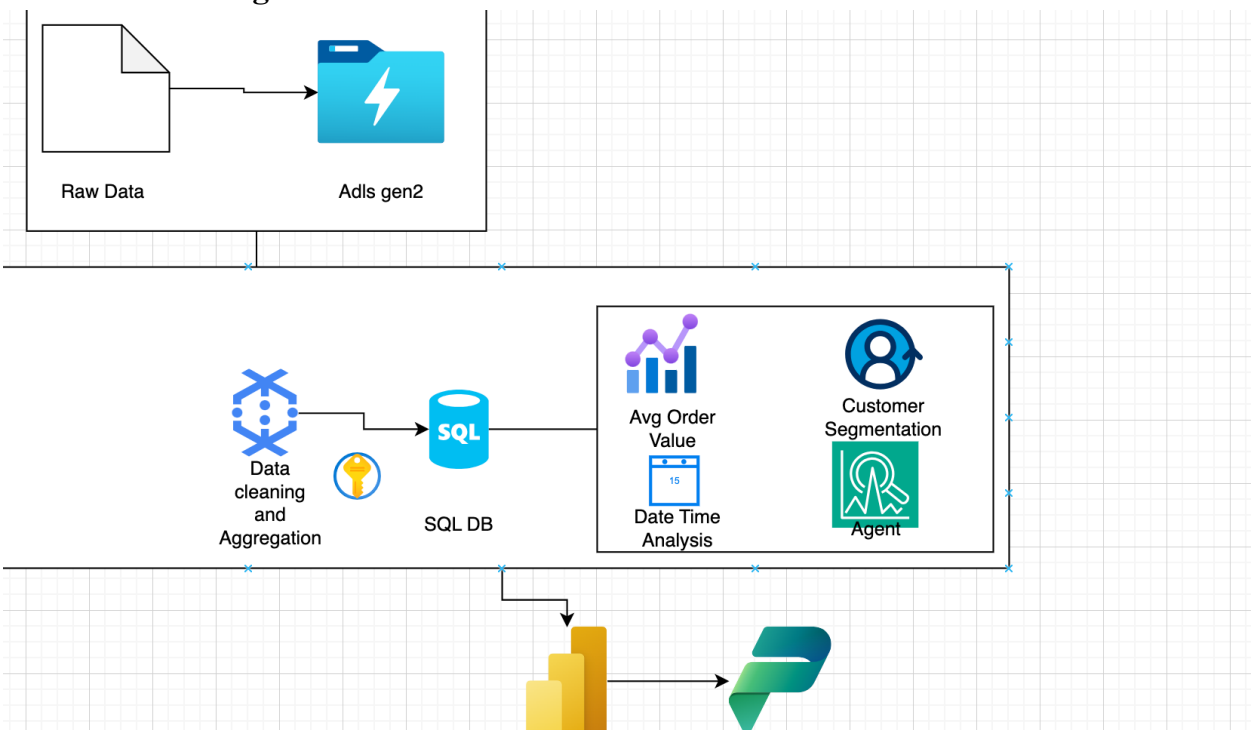
Customer 360 Data Integration

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.

Architecture Diagram:



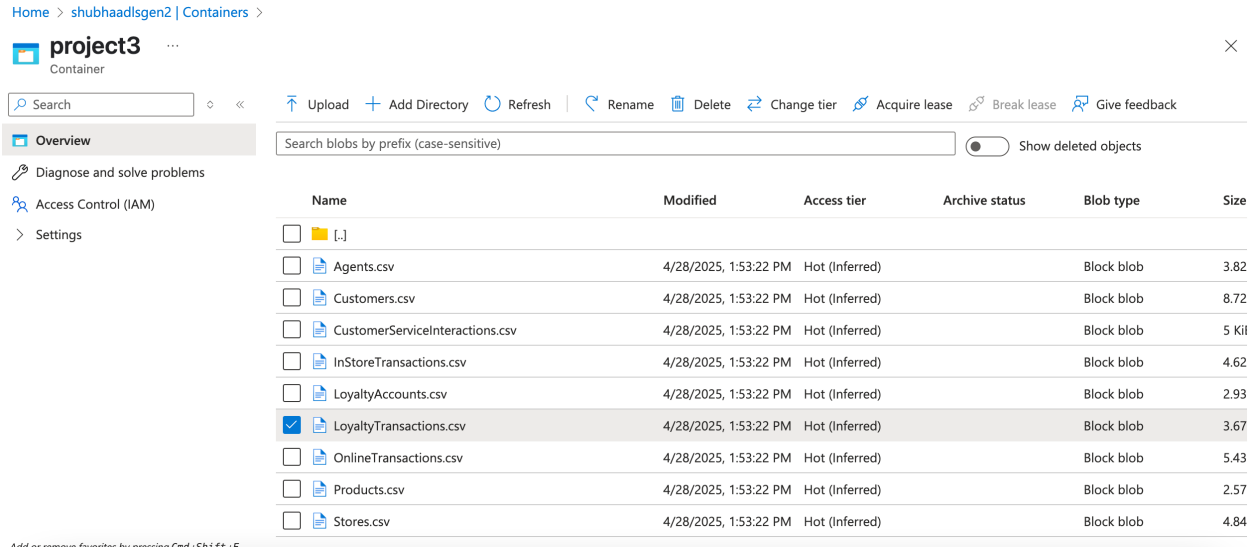
Tools and Technologies

- Azure Synapse Analytics
- Azure Data Lake Storage (ADLS)
- Azure SQL Database
- Power BI

Step-by-Step Process

Step 1: Ingest Data

- **Objective**: Ingest data from local system into ADLS for raw storage



Step 2: Create Tables in Azure SQL Database

Once the data is ingested into ADLS Gen2, we need to create tables in **Azure SQL Database** to store the cleansed and transformed data.

Run Cancel Disconnect Change | Database: shubhaproject | Estimated Plan Enabl

To Notebook

```sql
CREATE TABLE Agents (
    AgentID INT PRIMARY KEY,
    Name VARCHAR(100),
    Department VARCHAR(50),
    Shift VARCHAR(50)
);
select * from Agents
CREATE TABLE Customers(
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100),
    Address VARCHAR(255)
);
select * from Customers
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)
);
```

```sql
CREATE TABLE Stores (
    StoreID INT PRIMARY KEY,
    Location VARCHAR(100),
    Manager VARCHAR(100),
    OpenHours VARCHAR(50)
);
select * from Stores
CREATE TABLE InStoreTransactions (
    TransactionID INT PRIMARY KEY,
    CustomerID INT,
    StoreID INT,
    DateTime DATETIME,
    Amount float,
    PaymentMethod VARCHAR(50),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (StoreID) REFERENCES Stores(StoreID)
);
select * from InStoreTransactions

CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    Name VARCHAR(100),
    Category VARCHAR(50),
    Price float
);
select * from Products
```

```
45    CREATE TABLE Products (
46        ProductID INT PRIMARY KEY,
47        Name VARCHAR(100),
48        Category VARCHAR(50),
49        Price float
50    );
51    select * from Products
52    CREATE TABLE OnlineTransactions (
53        OrderID INT PRIMARY KEY,
54        CustomerID INT,
55        ProductID INT,
56        DateTime DATETIME,
57        PaymentMethod VARCHAR(50),
58        Amount float,
59        Status VARCHAR(20),
60        FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
61        FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
62    );
63    select * from OnlineTransactions
64    CREATE TABLE LoyaltyAccounts (
65        LoyaltyID INT PRIMARY KEY,
66        CustomerID INT,
67        PointsEarned INT,
68        TierLevel VARCHAR(20),
69        JoinDate DATE,
70        FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
71    );
```

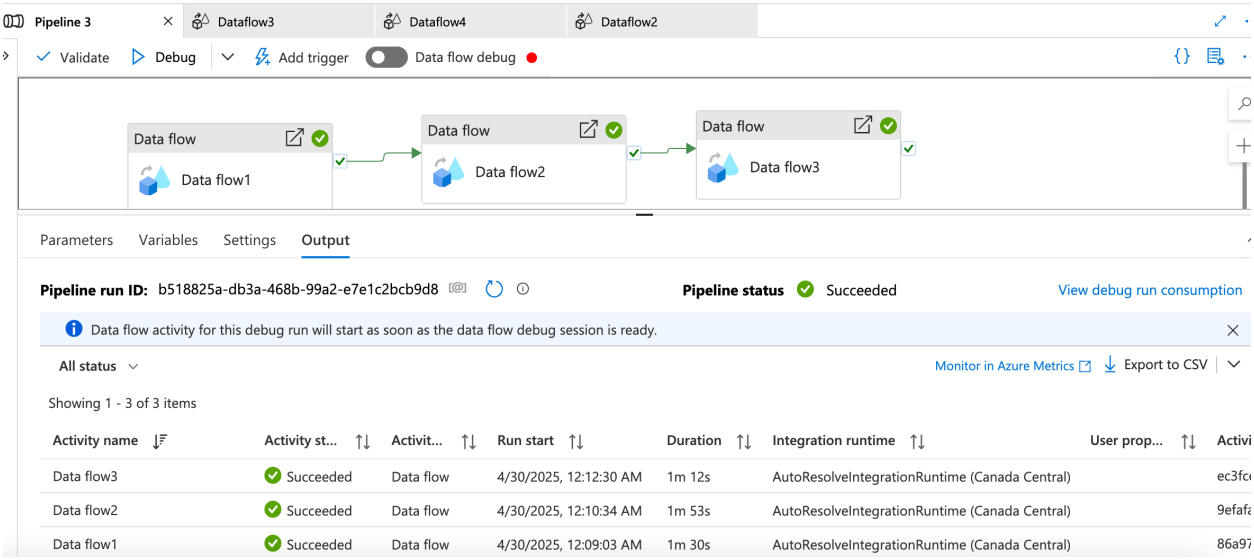
```
CREATE TABLE LoyaltyTransactions (
    LoyaltyID INT,
    DateTime DATETIME,
    PointsChange INT,
    Reason VARCHAR(100),
    PRIMARY KEY (LoyaltyID, DateTime),
    FOREIGN KEY (LoyaltyID) REFERENCES LoyaltyAccounts(LoyaltyID)
);
select * from LoyaltyTransactions
```

Step 3: Clean and transform the data using **Mapping Data Flows** in ADF, then load the cleansed data into Azure SQL Database.

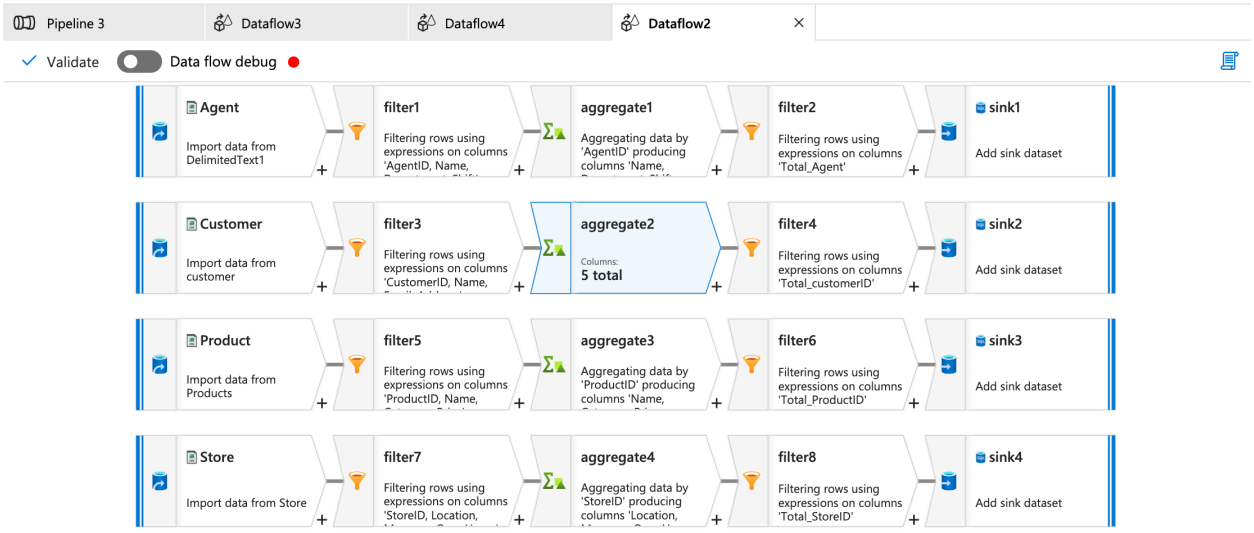
In this project, we're using three separate dataflows to handle different stages of data processing.

1. **Dataflow 1** processes independent tables (Agent, Customer, Store, Product).
2. **Dataflow 2** handles transactional data (Online Transactions, Instore Transactions, Loyalty Activity) while managing foreign key relationships.
3. **Dataflow 3** focuses on processing **Loyalty Transactions**.

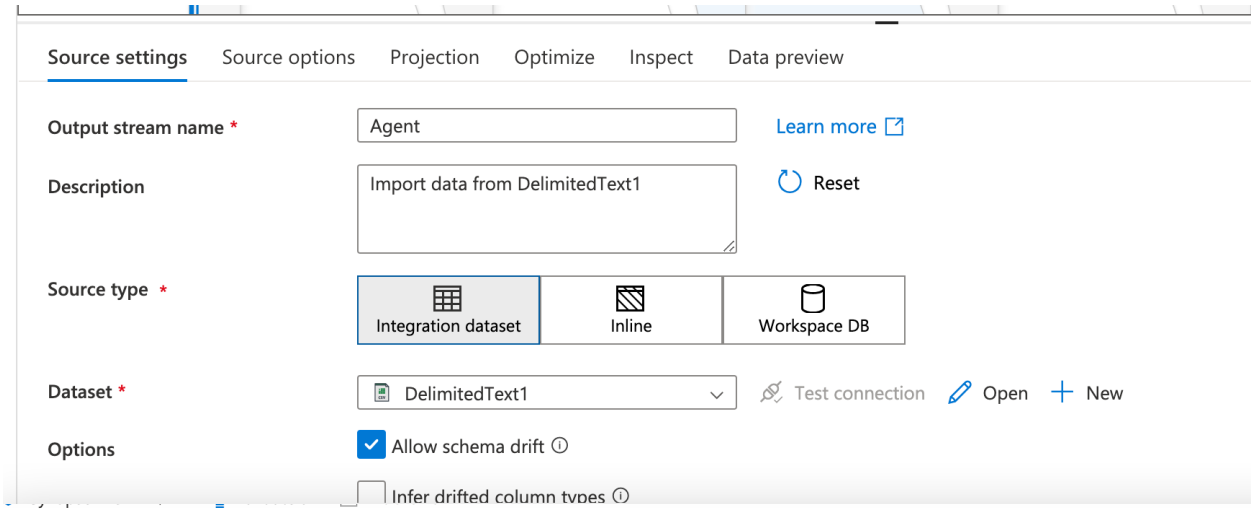
These dataflows are combined and executed in sequence to ensure a clean, structured flow of data through the pipeline.



let’s go through one of the cleaning transforming flow

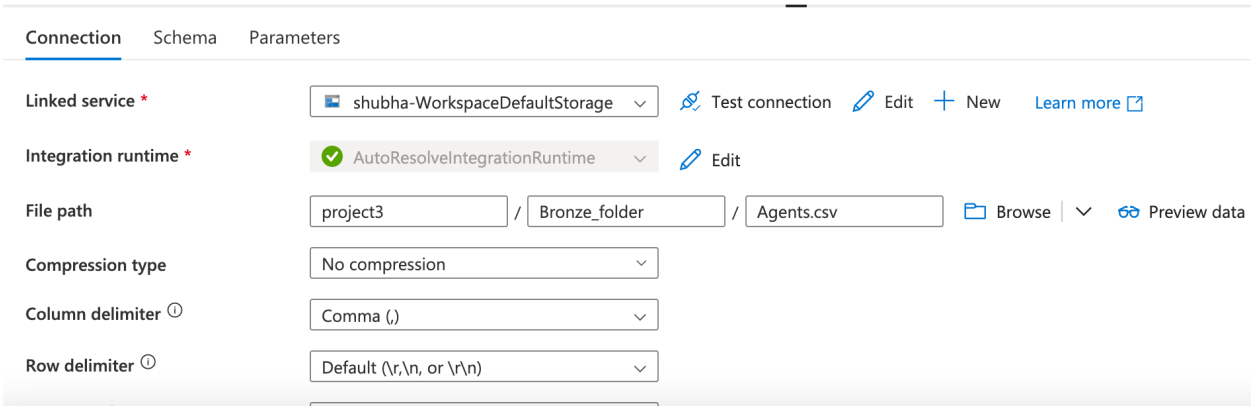


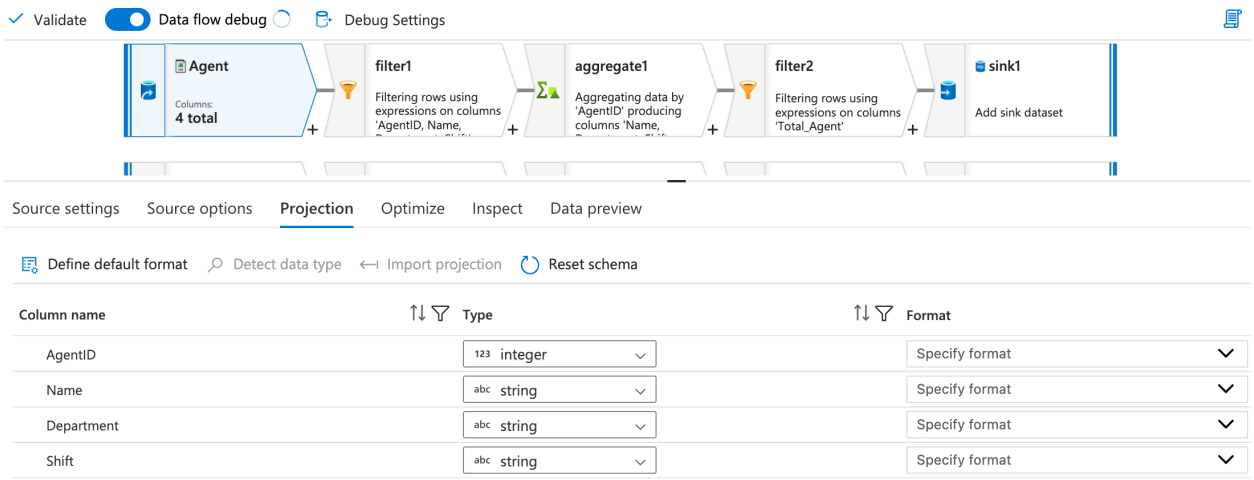
Source:


Source settings
Source options
Projection
Optimize
Inspect
Data preview
Output stream name *
Agent
Learn more
Description
Import data from DelimitedText1
Reset
Source type *
Integration dataset
Inline
Workspace DB
Dataset *
DelimitedText1
Test connection
Open
New
Options
Allow schema drift
Infer drifted column types


Pipeline 3
Dataflow3
Dataflow4
Dataflow2
DelimitedText1

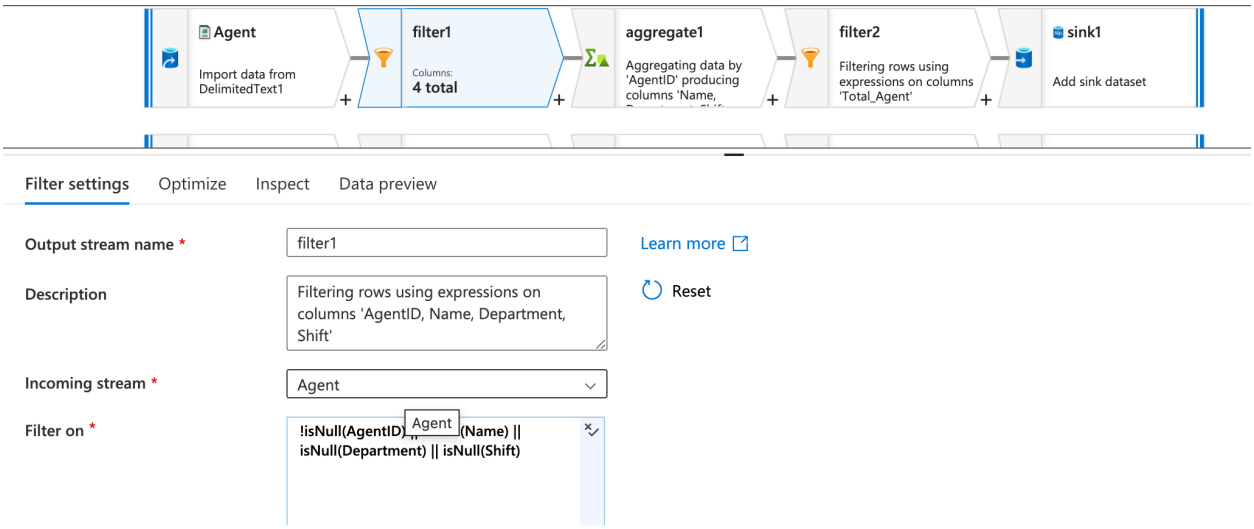

CSV
DelimitedText
DelimitedText1


Connection
Schema
Parameters
Linked service *
shubha-WorkspaceDefaultStorage
Test connection
Edit
New
Learn more
Integration runtime *
AutoResolveIntegrationRuntime
Edit
File path
project3
Bronze_folder
Agents.csv
Browse
Preview data
Compression type
No compression
Column delimiter
Comma (,)
Row delimiter
Default (\r,\n, or \r\n)



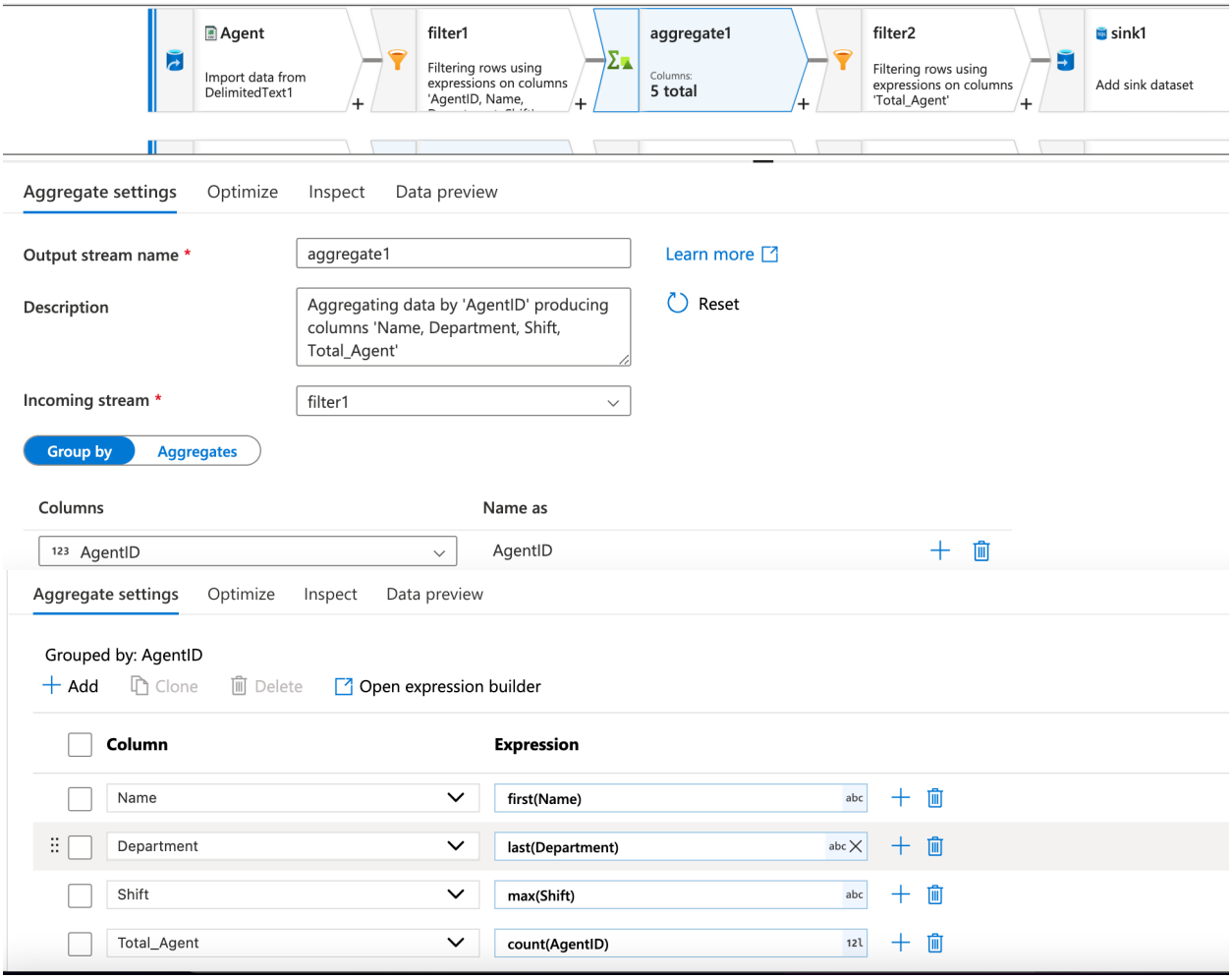
Filter Transformation:

- **Purpose**: This filter removes any rows where the **AgentID**, **Name**, **Department**, or **Shift** are null.



Aggregate Transformation:

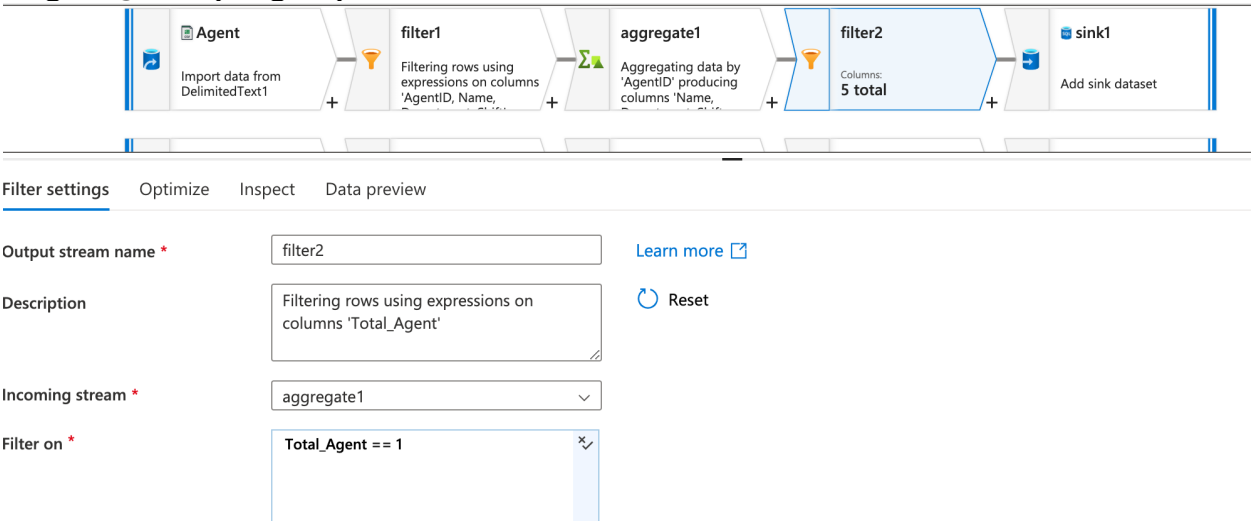
- **Purpose**: This step groups the data by **AgentID** and performs aggregation to clean and consolidate the data.
- **GroupBy**: The data is grouped by **AgentID**.
- **Aggregations**:
  - **Name**: The `first(Name)` function ensures that for each **AgentID**, the first **Name** encountered in the group is retained.
  - **Department**: The `last(Department)` function ensures that the last department value for each **AgentID** is kept.
  - **Shift**: The `max(Shift)` function retains the maximum (most recent) **Shift** value.
  - **Total_Agent**: The `count(AgentID)` counts the total number of occurrences of **AgentID** in the group.



Filter Transformation (After Aggregate):

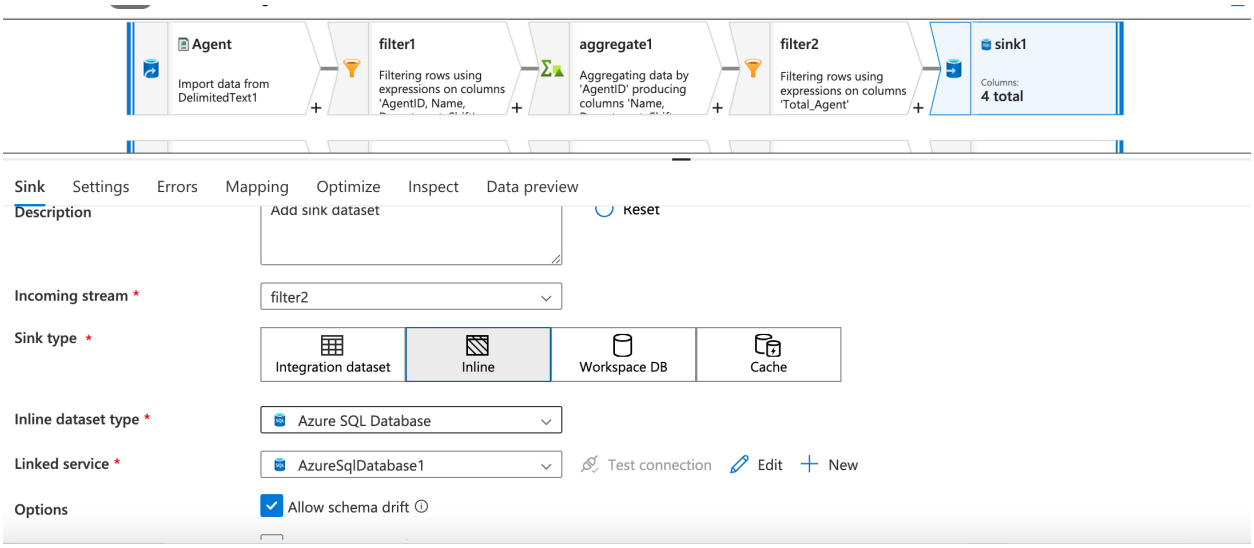
- **Purpose**: This filter ensures that only the groups where there is exactly **1** occurrence of **AgentID** are passed forward.
- **Logic**: If there are duplicates in the **AgentID** column (e.g., multiple records for the same agent), they are removed, ensuring that only records with a

single **AgentID** per group are retained.



Sink Transformation:

- **Purpose**: After all the transformations are applied, the data is written to the target SQL Server table.
- **Linked Service**: The sink is connected to an Azure SQL Database using the **AzureSqlDatabase1** linked service.



Edit linked service

Azure SQL Database Learn more

Database name *

shubhaproject

Authentication type *

SQL authentication

User name *

admin123

Password | Azure Key Vault

AKV linked service *

AzureKeyVault1

Secret name *

passwordsql

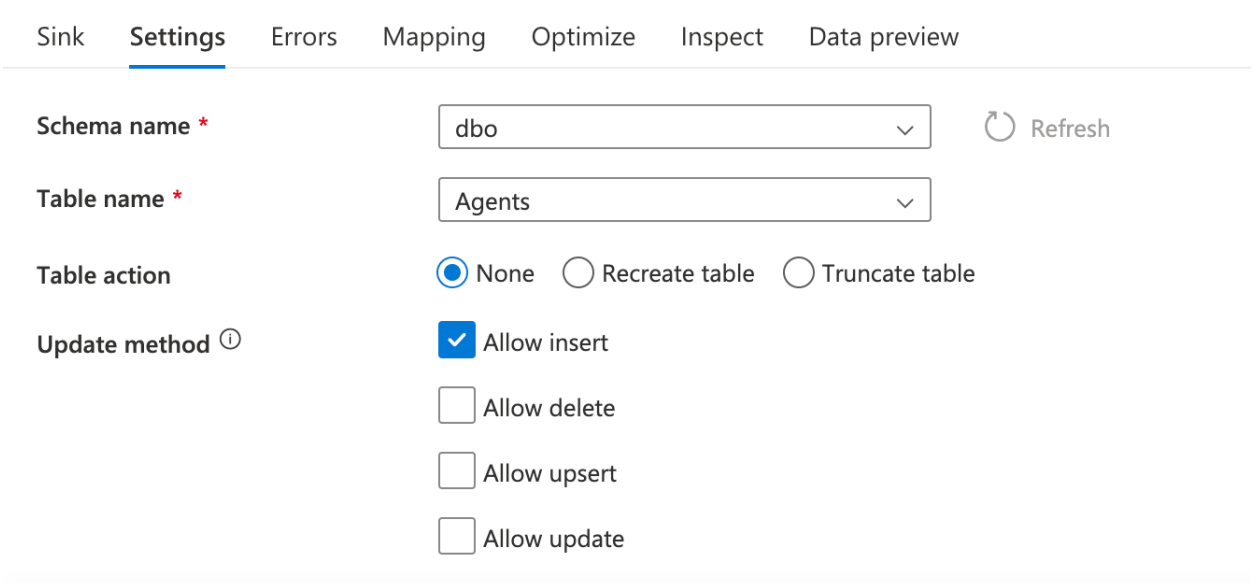
Edit

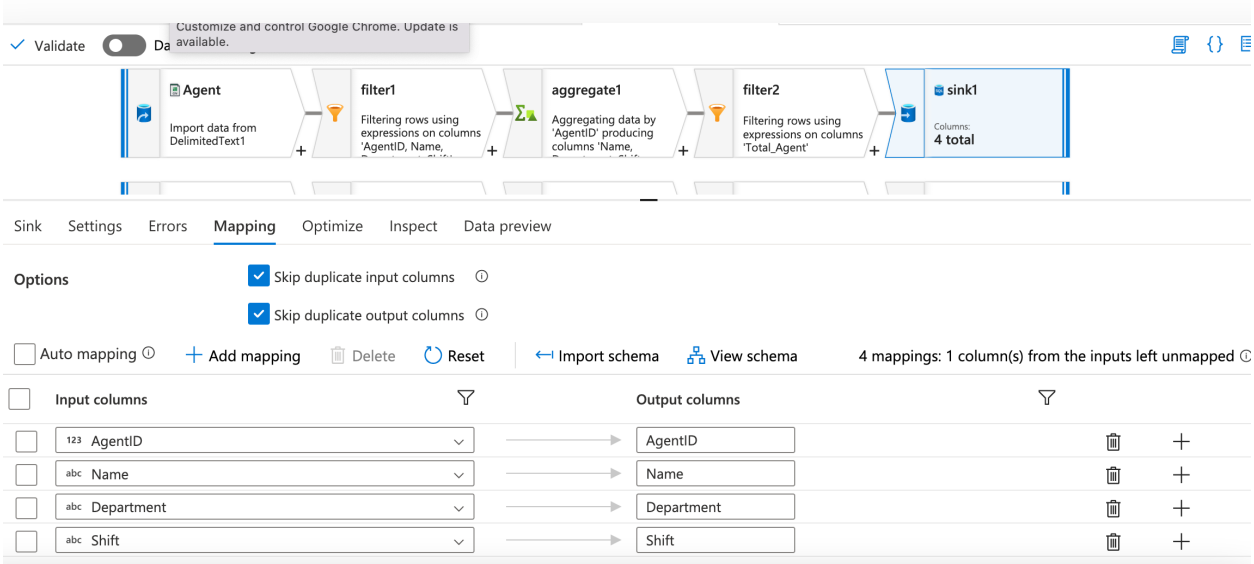
Secret version

27e191ada0014522a709f56b07b61397

Edit

Save | Cancel | Test connection





Data got successfully loaded in SQL tables

```
7  select * from Agents
```

Results grid

sages

	AgentID	Name	Department	Shift
3	3	Garrett Knapp	Sales	Morning
4	4	Daryl Benjamin	Sales	Evening
5	5	Matthew Long	Sales	Morning
6	6	Patricia Rhodes	Sales	Morning
7	7	Elizabeth James	Technical Supp…	Morning
8	8	Teresa Bennett	Technical Supp…	Morning
9	9	Amber Ross	Billing	Afterno…
1…	10	Tonya Jones	Technical Supp…	Evening
1…	11	Curtis Mcbride	Sales	Afterno…
1…	12	Justin Michael	Billing	Morning
1…	13	Scott Flowers	Billing	Evening
1…	14	Michael Jackson	Technical Supp…	Afterno…
1…	15	Kristen Crawford	Sales	Afterno…
1…	16	Jo Meyers	General Inquiry	Evening
1…	17	Brandon Jimenez	Technical Supp…	Evening
1…	18	Melissa White	General Inquiry	Morning
1…	19	Eddie Pierce	Sales	Evening
2…	20	Julia Owens	Sales	Evening

```
4  select * from Customers
```

sults Messages

CustomerID	Name	Email	Address
1	Mrs. Crystal Carroll	kaiserjacob@example.org	7566 Kelly Shoals Apt. 207, Port Joanne, FM 25401
2	Debra Newman DVM	brownashley@example.net	72713 Nelson Lodge Suite 286, Leonfort, NJ 49406
3	Thomas Mason	frankmark@example.com	3024 Riley Ferry Suite 573, Chadberg, MT 40822
4	Karla Hill	jnelson@example.org	9899 Hubbard Station, Lake Vanessa, CT 97851
5	Jeffrey Underwood	daniel48@example.org	11764 Hannah Plaza, Lake Vickichester, AR 85240
6	Justin Lowe	qwilson@example.org	494 Danielle Causeway, Lake Benjamin, NV 21217
7	Jason Daniels	elliottsonia@example.net	37283 Ramsey Light Suite 640, New Howard, IA 48396
8	Jennifer Hill	petersenrebecca@example.org	02411 Ruiz Corners Apt. 346, Oconnorberg, SD 93666
9	Stephanie Richardson	obates@example.net	974 Alyssa Heights, Farrellfurt, OH 22597
10	Jo Zimmerman	samuel23@example.net	25747 Clark Harbor, New Emilyport, RI 59995
11	Andrew Williams	robinsonjulian@example.org	431 Thomas Orchard, South Aliciaside, ME 26651
12	Robin Odom	deborah35@example.com	75190 John Ferry Apt. 305, North Renee, FL 93070

```
25  select * from CustomerServiceInteractions
26  CREATE TABLE Stores (
```

Results | Messages

	InteractionID	CustomerID	DateTime	AgentID	IssueType	ResolutionStatus
1	1	55	2025-01-20 08:17:26.000	33	Technical Issue	Resolved
2	2	93	2025-01-16 18:52:51.000	89	Technical Issue	Escalated
3	3	53	2025-03-16 13:10:05.000	96	Complaint	Resolved
4	4	30	2025-02-27 13:45:03.000	20	Other	Pending
5	5	49	2025-02-16 07:56:45.000	48	Technical Issue	Pending
6	6	42	2025-02-13 21:05:41.000	56	Product Inquiry	Resolved
7	7	73	2025-03-01 13:19:50.000	38	Other	Escalated
8	8	50	2025-01-15 16:44:36.000	66	Other	Escalated
9	9	63	2025-03-11 10:58:12.000	32	Complaint	Pending
10	10	21	2025-03-15 05:00:10.000	78	Product Inquiry	Escalated
11	11	69	2025-01-03 21:23:28.000	4	Complaint	Escalated
12	12	85	2025-02-17 10:30:24.000	34	Billing	Escalated

```
32  select * from Stores
33  CREATE TABLE InStoreTransactions (
```

Results | Messages

	StoreID	Location	Manager	OpenHours
1	1	Angelamouth	Michelle Robinson	8:00 AM – 8:00 PM
2	2	Tammychester	Jonathan Frey	8:00 AM – 8:00 PM
3	3	Vanessachester	Barbara Lynch	8:00 AM – 7:00 PM
4	4	Maytown	Frederick Dean	9:00 AM – 8:00 PM
5	5	Kathleenfort	Donald Ellis	10:00 AM – 7:00 PM
6	6	South Martin	Tamara Hernandez	10:00 AM – 6:00 PM
7	7	Rebeccastad	Amanda Diaz	9:00 AM – 8:00 PM
8	8	North Mariaburgh	Lisa Jackson	9:00 AM – 7:00 PM
9	9	Stephenfurt	Michael Hays	10:00 AM – 9:00 PM
10	10	Thompsonshire	Mary Barker	9:00 AM – 9:00 PM
11	11	East Justin	Jose Brown	9:00 AM – 8:00 PM

Results grid

```
43  select * from InStoreTransactions
44
```

Results | Messages

	TransactionID	CustomerID	StoreID	DateTime	Amount	PaymentMethod
1	1	88	100	2025-03-12 06:49:37.000	135.74000549316406	Credit Card
2	2	60	12	2025-01-21 23:09:14.000	34.13999938964844	Credit Card
3	3	54	51	2025-01-29 22:12:51.000	111.41999816894531	Debit Card
4	4	32	94	2025-02-08 23:51:36.000	179.4499969482422	Debit Card
5	5	33	45	2025-01-18 04:05:21.000	177.9600067138672	Gift Card
6	6	92	91	2025-03-09 03:29:17.000	196.50999450683594	Debit Card
7	7	44	10	5-02-19 19:28:55.000	198.1199951171875	Debit Card
8	8	91	51	2025-02-15 14:38:42.000	19.479999542236328	Gift Card
9	9	76	77	2025-01-19 09:16:43.000	196.52999877929688	Debit Card
10	10	65	12	2025-01-30 06:43:44.000	83.98999786376953	Gift Card
11	11	13	77	2025-01-29 21:46:00.000	180.80999755859375	Credit Card

Results grid

```
51  select * from Products
52  CREATE TABLE OnlineTransactions (
```

Results | Messages

Results grid

	ProductID	Name	Category	Price
1	1	Consider	Electronics	183.50999450683594
2	2	At	Toys	403.2799987792969
3	3	Risk	Clothing	186.52999877929688
4	4	Serious	Clothing	275.3900146484375
5	5	Discover	Clothing	23.389999389648438
6	6	Area	Books	316.8399963378906
7	7	Happy	Toys	137.5800018310547
8	8	Effect	Electronics	202.6300048828125
9	9	Star	Books	119.08000183105469
10	10	Structure	Books	273.7099914550781
11	11	Join	Books	253.6999969482422
12	12	Course	Books	298.29998779296875

```
63  select * from OnlineTransactions
```

Results | Messages

	OrderID	CustomerID	ProductID	DateTime	PaymentMethod	Amount	St
1	1	34	91	2025-01-20 22:34:31.000	Gift Card	189.89999389648438	N
2	2	17	59	2025-02-10 20:42:13.000	Gift Card	193.25	N
3	3	13	57	2025-02-16 18:26:04.000	Debit Card	183.82000732421875	N
4	4	90	19	2025-02-21 05:04:59.000	Debit Card	175.3699951171875	N
5	5	61	99	2025-01-19 05:42:37.000	Credit Card	174.6199951171875	N
6	6	75	67	2025-02-26 09:47:16.000	Debit Card	144.47999572753906	N
7	7	63	44	2025-02-07 09:48:20.000	PayPal	91.73999786376953	N
8	8	55	6	2025-01-19 20:32:41.000	Credit Card	16.579999923706055	N
9	9	96	67	2025-02-16 19:22:55.000	Credit Card	22.68000030517578	N
10	10	49	13	2025-01-25 02:24:02.000	Credit Card	27.549999237060547	N
11	11	6	91	2025-01-09 14:00:29.000	Credit Card	94.30999755859375	N
12	12	26	53	2025-01-25 03:04:55.000	PayPal	87.73999786376953	N

2025-02-26 09:47:16.000

```
72    select * from LoyaltyAccounts
```

Results | Messages

	LoyaltyID	CustomerID	PointsEarned	TierLevel	JoinDate
1	1	5	399	Gold	2023-11-27
2	2	17	606	Gold	2021-06-26
3	3	13	4782	Gold	2023-04-11
4	4	62	1824	Platinum	2020-12-17
5	5	84	894	Gold	2020-03-18
6	6	68	4291	Platinum	2020-09-25
7	7	90	2845	Bronze	2022-09-26
8	8	51	1697	Silver	2022-04-14
9	9	5	1912	Silver	2023-07-03
10	10	69	3846	Bronze	2024-09-28
11	11	53	2155	Silver	2022-04-24
12	12	86	4396	Bronze	2020-11-01

```
81    select * from LoyaltyTransactions
82
```

Results | Messages

	LoyaltyID	DateTime	PointsChange	Reason
1	1	2025-01-13 20:04:21.000	166	Purchase
2	6	2025-01-12 22:56:22.000	29	Referral
3	8	2025-02-21 03:46:32.000	122	Referral
4	9	2025-02-24 08:12:28.000	186	Correction
5	13	2025-01-09 07:30:47.000	172	Promotion
6	14	2025-01-17 01:13:54.000	108	Referral
7	15	2025-03-15 21:15:07.000	-34	Referral
8	20	2025-03-15 14:39:12.000	-33	Promotion
9	26	2025-02-05 12:59:40.000	53	Purchase
10	29	2025-01-19 08:39:30.000	27	Correction
11	30	2025-03-15 05:27:29.000	14	Referral
12	34	2025-03-14 06:28:18.000	57	Referral

Results grid

Step 4: Creating Views

- View 1 - for **Average Order Value (AOV)**
    - SUM(Amount) / COUNT(OrderID) per product, category, and location.

```sql
CREATE VIEW View_AverageOrderValue AS
SELECT
    p.ProductID,
    p.Name AS ProductName,
    p.Category,
    s.StoreID,
    s.Location,
    SUM(t.Amount) / COUNT(t.OrderID) AS AverageOrderValue
FROM
    OnlineTransactions t
INNER JOIN
    dbo.Products p ON t.ProductID = p.ProductID
LEFT JOIN
    dbo.Stores s ON s.StoreID IS NOT NULL  -- since online might not have a store directly, kept flexible
GROUP BY
    p.ProductID, p.Name, p.Category, s.StoreID, s.Location;
```

Results Messages

	ProductID	ProductName	Category	StoreID	Location	AverageOrderValue
2	44	Movement	Toys	91	Dariusberg	101.10499954223633
3	66	Leader	Electronics	6	South Martin	104.86000061035156
4	20	Talk	Electronics	48	Laurenberg	114.12000274658203
5	43	Pick	Clothing	27	West Karentown	164.0399932861328
6	52	Final	Home Goods	58	South Joseph	182.25999450683594
7	67	Hand	Electronics	93	West Terri	102.7699966430664
8	29	Seven	Home Goods	56	North Helen	141.88999938964844
9	2	At	Toys	9	Stephenfurt	159.1300048828125
1…	66	Leader	Electronics	29	Beverlybury	104.86000061035156
1…	61	Defense	Home Goods	66	West Ryanport	127.51000213623047

- 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."

Run | Cancel | Disconnect | Change | Database: shubhaproject | Estimated Plan | Enable Actual Plan | Parse | Enable SQL

```
CREATE VIEW View_CustomerSegmentation1 AS
WITH CustomerSpending AS (
    SELECT
        c.CustomerID,
        c.Name,
        SUM(t.Amount) AS TotalSpend,
        COUNT(t.OrderID) AS PurchaseFrequency,
        l.TierLevel
    FROM
        Customers c
    LEFT JOIN
        OnlineTransactions t ON c.CustomerID = t.CustomerID
    LEFT JOIN
        LoyaltyAccounts l ON c.CustomerID = l.CustomerID
    GROUP BY
        c.CustomerID, c.Name, l.TierLevel
),
Percentile AS (
    -- Calculate the 90th percentile spend threshold over the entire dataset
    SELECT
        PERCENTILE_CONT(0.9) WITHIN GROUP (ORDER BY TotalSpend) OVER () AS HighValueThreshold
    FROM CustomerSpending
)
SELECT
    cs.CustomerID,
    cs.Name,
    cs.TotalSpend,
    cs.PurchaseFrequency,
    cs.TierLevel,
    CASE
        WHEN cs.TotalSpend >= (SELECT TOP 1 HighValueThreshold FROM Percentile) THEN 'High-Value Customer'
        WHEN cs.PurchaseFrequency = 1 THEN 'One-Time Buyer'
        WHEN cs.TierLevel = 'Gold' OR cs.TierLevel = 'Platinum' THEN 'Loyalty Champion'
        ELSE 'Regular Customer'
    END AS CustomerSegment
FROM
    CustomerSpending cs;
```

Results | Messages

	Customer…	Name	TotalSpend	Purch…	TierLevel	CustomerS…
43	41	Erin Barrett	29.44000053405…	2	Bronze	Regular Custo…
44	43	Jose Jackson	17.37999916076…	1	Bronze	One-Time Buyer
45	49	Austin Marshall	27.54999923706…	1	Bronze	One-Time Buyer
46	51	Tyler Knight	460.8900146484…	3	Bronze	High-Value Cu…
47	69	Mrs. Cassandra Ford	*NULL*	0	Bronze	Regular Custo…
48	73	Linda Boyd	295.1199951171…	2	Bronze	Regular Custo…
49	83	Charles Smith	49.43999862670…	1	Bronze	One-Time Buyer
50	86	Craig Peters	*NULL*	0	Bronze	Regular Custo…
51	90	Katherine Mccarthy	350.7399902343…	2	Bronze	High-Value Cu…

Results grid

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

To Notebook

```sql
CREATE VIEW View_PeakTimes AS
SELECT
    'Online' AS TransactionType,
    DATENAME(WEEKDAY, DateTime) AS DayOfWeek,
    DATEPART(HOUR, DateTime) AS HourOfDay,
    COUNT(OrderID) AS TransactionCount
FROM
   OnlineTransactions
GROUP BY
    DATENAME(WEEKDAY, DateTime), DATEPART(HOUR, DateTime)

UNION ALL

SELECT
    'Instore' AS TransactionType,
    DATENAME(WEEKDAY, DateTime) AS DayOfWeek,
    DATEPART(HOUR, DateTime) AS HourOfDay,
    COUNT(TransactionID) AS TransactionCount
FROM
    InStoreTransactions
GROUP BY
    DATENAME(WEEKDAY, DateTime), DATEPART(HOUR, DateTime);
```

Results | Messages

	TransactionType	DayOfWeek	HourOfDay	TransactionCount
1	Online	Monday	0	1
2	Online	Sunday	0	1

View 4 - for Number of interactions and resolution success rates per agent (**`ResolutionStatus`**).

To Notebook

```sql
CREATE VIEW View_AgentResolutionStats AS
SELECT
    a.AgentID,
    a.Name AS AgentName,
    a.Department,
    a.Shift,
    COUNT(csi.InteractionID) AS TotalInteractions,
    SUM(CASE WHEN csi.ResolutionStatus = 'Resolved' THEN 1 ELSE 0 END) AS ResolvedCount,
    ROUND(
        100.0 * SUM(CASE WHEN csi.ResolutionStatus = 'Resolved' THEN 1 ELSE 0 END)
        / NULLIF(COUNT(csi.InteractionID), 0), 2
    ) AS ResolutionSuccessRate
FROM
    dbo.Agents a
LEFT JOIN
    dbo.CustomerServiceInteractions csi ON a.AgentID = csi.AgentID
GROUP BY
    a.AgentID, a.Name, a.Department, a.Shift;
```

Results | Messages

	AgentID	AgentName	Department	Shift	TotalInteractions	ResolvedCount	ResolutionSucces
1	1	Jonathan Williams	General Inquiry	Morning	1	0	0.000000000000
2	2	Terry Edwards	Billing	Evening	1	0	0.000000000000
3	3	Garrett Knapp	Sales	Morning	0	0	*NULL*
4	4	Daryl Benjamin	Sales	Evening	2	0	0.000000000000
5	5	Matthew Long	Sales	Morning	1	0	0.000000000000

-