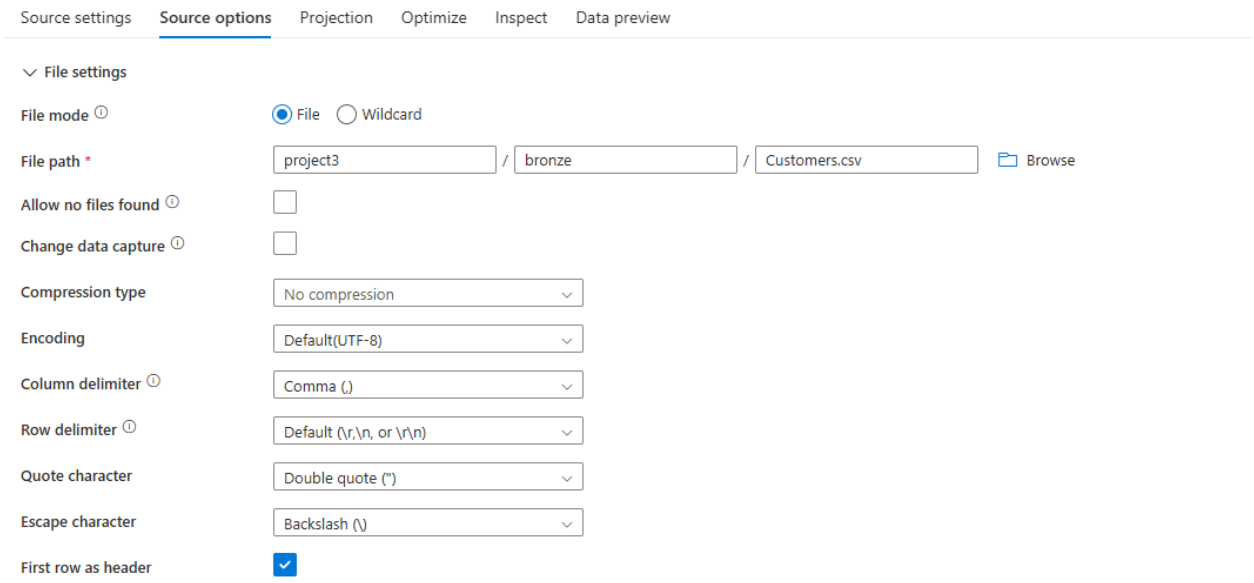
Linked Service: ADLS Gen 2



Source Options Tab:

File path: browse and select the file from storage account as shown below

Select first row as header



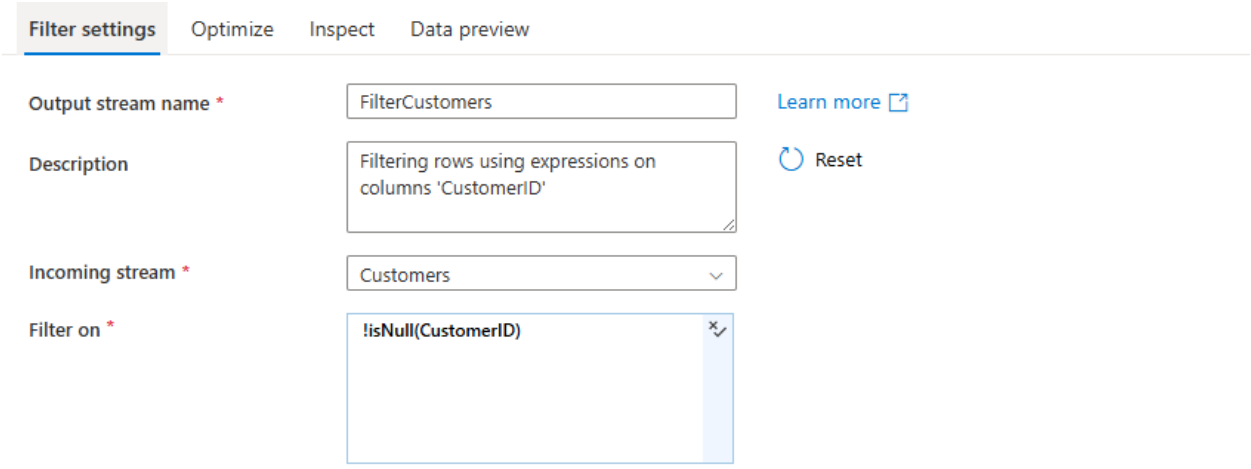
Projection Tab:

Click on Import Schema

<u>Filter Transformation</u>

Click on plus icon on bottom of source transformation and add filter transformation

Filter Settings Tab:



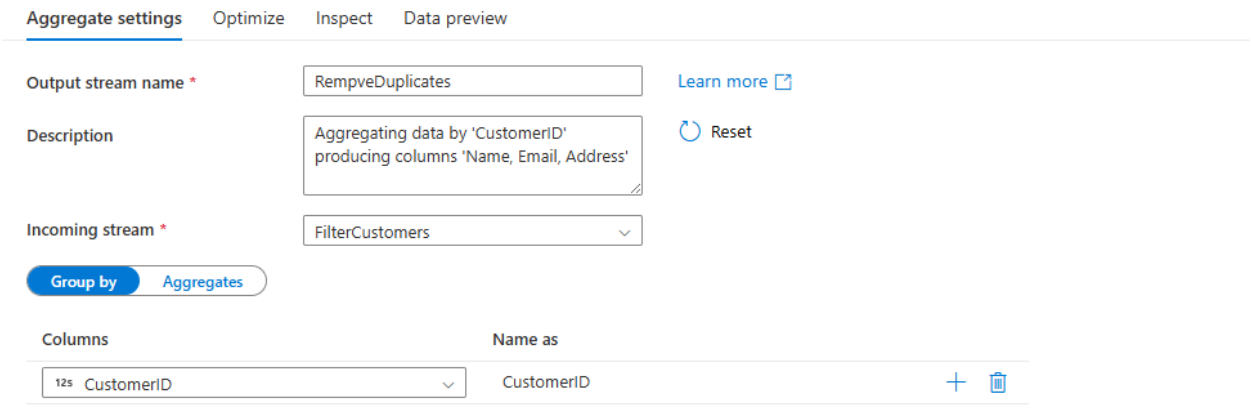
Expression used in filter on field is: !isNull(CustomerID)

Aggregate Transformation (To Remove Duplicates)

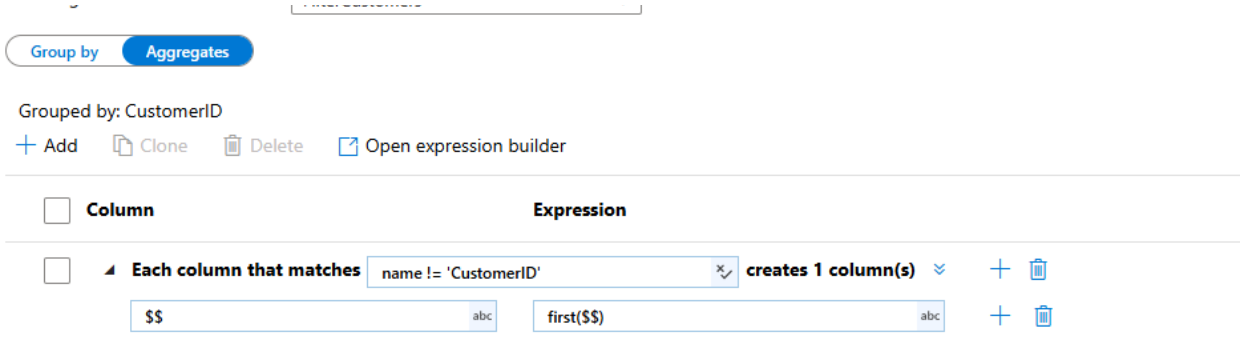
Click on plus icon on bottom of filter transformation and add aggregate transformation

Aggregate settings tab

Select ID column (here, CustomerID) in Group by tab as shown below:



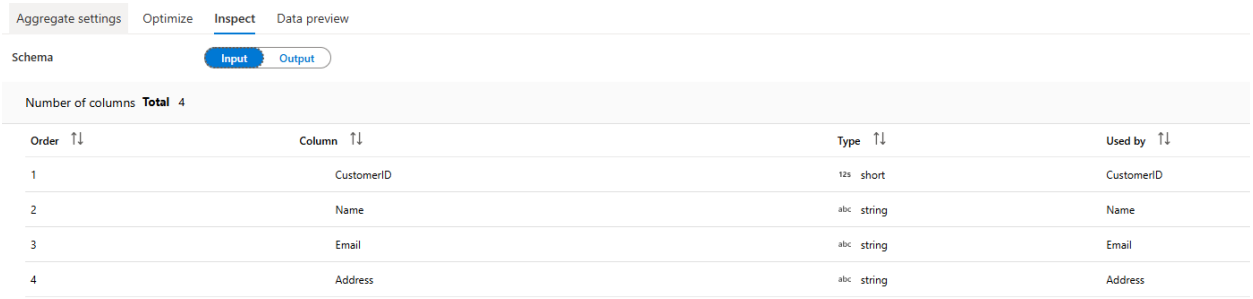
Click on Aggregates tab now and click on plus icon and take column pattern and delete the default field.



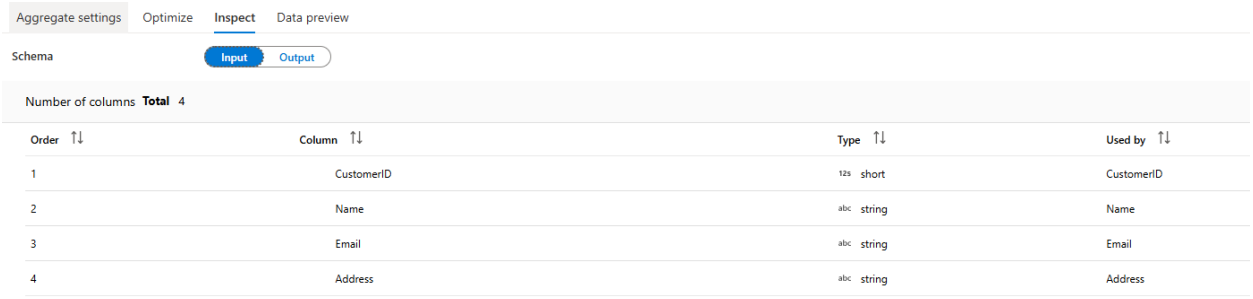
Write this expression in

Each column that matches: name != ‘CustomerID’

Check the data in Inspect -> Input



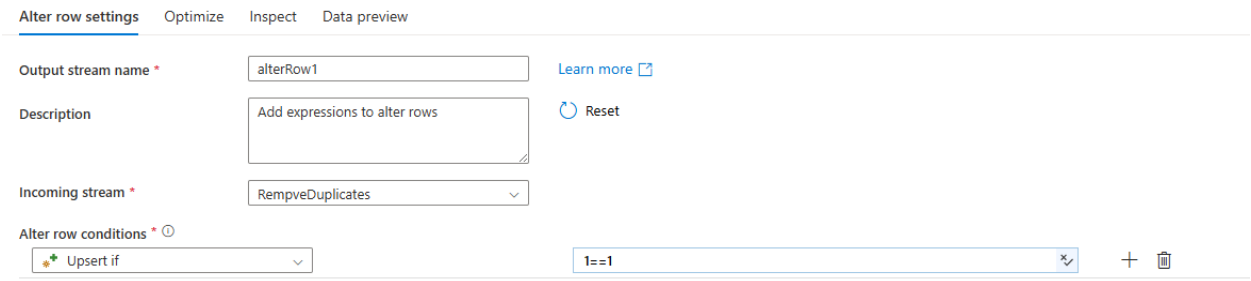
Inspect -> Output



Alter Row Transformation

Click on plus icon on bottom of aggregate transformation and alter row sink transformation

Alter row conditions: Upsert if
Expression: 1==1



Sink Transformation:

Click on plus icon on bottom of aggregate transformation and add sink transformation

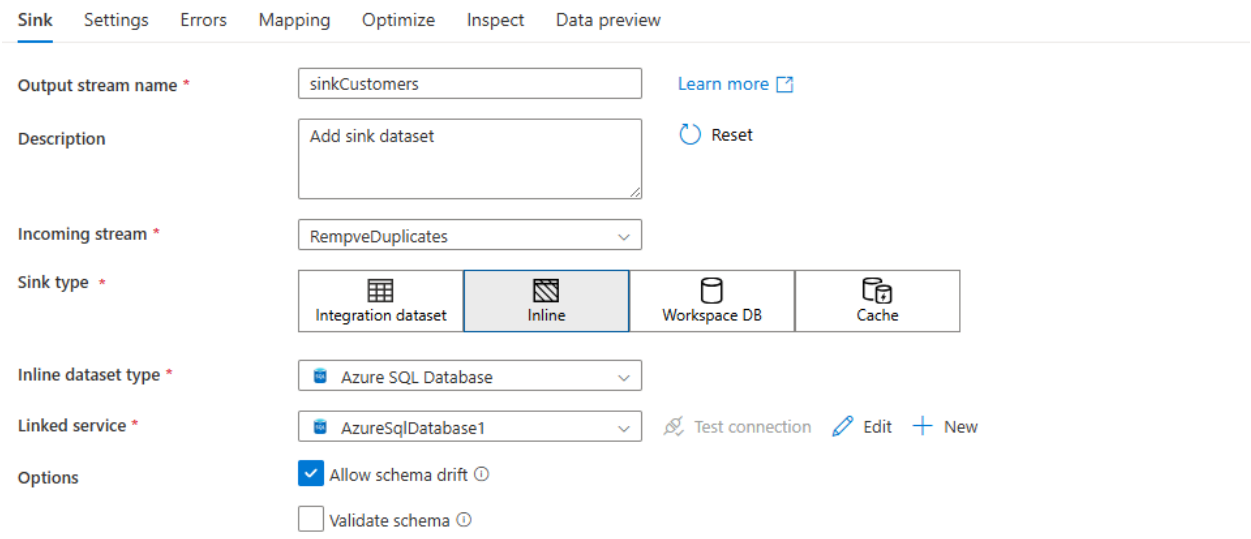
Sink tab:

Output stream name: sinkCustomers

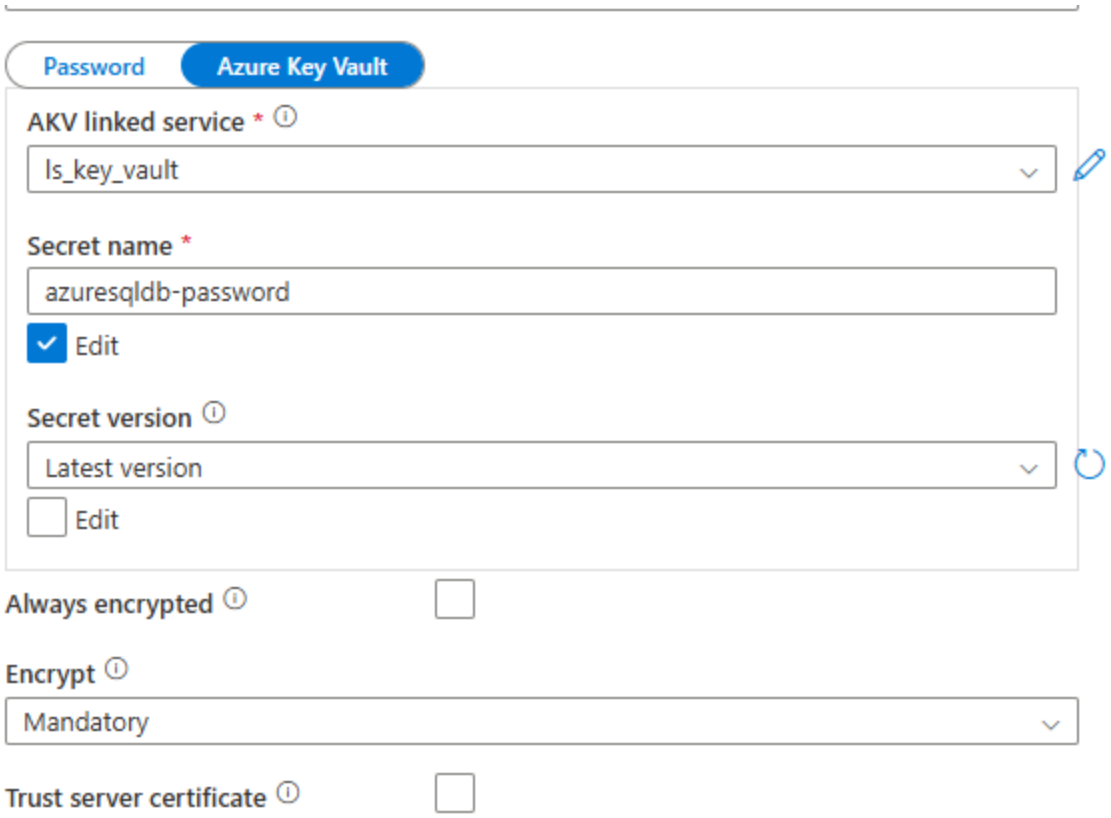
Sink type: Inline

Inline dataset type: Azure SQL Database

Linked service: Azure SQL Database



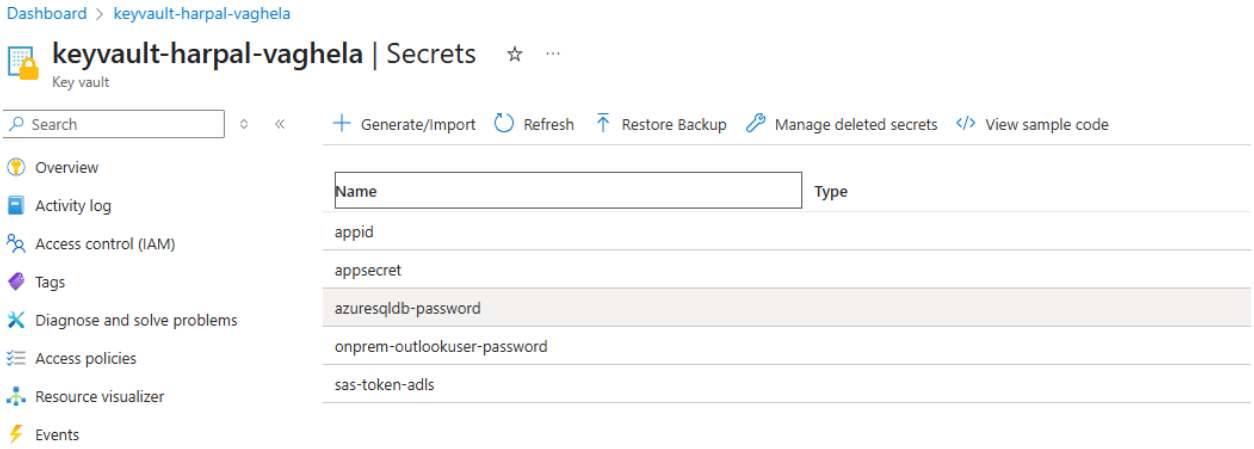
While Creating Linked Service, use Azure Key vault option to store Azure SQL server password in to key vault.



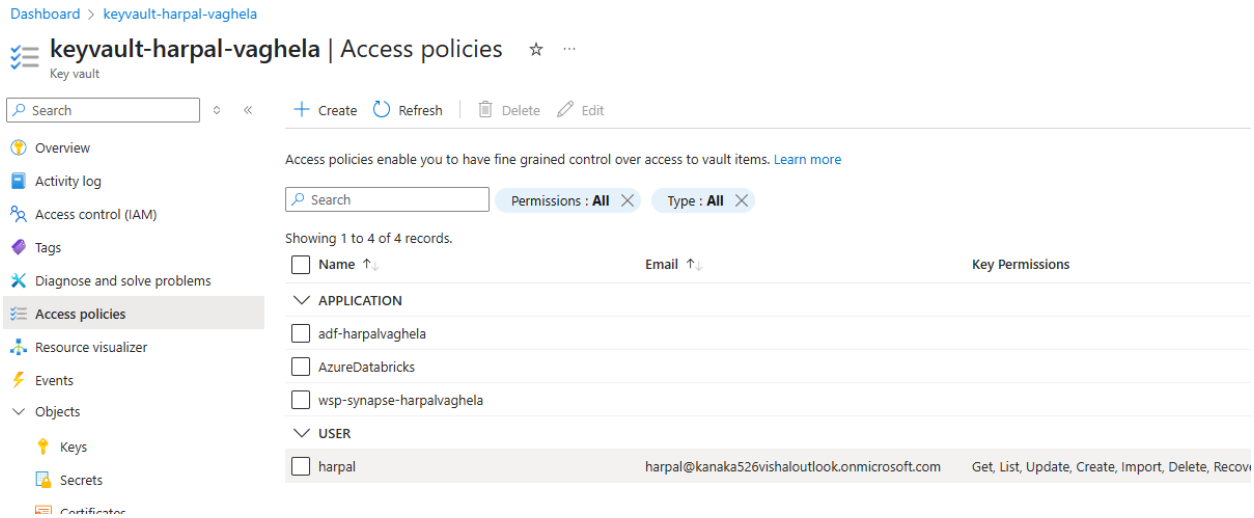
Step to create new azure sql password into key vault:

Open **Key Vault** and click on **Generate/Import**

Write Secret name and Secret value (password of Azure SQL Server) and click on Create.



Also, in Key Vault Access Policies give permission (Get, List for Synapse Workspace to access key vault secrets)



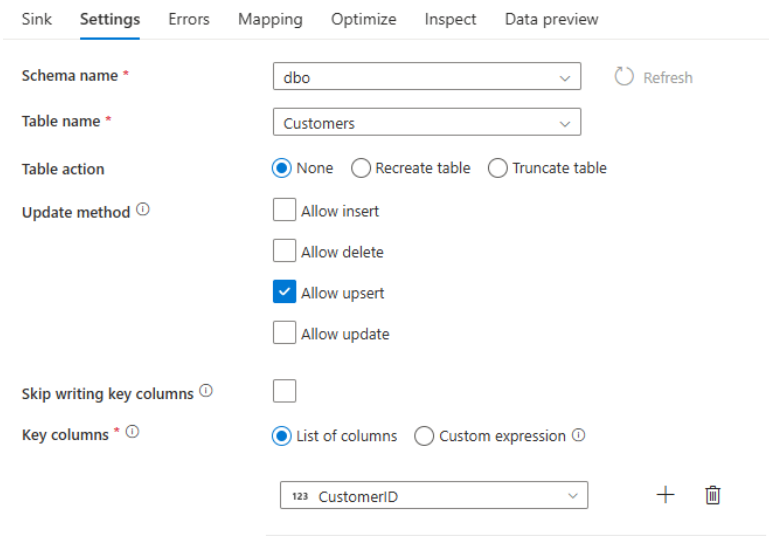
Settings tab:

Schema Name : dbo (select from a dropdown list)

Table Name: Customers (select from a dropdown list)

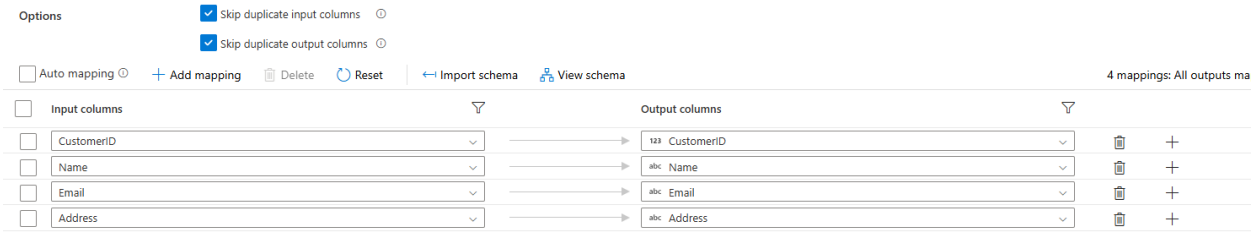
Update Method: Allow Upsert

Key Columns: CustomerID



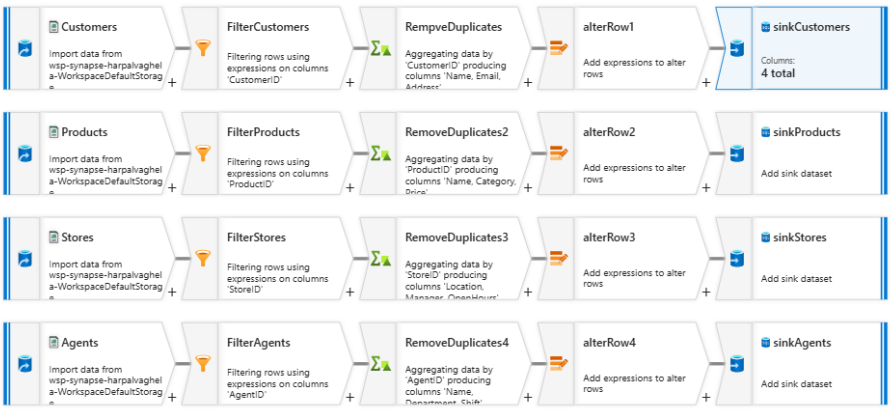
Mapping tab:

Click on Import Schema and map the columns as shown below:



Similarly, we will do transformations steps or Products, Stores, Agents data files.

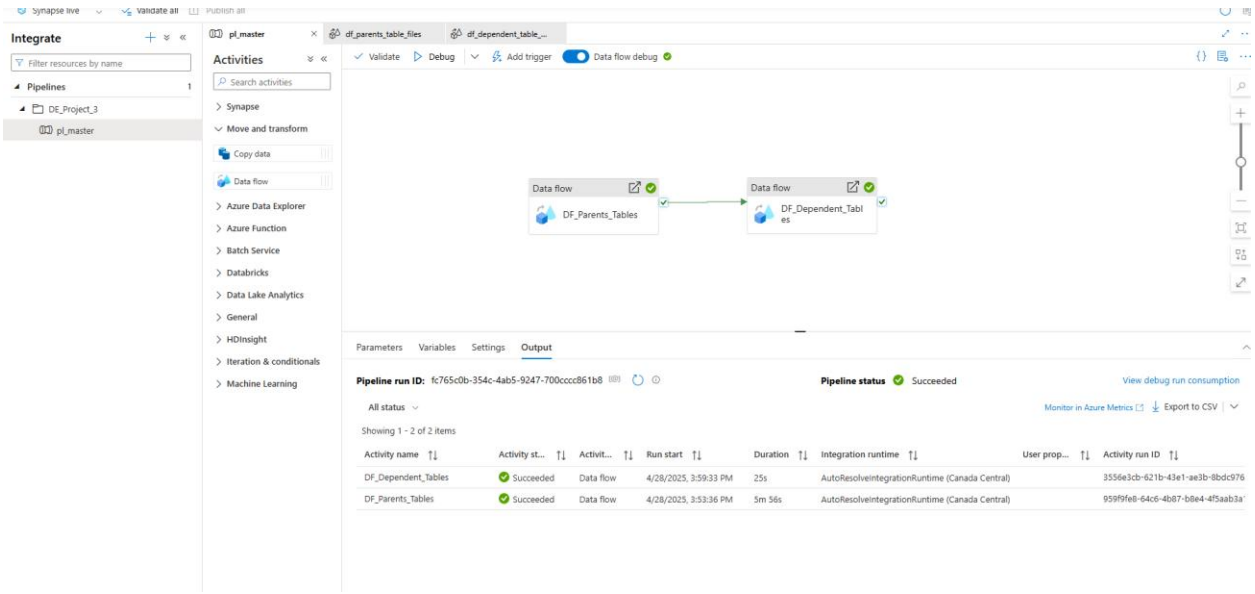
Here is an final design in dataflow1:



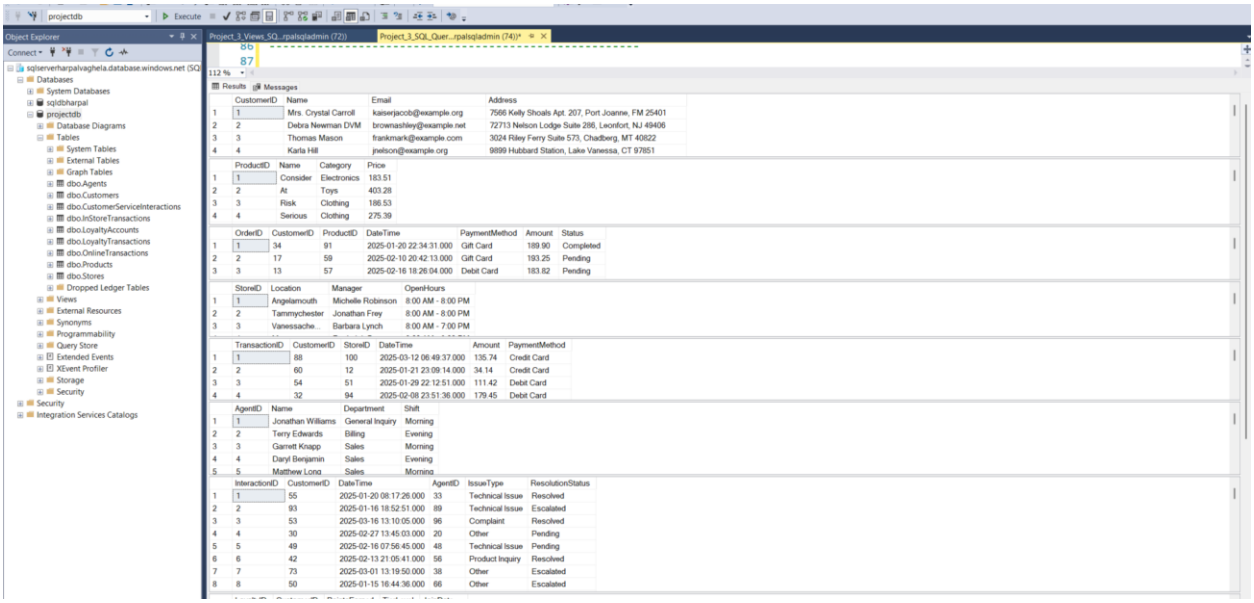
Step 1.5 Design of Dataflow2:

Similarly, we have to design for OnlineTransaction, InStoreTransactions, CustomerService, LoyaltyAccounts, LoyaltyTransactions

Step 1.6 Run the Pipeline



Step 2: Check the data into SSMS



We have successfully loaded the cleaned data into azure SQL database.

Step 3: Create Views for Gold Layer

View 1 - Average Order Value (AOV) (AOV = Total Sales Amount / Total Number of Orders)

SUM(Amount) / COUNT(OrderID) per product, category, and location.

- We listed each product along with its category and the store location.
- We calculated the average amount customers spend each time they order a product.
- We connected each online sale to its product to know what was sold.
- We also tried to link customers to store locations, even for online sales.
- Finally, we grouped the results by product, category, and store to show clear average spending for each.

```sql
---1) Average Order Value (AOV)

CREATE VIEW View_AverageOrderValue AS
SELECT
    P.ProductID,
    P.Name AS ProductName,
    P.Category,
    S.Location AS StoreLocation,
    CAST(SUM(OT.Amount) * 1.0 / COUNT(OT.OrderID) AS DECIMAL(10,2)) AS AverageOrderValue
FROM
    OnlineTransactions OT
JOIN
    Products P ON OT.ProductID = P.ProductID
LEFT JOIN
    InStoreTransactions IST ON OT.CustomerID = IST.CustomerID
LEFT JOIN
    Stores S ON IST.StoreID = S.StoreID
GROUP BY
    P.ProductID, P.Name, P.Category, S.Location;

-----------------------
Select * from View_AverageOrderValue;
```

- This helps business users and analysts easily see how much customers typically spend per order, broken down by Product, Category, and Store Location.

Output : Select * from View_AverageOrderValue;

Results Messages

	ProductID	ProductName	Category	StoreLocation	AverageOrderValue
37	19	Early	Electronics	East Laura	175.37
38	62	Describe	Clothing	Ellisstad	44.52
39	37	Wish	Books	Gabrielafurt	136.61
40	57	Truth	Books	Gabrielafurt	199.26
41	44	Movement	Toys	Haroldmouth	110.47
42	45	Nor	Toys	Haroldmouth	14.72
43	69	Almost	Home Goods	Haroldmouth	110.29
44	88	Much	Home Goods	Haroldmouth	97.94
45	7	Happy	Toys	Haysville	174.51
46	23	Ahead	Home Goods	Haysville	159.27
47	44	Movement	Toys	Haysville	91.74
48	77	Past	Books	Haysville	145.52
49	7	Happy	Toys	Hodgesland	173.83
50	55	Sign	Books	Hodgesland	25.96
51	8	Effect	Electronics	Hollyburgh	166.46
52	6	Area	Books	Jasonmouth	143.57
53	13	Play	Home Goods	Jeffreyport	27.55
54	78	Position	Clothing	Jeffreyport	162.58
55	48	Western	Home Goods	Jermaineview	71.25
56	83	Night	Books	Jermaineview	16.63
57	46	Citizen	Books	Josephfort	195.76
58	53	Daughter	Electronics	Josephfort	87.74

View 2 - 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."

- We looked at each customer's total spending and how often they made purchases.
- We combined both online and in-store purchases to get a full view of customer activity.
- We checked if the customer belongs to a loyalty program and noted their tier level.
- We identified the top 10% spenders as "High-Value Customers."
- Customers who only purchased once are marked as "One-Time Buyers."

- Customers with high loyalty status like Gold or Platinum are called "Loyalty Champions."
- Everyone else is grouped as a "Regular Customer" for simpler analysis.

We can analyse these details from this view2:

- Total money spent (online + in-store combined)
- Number of purchases made.
- Their loyalty program tier (if they have one)
- A customer segment label:
  (High-Value Customer, One-Time Buyer, Loyalty Champion, or Regular Customer).
- Customers who are in the top 10% by total spending (high-priority for special offers or retention programs).
- Customers who purchased only once, it could be targeted for re-engagement campaigns.
- Customers who are highly engaged in the loyalty program (Gold/Platinum tier).
- Tailor promotions, discounts, and communications specifically for each group.

```sql
---============================================================================
---2) Customer Segmentation based on Spend, Frequency, and Loyalty Tier

CREATE VIEW View_CustomerSegmentation AS
WITH CustomerSpend AS (
    SELECT
        C.CustomerID,
        C.Name AS CustomerName,
        SUM(COALESCE(OT.Amount, 0) + COALESCE(IST.Amount, 0)) AS TotalSpend,
        COUNT(DISTINCT OT.OrderID) + COUNT(DISTINCT IST.TransactionID) AS PurchaseFrequency,
        LA.TierLevel
    FROM
        Customers C
    LEFT JOIN
        OnlineTransactions OT ON C.CustomerID = OT.CustomerID
    LEFT JOIN
        InStoreTransactions IST ON C.CustomerID = IST.CustomerID
    LEFT JOIN
        LoyaltyAccounts LA ON C.CustomerID = LA.CustomerID
    GROUP BY
        C.CustomerID, C.Name, LA.TierLevel
),
Percentile AS (
    SELECT
        PERCENTILE_CONT(0.9) WITHIN GROUP (ORDER BY TotalSpend)
        OVER () AS Spend90Percentile
    FROM CustomerSpend
)
SELECT
    CS.CustomerID,
    CS.CustomerName,
    CS.TotalSpend,
    CS.PurchaseFrequency,
    CS.TierLevel,
    CASE
        WHEN CS.TotalSpend >= (SELECT TOP 1 Spend90Percentile FROM Percentile) THEN 'High-Value Customer'
        WHEN CS.PurchaseFrequency = 1 THEN 'One-Time Buyer'
        WHEN CS.TierLevel IN ('Gold', 'Platinum') THEN 'Loyalty Champion'
        ELSE 'Regular Customer'
    END AS CustomerSegment
FROM
    CustomerSpend CS;
```

Output: Select * from View_CustomerSegmentation;

Results Messages

	CustomerID	CustomerName	TotalSpend	PurchaseFrequency	TierLevel	CustomerSegment
1	1	Mrs. Crystal Carroll	299.56	3	Bronze	Regular Customer
2	1	Mrs. Crystal Carroll	299.56	3	Platinum	Loyalty Champion
3	1	Mrs. Crystal Carroll	299.56	3	Silver	Regular Customer
4	2	Debra Newman DVM	47.20	1	NULL	One-Time Buyer
5	3	Thomas Mason	0.00	0	Gold	Loyalty Champion
6	3	Thomas Mason	0.00	0	Silver	Regular Customer
7	4	Karla Hill	0.00	0	NULL	Regular Customer
8	5	Jeffrey Underwood	175.38	2	Gold	Loyalty Champion
9	5	Jeffrey Underwood	175.38	2	Silver	Regular Customer
10	6	Justin Lowe	94.31	1	Gold	One-Time Buyer
11	7	Jason Daniels	58.81	1	NULL	One-Time Buyer
12	8	Jennifer Hill	90.37	1	Gold	One-Time Buyer
13	9	Stephanie Richardson	1058.88	3	Platinum	High-Value Customer
14	10	Jo Zimmerman	0.00	0	Bronze	Regular Customer
15	10	Jo Zimmerman	0.00	0	Gold	Loyalty Champion
16	11	Andrew Williams	534.29	3	Platinum	Loyalty Champion
17	12	Robin Odom	402.86	3	Bronze	Regular Customer
18	13	Gina Anderson	1386.48	3	Gold	High-Value Customer
19	14	Devin Smith	50.29	1	Gold	One-Time Buyer
20	14	Devin Smith	50.29	1	Platinum	One-Time Buyer
21	15	Sarah Roberts	0.00	0	NULL	Regular Customer
22	16	Andre Wright	160.07	2	Silver	Regular Customer
23	17	Karen Roberts	292.74	2	Bronze	Regular Customer
24	17	Karen Roberts	292.74	2	Gold	Loyalty Champion
25	18	Alisha Phillips	1191.46	4	Silver	High-Value Customer
26	19	Brandon Armstrong	266.40	2	NULL	Regular Customer
27	20	Luis Jones	897.08	4	Gold	Loyalty Champion
28	20	Luis Jones	897.08	4	Silver	Regular Customer
29	21	Anna Williams	239.86	2	Silver	Regular Customer
30	22	Evelyn Burton	214.25	2	Silver	Regular Customer

View 3 - Analyze DateTime to find peak days and times in-store vs. online.

- We combined online and in-store transaction data into a single view.
- We extracted the day of the week (like Monday, Tuesday) and the hour of the day from each transaction.
- We counted how many transactions happened in each day-hour slot for both channels.
- We grouped the results by day, hour, and channel (Online or In-Store).
- This helps identify when customers shop the most, both by day and time.

```sql
---3) Peak Days and Times Analysis (Online vs In-Store)

CREATE VIEW View_PeakTimes AS
SELECT
    'Online' AS Channel,
    DATENAME(WEEKDAY, OT.DateTime) AS DayOfWeek,
    DATEPART(HOUR, OT.DateTime) AS HourOfDay,
    COUNT(*) AS TransactionCount
FROM
    OnlineTransactions OT
GROUP BY
    DATENAME(WEEKDAY, OT.DateTime),
    DATEPART(HOUR, OT.DateTime)

UNION ALL

SELECT
    'InStore' AS Channel,
    DATENAME(WEEKDAY, IST.DateTime) AS DayOfWeek,
    DATEPART(HOUR, IST.DateTime) AS HourOfDay,
    COUNT(*) AS TransactionCount
FROM
    InStoreTransactions IST
GROUP BY
    DATENAME(WEEKDAY, IST.DateTime),
    DATEPART(HOUR, IST.DateTime);

-----------------
Select * from View_PeakTimes
```

We can analyse these details from this view3:

- Find out which weekdays (Monday, Friday, Saturday, etc.) are busiest for both online and in-store sales.
- See which hours (morning, afternoon, evening) have the highest number of purchases.
- Understand if customers prefer shopping online during specific times and visiting stores at different times.
- For in-store, schedule more employees during peak hours.
- For online, run targeted ads or flash sales during high-traffic periods.
- Reduce wait times, improve service quality, and plan better based on when customers are most active.

Output: Select * from View_PeakTimes

Results Messages

	Channel	DayOfWeek	HourOfDay	TransactionCount
1	Online	Monday	0	1
2	Online	Sunday	0	1
3	Online	Tuesday	0	1
4	Online	Friday	1	1
5	Online	Saturday	1	1
6	Online	Tuesday	1	1
7	Online	Monday	2	2
8	Online	Saturday	2	1
9	Online	Wednesday	2	1
10	Online	Saturday	3	2
11	Online	Sunday	3	1
12	Online	Tuesday	3	1
13	Online	Friday	4	1
14	Online	Thursday	4	2
15	Online	Tuesday	4	1
16	Online	Wednesday	4	2
17	Online	Friday	5	3
18	Online	Sunday	5	1
19	Online	Tuesday	5	1
20	Online	Wednesday	5	2
21	Online	Monday	6	1
22	Online	Sunday	6	1
23	Online	Thursday	6	1
24	Online	Wednesday	6	1
25	Online	Friday	7	1
26	Online	Tuesday	7	2
27	Online	Monday	8	1
28	Online	Saturday	8	2
29	Online	Wednesday	8	1
30	Online	Friday	9	1

View 4 - Number of interactions and resolution success rates per agent (ResolutionStatus)

- We listed each agent along with the total number of customer interactions they handled.
- We counted how many of those interactions were successfully resolved.
- We calculated each agent's resolution success rate as a percentage (with 2 decimal places).
- We used a left join to make sure even agents with zero interactions are included.
- This view helps measure and compare agent performance based on how effectively they resolve customer issues.

```
105
106 CREATE VIEW View_AgentPerformance AS
107 SELECT
108     A.AgentID,
109     A.Name AS AgentName,
110     COUNT(CSI.InteractionID) AS TotalInteractions,
111     SUM(CASE WHEN CSI.ResolutionStatus = 'Resolved' THEN 1 ELSE 0 END) AS ResolvedInteractions,
112     CASE
113         WHEN COUNT(CSI.InteractionID) = 0 THEN NULL
114          ELSE CAST((SUM(CASE WHEN CSI.ResolutionStatus = 'Resolved' THEN 1 ELSE 0 END) * 1.0 / COUNT(CSI.InteractionID)) * 100 AS DECIMAL(10,2))
115     END AS ResolutionSuccessRate
116 FROM
117     Agents A
118 LEFT JOIN
119     CustomerServiceInteractions CSI ON A.AgentID = CSI.AgentID
120 GROUP BY
121     A.AgentID, A.Name;
122
123 ----------------------
124 Select * from View_AgentPerformance;
```

We can analyse these details from this view4:

- Identify agents who have the highest success rates in resolving customer issues.
- Spot agents who have a low resolution success rate and might need additional coaching.
- See which agents are handling more customer interactions — this helps in balancing team workloads.
- Track how well the overall support team is resolving issues, and how it improves over time.
- Set performance goals or KPIs (Key Performance Indicators) based on actual agent success rates.

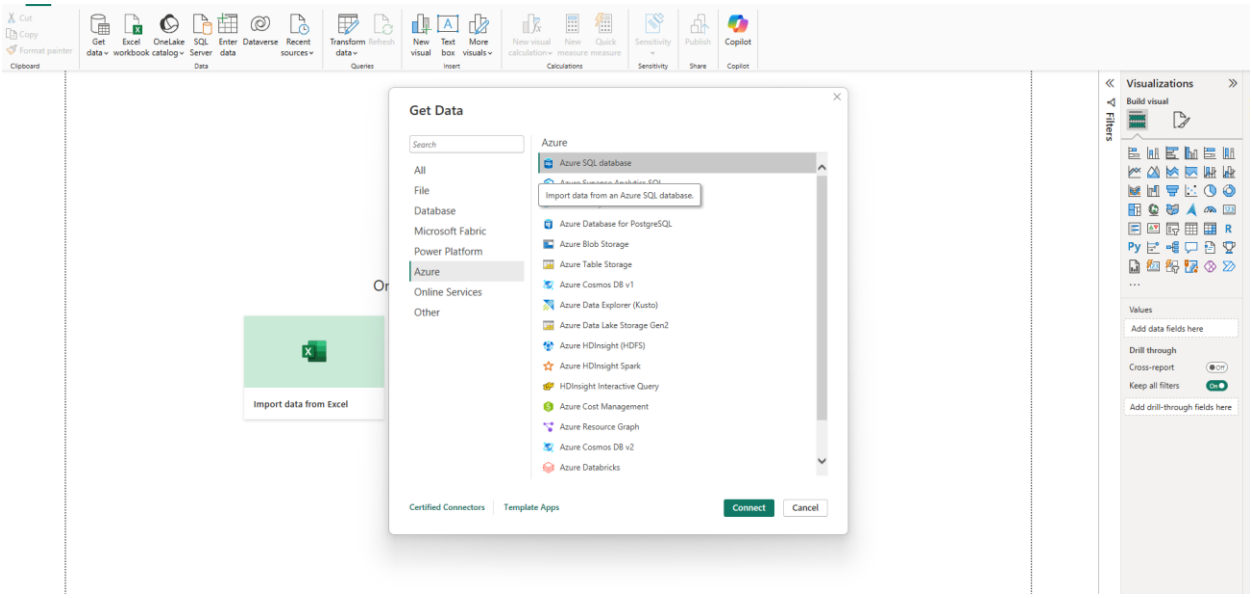
Output: Select * from View_AgentPerformance;

Results Messages

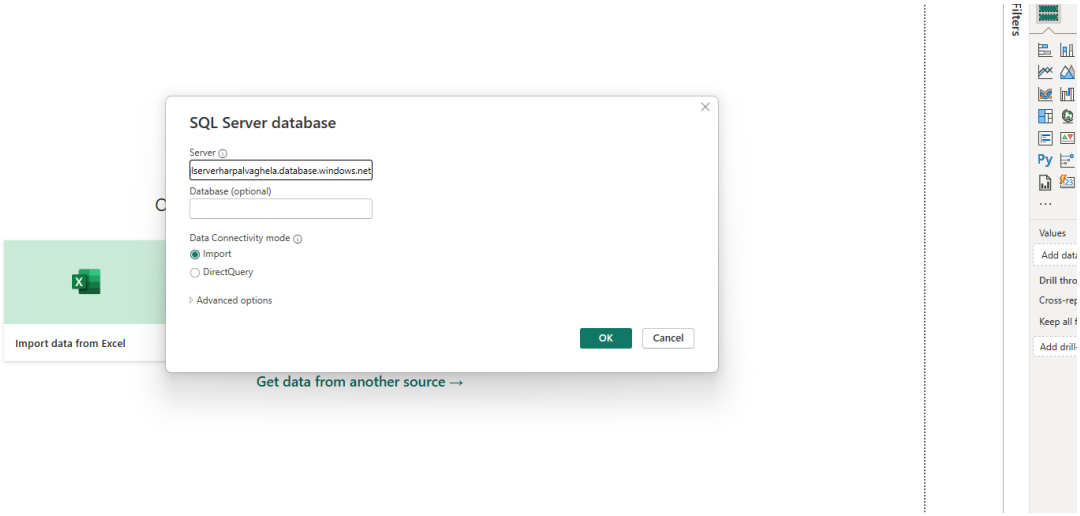
	AgentID	AgentName	TotalInteractions	ResolvedInteractions	ResolutionSuccessRate
1	1	Jonathan Williams	1	0	0.00
2	2	Terry Edwards	1	0	0.00
3	3	Garrett Knapp	0	0	NULL
4	4	Daryl Benjamin	2	0	0.00
5	5	Matthew Long	1	0	0.00
6	6	Patricia Rhodes	0	0	NULL
7	7	Elizabeth James	2	2	100.00
8	8	Teresa Bennett	1	0	0.00
9	9	Amber Ross	2	1	50.00
10	10	Tonya Jones	1	0	0.00
11	11	Curtis Mcbride	1	1	100.00
12	12	Justin Michael	0	0	NULL
13	13	Scott Flowers	1	0	0.00
14	14	Michael Jackson	0	0	NULL
15	15	Kristen Crawford	1	1	100.00
16	16	Jo Meyers	0	0	NULL
17	17	Brandon Jimenez	1	0	0.00
18	18	Melissa White	2	0	0.00
19	19	Eddie Pierce	1	0	0.00
20	20	Julia Owens	1	0	0.00
21	21	Cindy Gomez	3	2	66.67
22	22	Greg Smith	0	0	NULL
23	23	Shane Hernandez	0	0	NULL
24	24	Gregory Chavez	0	0	NULL
25	25	Elizabeth Casey	0	0	NULL
26	26	Doris Knight	0	0	NULL
27	27	Shawn Gill	2	1	50.00
28	28	Ricky Davenport	1	0	0.00
29	29	Jessica Mora	2	1	50.00
30	30	Vanessa Henson	0	0	NULL

Step 4: Power BI Dashboard/Report

Select Azure -> Azure SQL Database and click on connect



Enter Server URL from Azure Portal (Azure SQL Server-> Overview tab)



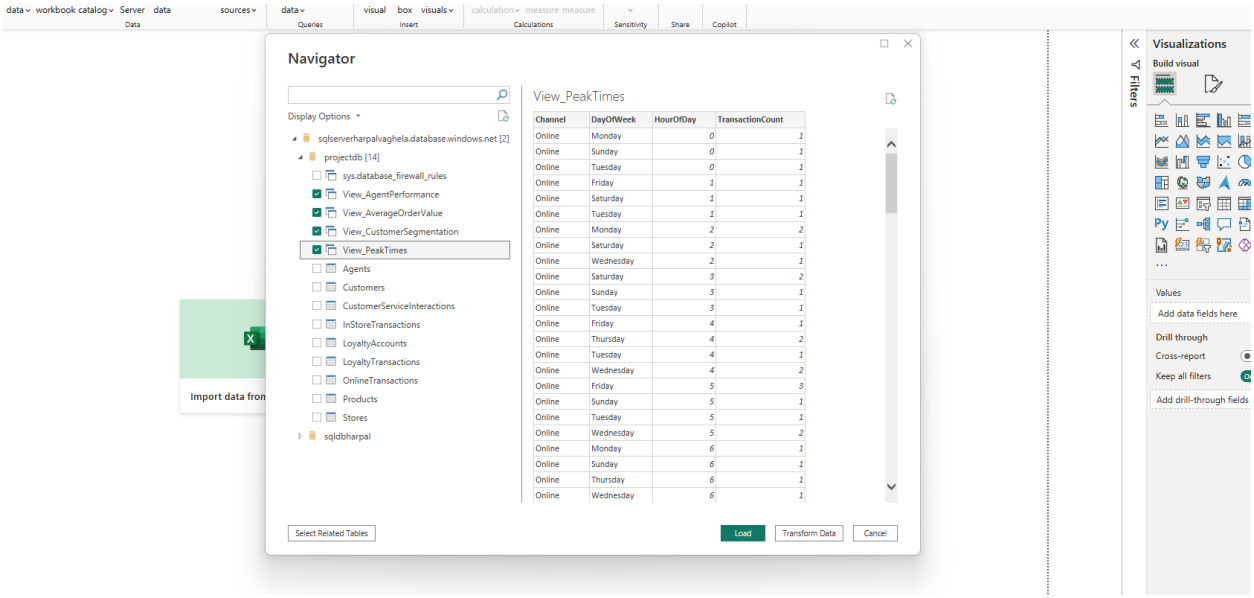
Data Connectivity Mode

- Import Mode: Power BI copies the full data from your source (SQL Database) into Power BI's internal storage (in-memory).
  - Fastest performance for visuals because data is preloaded.
  - Best for small to medium datasets.
  - You need to refresh manually or schedule refreshes when source data changes.
  - Report is disconnected from live source after load.
  - Good when performance is critical and data changes aren't real-time.

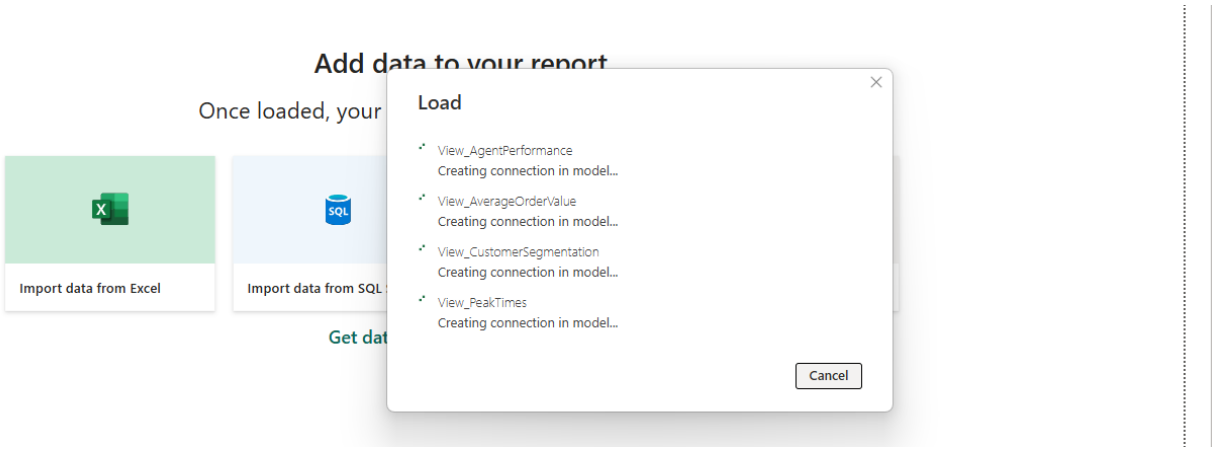
- DirectQuery Mode: Power BI does NOT import data. Instead, every time you open a report or interact with a visual (like filter or click), it queries the database live.
  - Always live data, no refresh needed.
  - Slower performance (depends on database speed and network).
  - Useful for very large datasets where importing is impractical.
  - Database needs to be highly optimized (indexes, partitions, etc.) to support fast querying.
  - Good when you need real-time reporting or when datasets are very large.

Click on Ok.

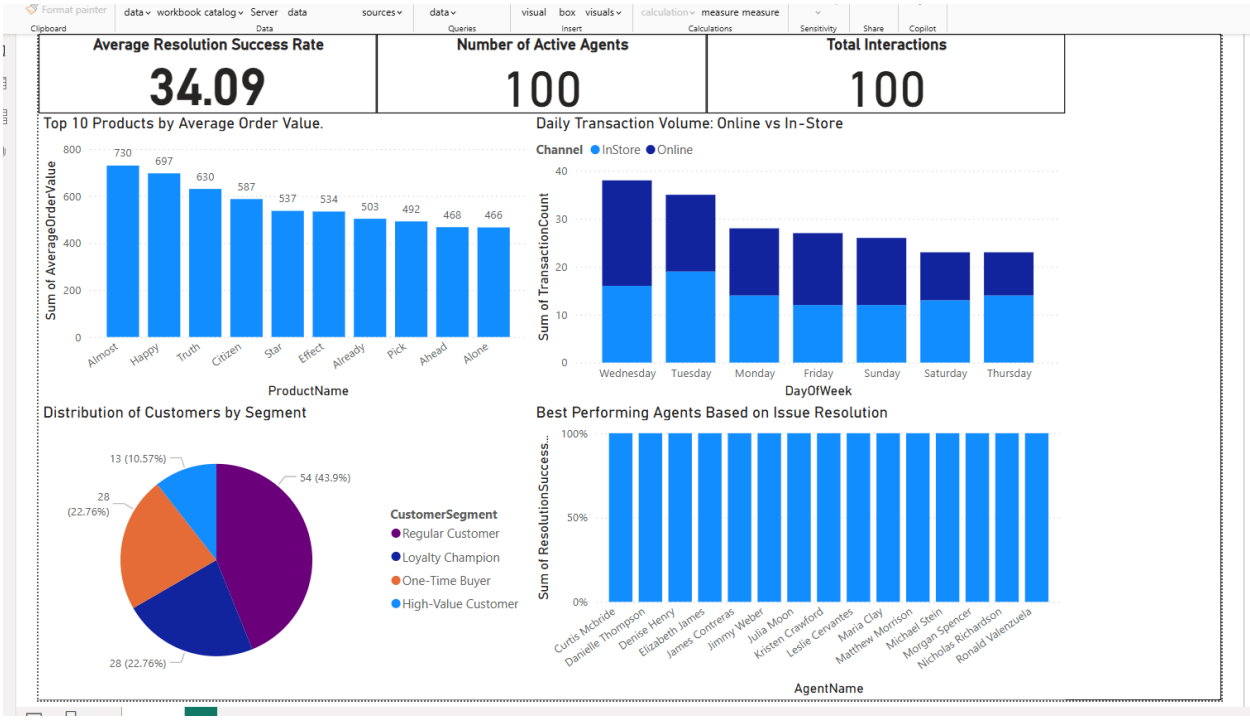
Select the views which we have created.



Click on Load.

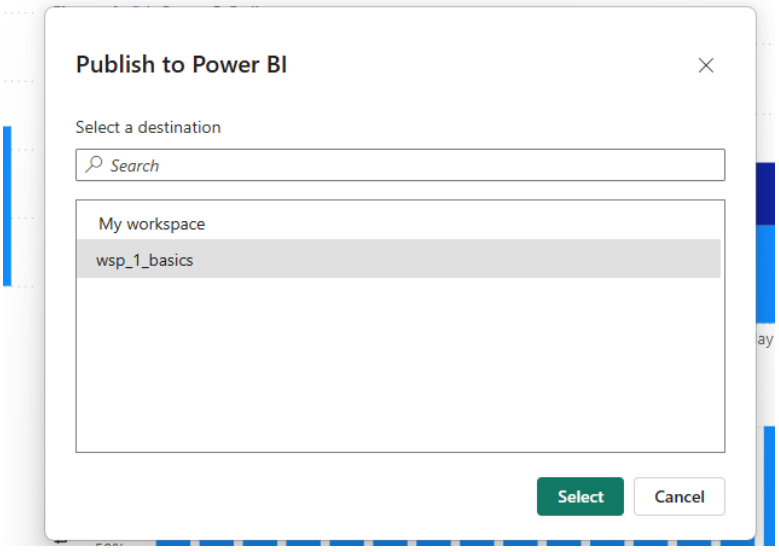


We have created this dashboard in PowerBI



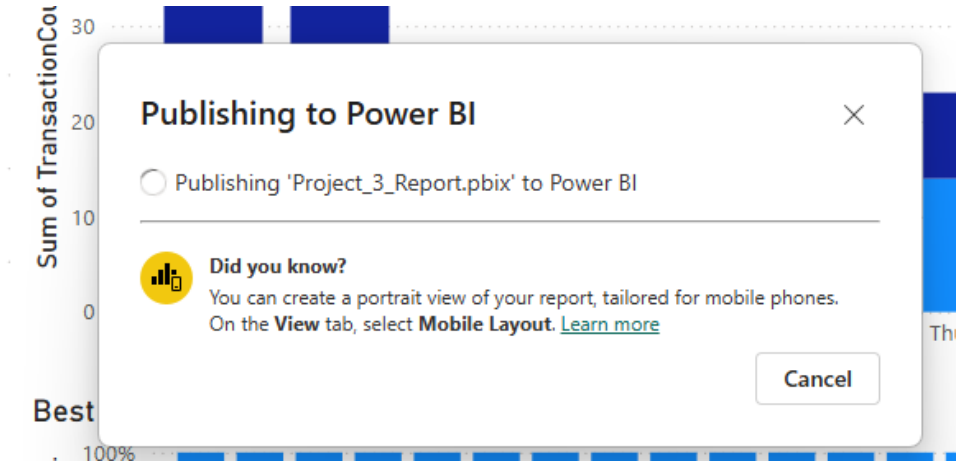
Step 5: Publish to Fabric Workspace

Now, let's publish it into Fabric Workspace.



Click on Publish icon in Home -> Publish

Select the Fabric Workspace



Check the report in to fabric workspace.

Points to Remember

- Ingest data properly into Bronze layer (ADLS folders organized upfront).
- Create Parent tables first (Customers, Products) to avoid constraint errors.
- Filter null records early and remove duplicates before moving to Silver layer.
- Use Alter Row in dataflows for correct upsert operations into SQL tables.
- Secure passwords via Azure Key Vault instead of hardcoding credentials.
- Gold Layer views simplify business insights (AOV, Customer Segments, Peak Times, Agent Performance).
- Import Mode = better performance, DirectQuery = live data, slower.
- Apply Top N filters in Power BI visuals (Top 10 products, Top 10 agents) for clarity.

- Gold Layer is for business reporting, make it clean, aggregated, and ready for analysis.