BOOTCAMP PROJECT 3

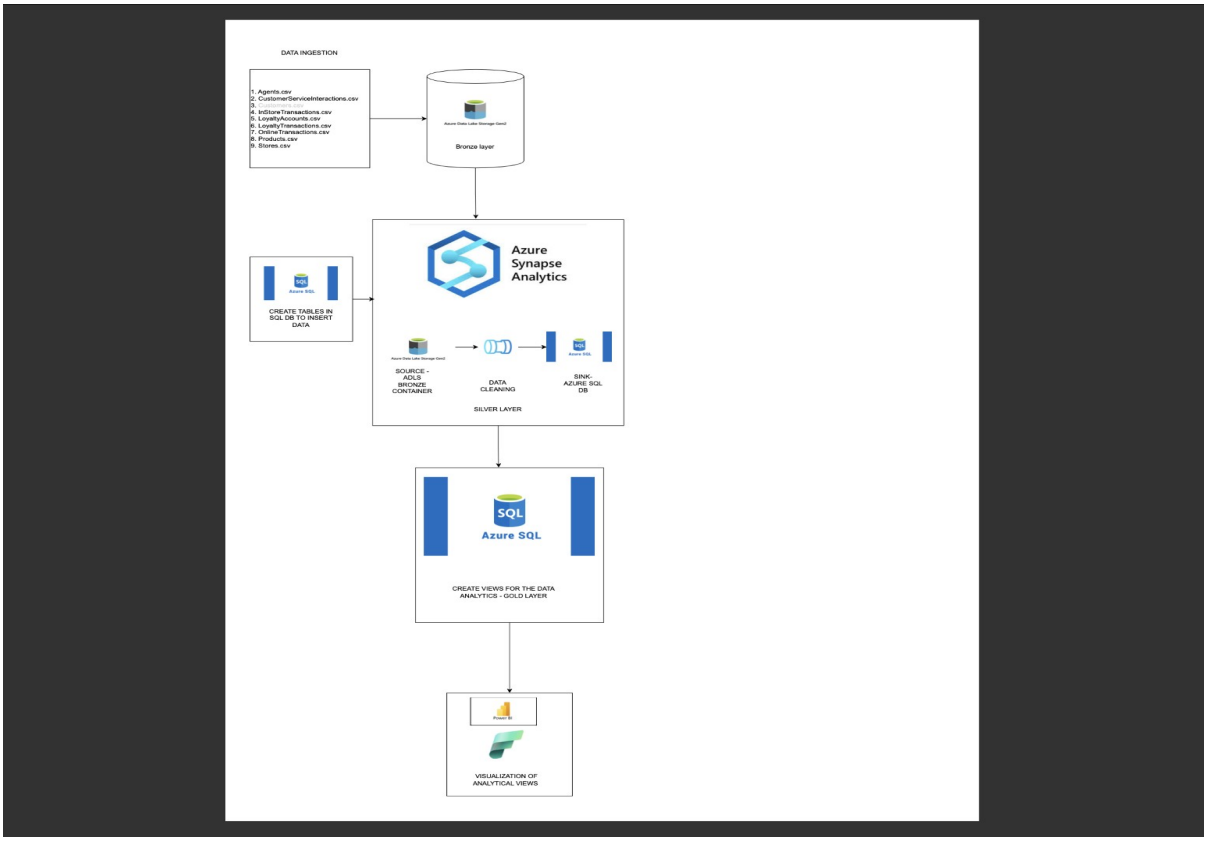
CUSTOMER 360 DATA INTEGRATION

SUBMITTED BY – DRUSYA SURESH

PROJECT 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 USED

- Azure Data Lake Storage Gen 2
- Azure synapse Analytics
- Azure sql database
- Power BI

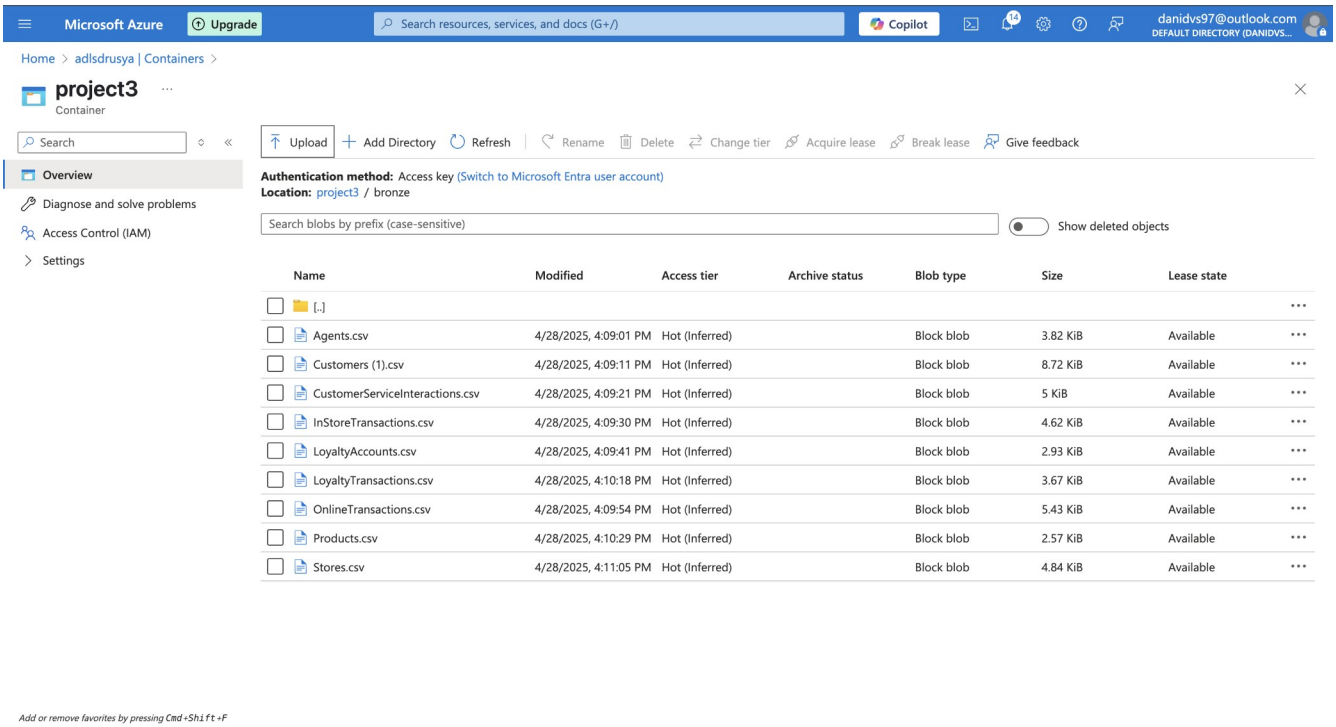
STEPS INVOLVED

- Upload the raw data sets to the ADLS storage – Bronze container.
- Create tables in SQL database to load the data.
- Using synapse pipelines clean and standardize the data and load to SQL database tables – Silver layer.
- Create views on top of the tables for aggregate and analytics data calculations.
- Visualize the views with the help of power BI and publish report in Fabric.

STEP 1:

Upload the dataset from Kaggle into the ADLS storage account in Bronze container – raw data.

Source - Kaggle's Customer 360 Data.



STEP 2:

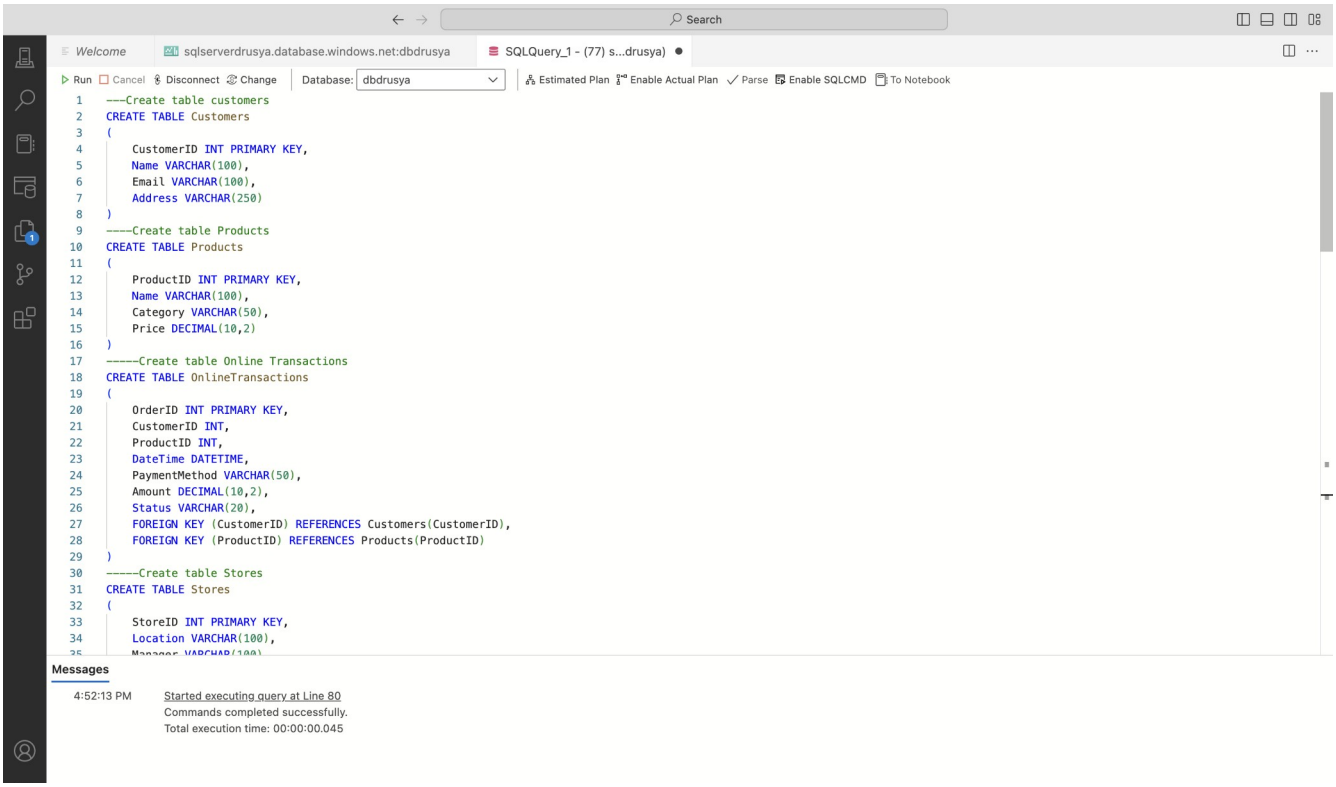
Create tables in Azure SQL to load the data after cleaning. The scripts used to create the tables and the screenshots are shown below.

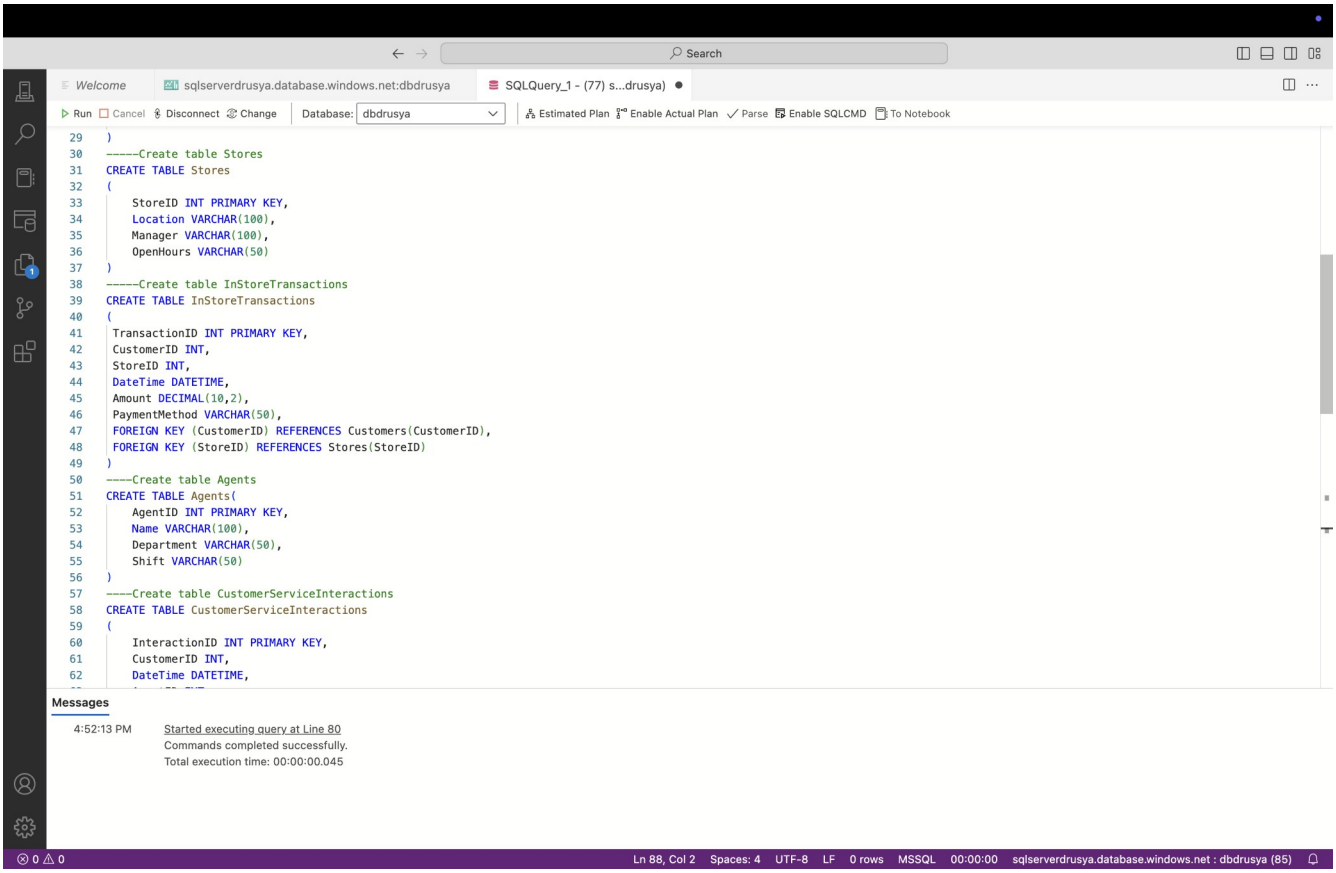
```
---Create table customers
CREATE TABLE Customers
(
CustomerID INT PRIMARY KEY,
Name VARCHAR(100),
Email VARCHAR(100),
Address VARCHAR(250)
)


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


-----Create table Online Transactions
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
CREATE TABLE Stores
(
StoreID INT PRIMARY KEY,
Location VARCHAR(100),
Manager VARCHAR(100),
```

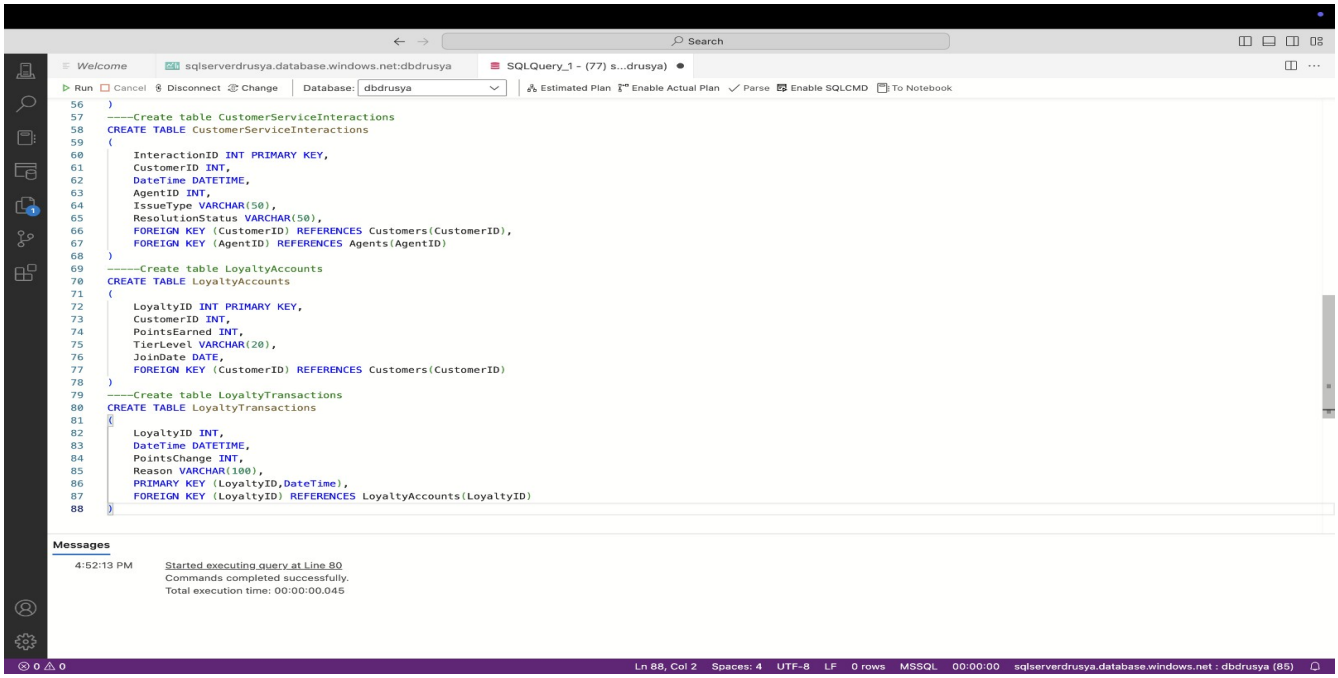
```
OpenHours VARCHAR(50)
)


-----Create table InStoreTransactions
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
CREATE TABLE Agents(
AgentID INT PRIMARY KEY,
Name VARCHAR(100),
Department VARCHAR(50),
Shift VARCHAR(50)
)

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

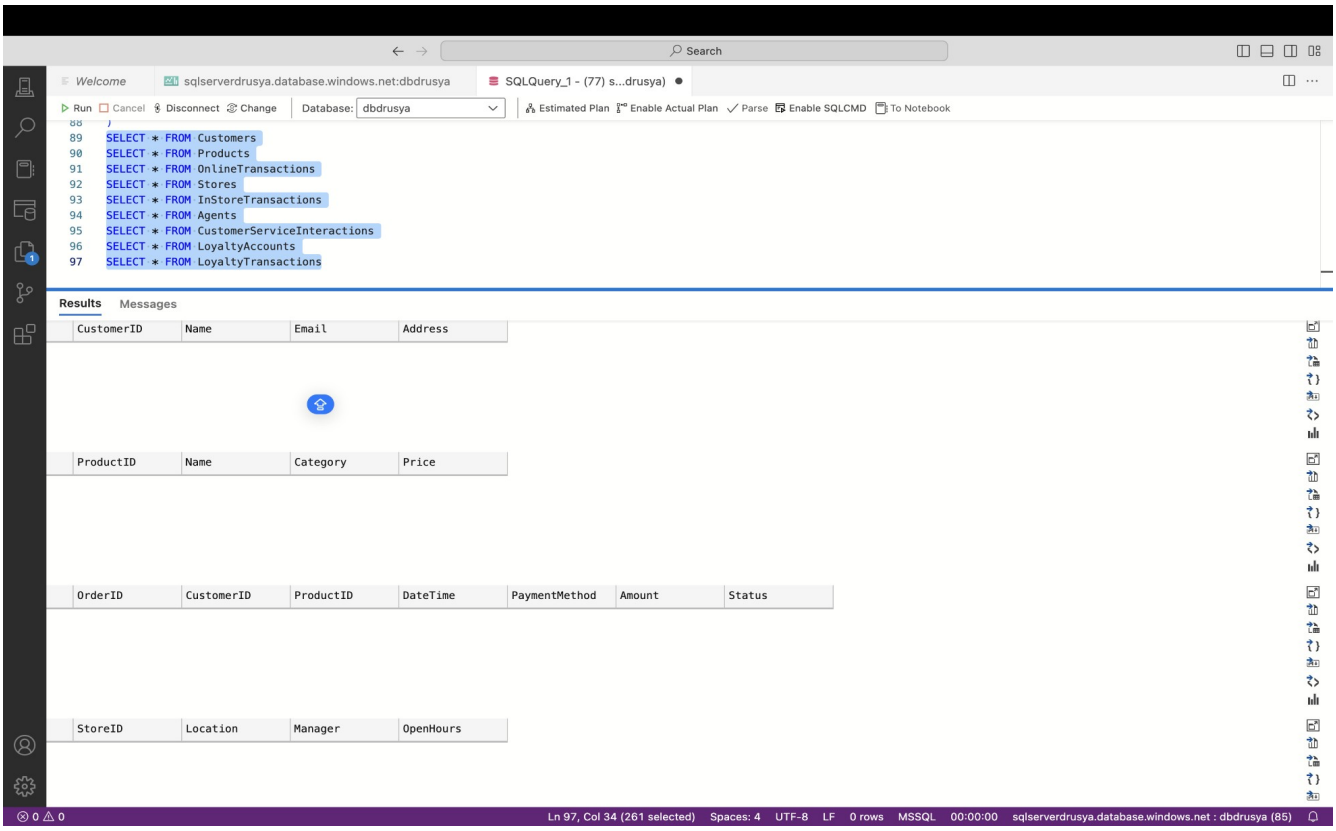
)

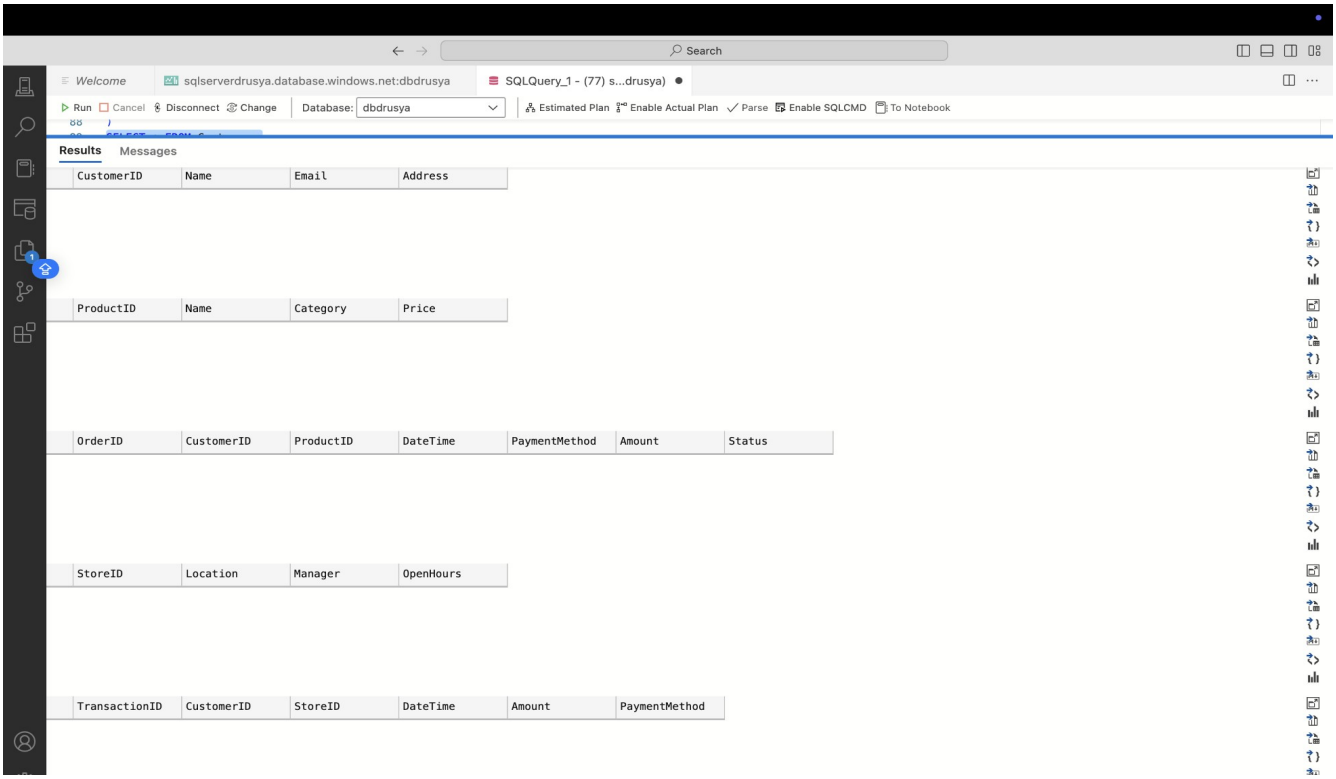
----Create table LoyaltyTransactions
CREATE TABLE LoyaltyTransactions
(
LoyaltyID INT,
DateTime DATETIME,
PointsChange INT,
Reason VARCHAR(100),
PRIMARY KEY (LoyaltyID,DateTime),
FOREIGN KEY (LoyaltyID) REFERENCES LoyaltyAccounts(LoyaltyID)
)




Search
Welcome
sqlserverdrusya.database.windows.net:dbdrusya
SQLQuery_1 - (77) s...drusya)
Run
Cancel
Disconnect
Change
Database: dbdrusya
Estimated Plan
Enable Actual Plan
Parse
Enable SQLCMD
To Notebook
)
-----Create table Stores
CREATE TABLE Stores
(
StoreID INT PRIMARY KEY,
Location VARCHAR(100),
Manager VARCHAR(100),
OpenHours VARCHAR(50)
)
-----Create table InStoreTransactions
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
CREATE TABLE Agents(
AgentID INT PRIMARY KEY,
Name VARCHAR(100),
Department VARCHAR(50),
Shift VARCHAR(50)
)
----Create table CustomerServiceInteractions
CREATE TABLE CustomerServiceInteractions
(
InteractionID INT PRIMARY KEY,
CustomerID INT,
DateTime DATETIME,
Messages
4:52:13 PM
Started executing query at Line 80
Commands completed successfully.
Total execution time: 00:00:00.045
Ln 88, Col 2
Spaces: 4
UTF-8
LF
0 rows
MSSQL
00:00:00
sqlserverdrusya.database.windows.net : dbdrusya (85)


Search
Welcome
sqlserverdrusya.database.windows.net:dbdrusya
SQLQuery_1 - (77) s...drusya)
Run
Cancel
Disconnect
Change
Database: dbdrusya
Estimated Plan
Enable Actual Plan
Parse
Enable SQLCMD
To Notebook
)
----Create table CustomerServiceInteractions
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
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
CREATE TABLE LoyaltyTransactions
(
LoyaltyID INT,
DateTime DATETIME,
PointsChange INT,
Reason VARCHAR(100),
PRIMARY KEY (LoyaltyID,DateTime),
FOREIGN KEY (LoyaltyID) REFERENCES LoyaltyAccounts(LoyaltyID)
)
Messages
4:52:13 PM
Started executing query at Line 80
Commands completed successfully.
Total execution time: 00:00:00.045
Ln 88, Col 2
Spaces: 4
UTF-8
LF
0 rows
MSSQL
00:00:00
sqlserverdrusya.database.windows.net : dbdrusya (85)


Search
Welcome
sqlserverdrusya.database.windows.net:dbdrusya
SQLQuery_1 - (77) s...drusya)
Run
Cancel
Disconnect
Change
Database:
dbdrusya
Estimated Plan
Enable Actual Plan
Parse
Enable SQLCMD
To Notebook
SELECT * FROM Customers
SELECT * FROM Products
SELECT * FROM OnlineTransactions
SELECT * FROM Stores
SELECT * FROM InStoreTransactions
SELECT * FROM Agents
SELECT * FROM CustomerServiceInteractions
SELECT * FROM LoyaltyAccounts
SELECT * FROM LoyaltyTransactions
Results
Messages
CustomerID
Name
Email
Address
ProductID
Name
Category
Price
OrderID
CustomerID
ProductID
DateTime
PaymentMethod
Amount
Status
StoreID
Location
Manager
OpenHours
Ln 97, Col 34 (261 selected)
Spaces: 4
UTF-8
LF
0 rows
MSSQL
00:00:00
sqlserverdrusya.database.windows.net : dbdrusya (85)


Search
Welcome
sqlserverdrusya.database.windows.net:dbdrusya
SQLQuery_1 - (77) s...drusya)
Run
Cancel
Disconnect
Change
Database:
dbdrusya
Estimated Plan
Enable Actual Plan
Parse
Enable SQLCMD
To Notebook
Results
Messages
CustomerID
Name
Email
Address
ProductID
Name
Category
Price
OrderID
CustomerID
ProductID
DateTime
PaymentMethod
Amount
Status
StoreID
Location
Manager
OpenHours
TransactionID
CustomerID
StoreID
DateTime
Amount
PaymentMethod

STEP 3

Created a pipeline in Azure synapse analytics and using the data flow activity I cleaned the data in the files using the following transformations.

Derived column: This transformation will:

- **Trim** whitespace from all text columns
- **Standardize casing** for `Department` and `Shift`

Filter transformation: Ensure only **valid rows** are passed downstream by removing those with:

- `null` values (i.e., missing fields)
- empty strings (`""`)
- whitespace-only values (like `"  "`)

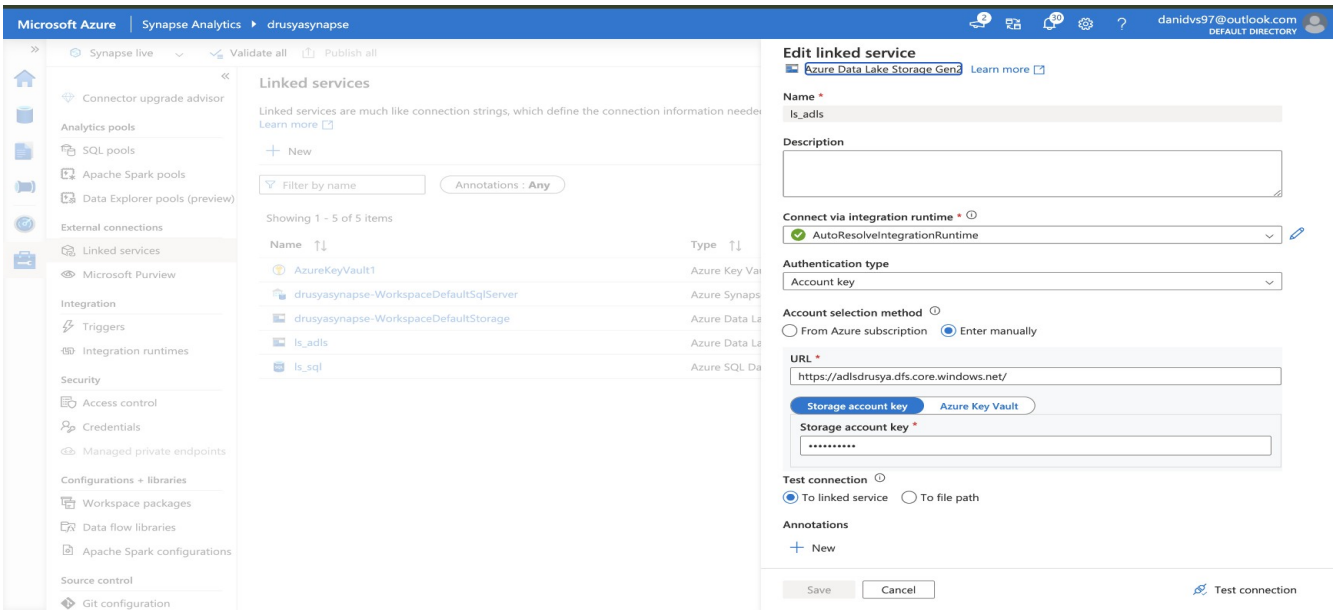
Aggregate transformation: To remove the duplicates in the dataset.

For the data flows my source is the raw file in the bronze layer and sink is the SQL db.Since I have primary key and foreign key complications I used two separate dataflows inside my pipeline and connected them.

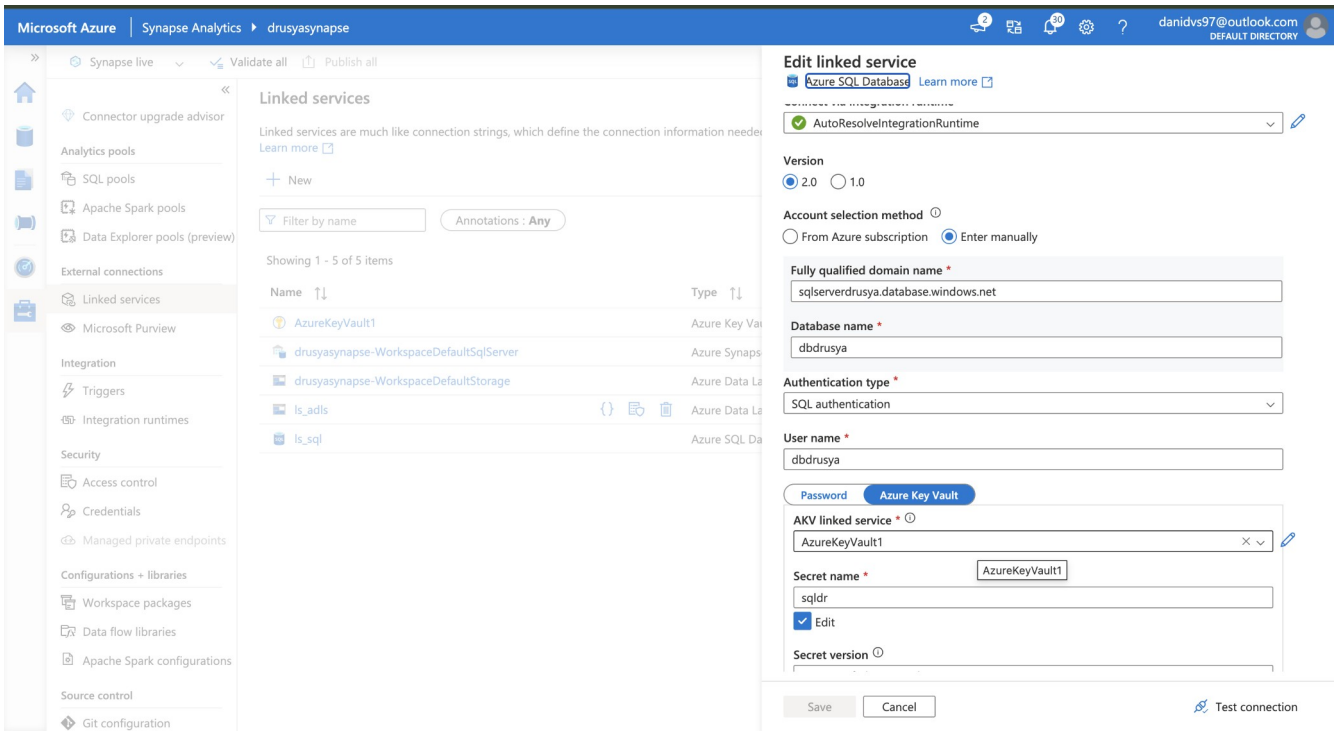
Alter row transformation: The **Alter Row transformation** in Azure Data Flow is used to **control row-level operations** (Insert, Update, Delete, Upsert) when writing to **sink destinations** like SQL tables.

Linked services:

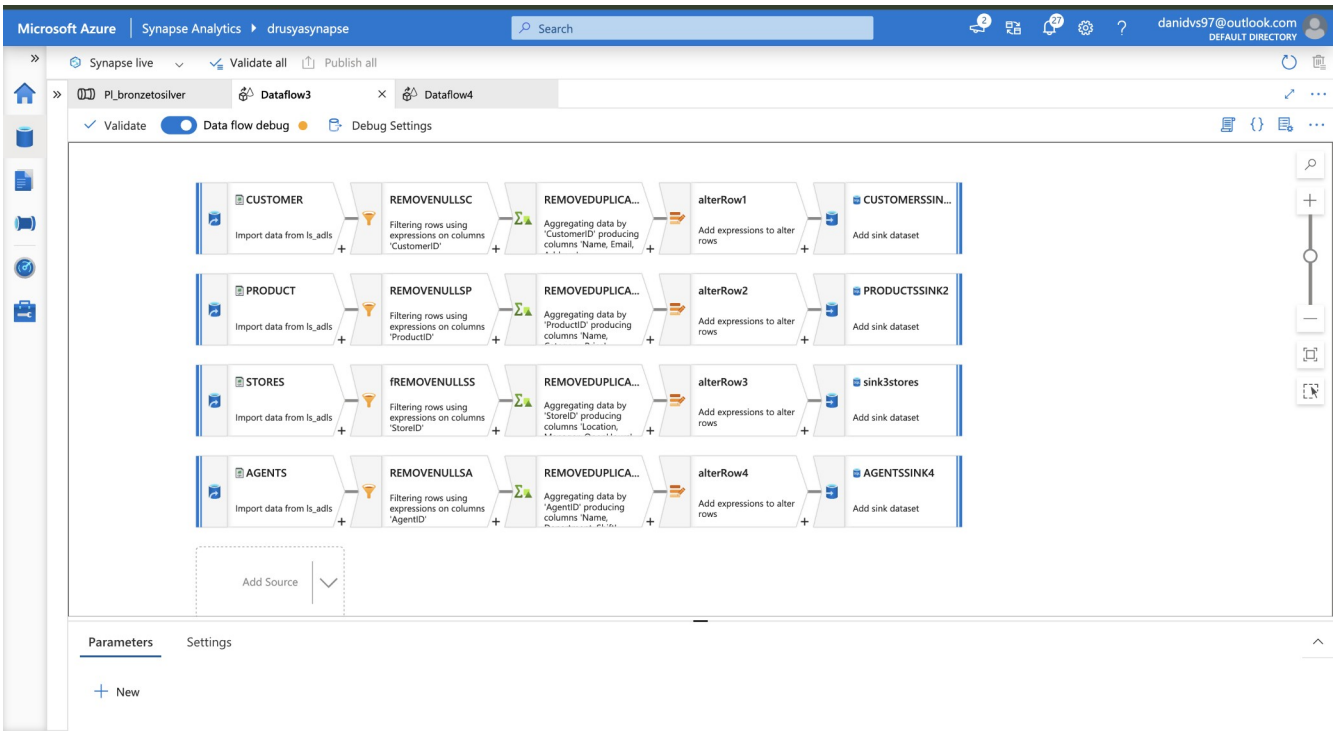
For ADLS storage account:



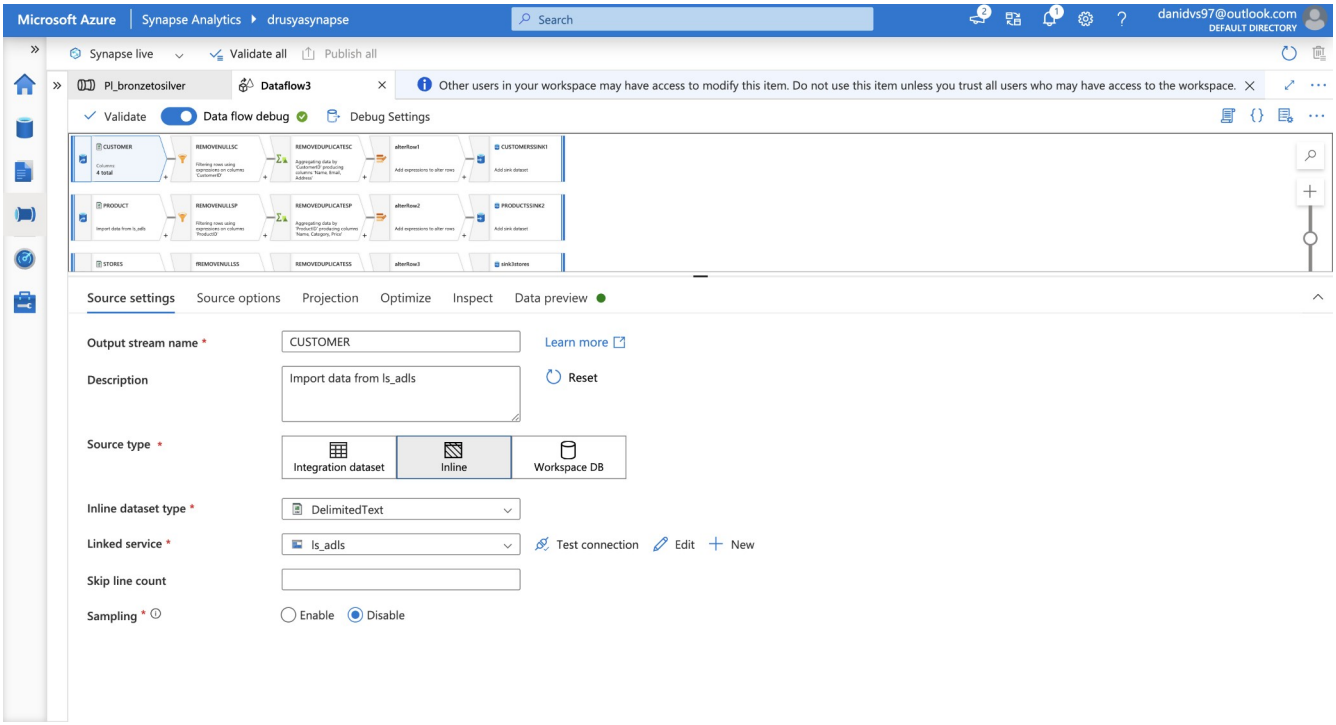
For SQL server used Azure key vault integration.

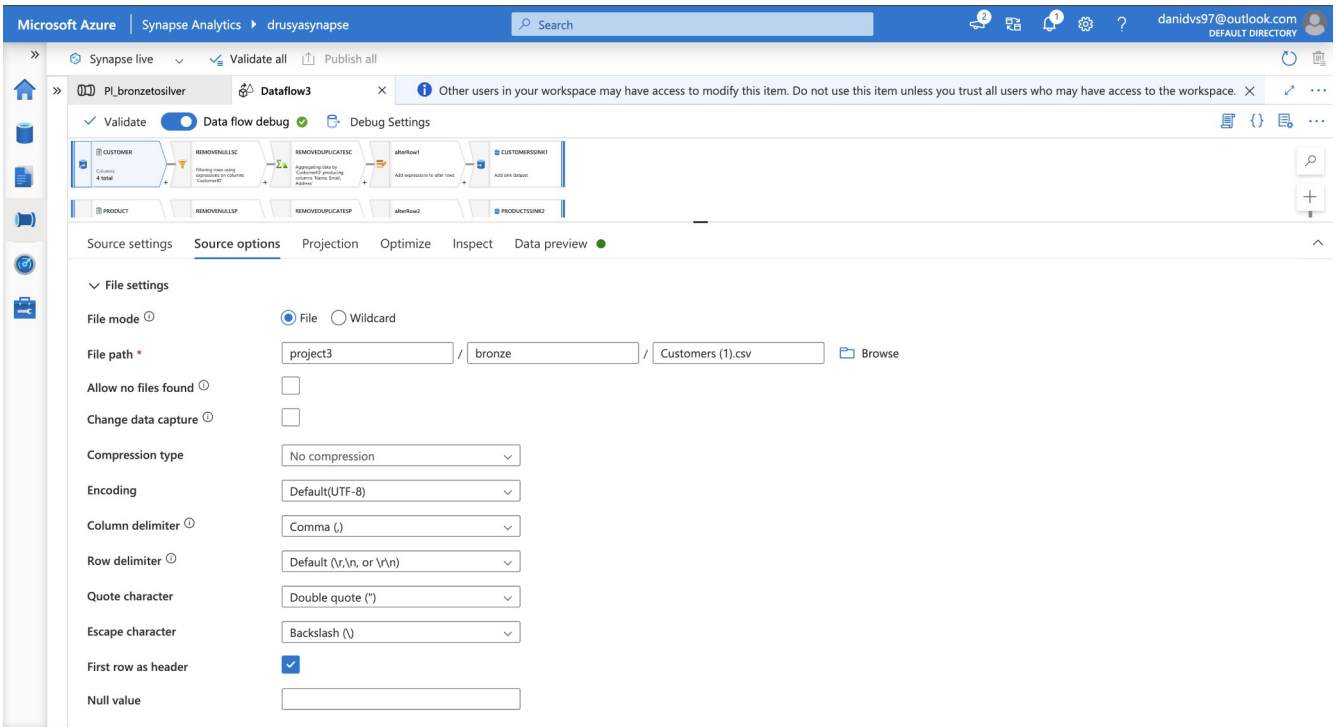


Data flow 1: For tables with primary keys.

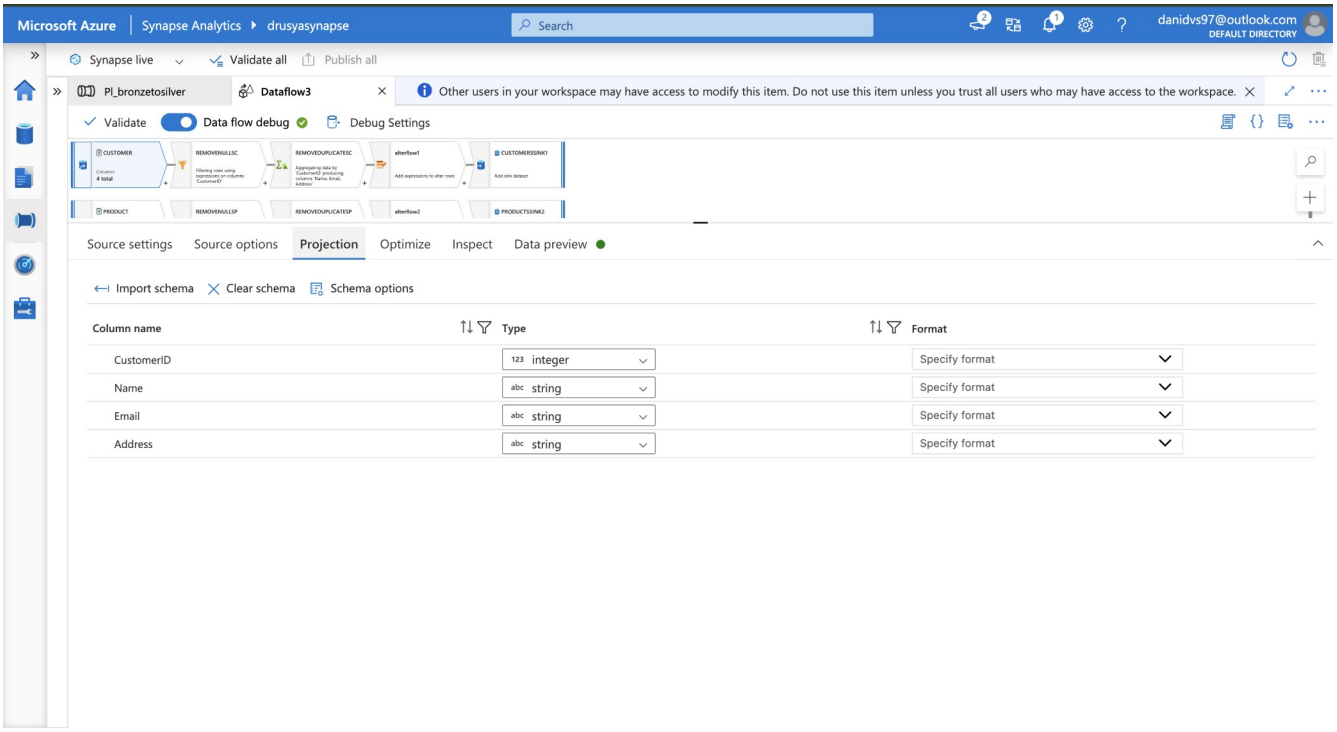


Source: Source is my csv file in Bronze container in the adls. I selected the inline option as source type and delimited text as dataset type. In the settings page I gave my source file path and checked the first row as header option.

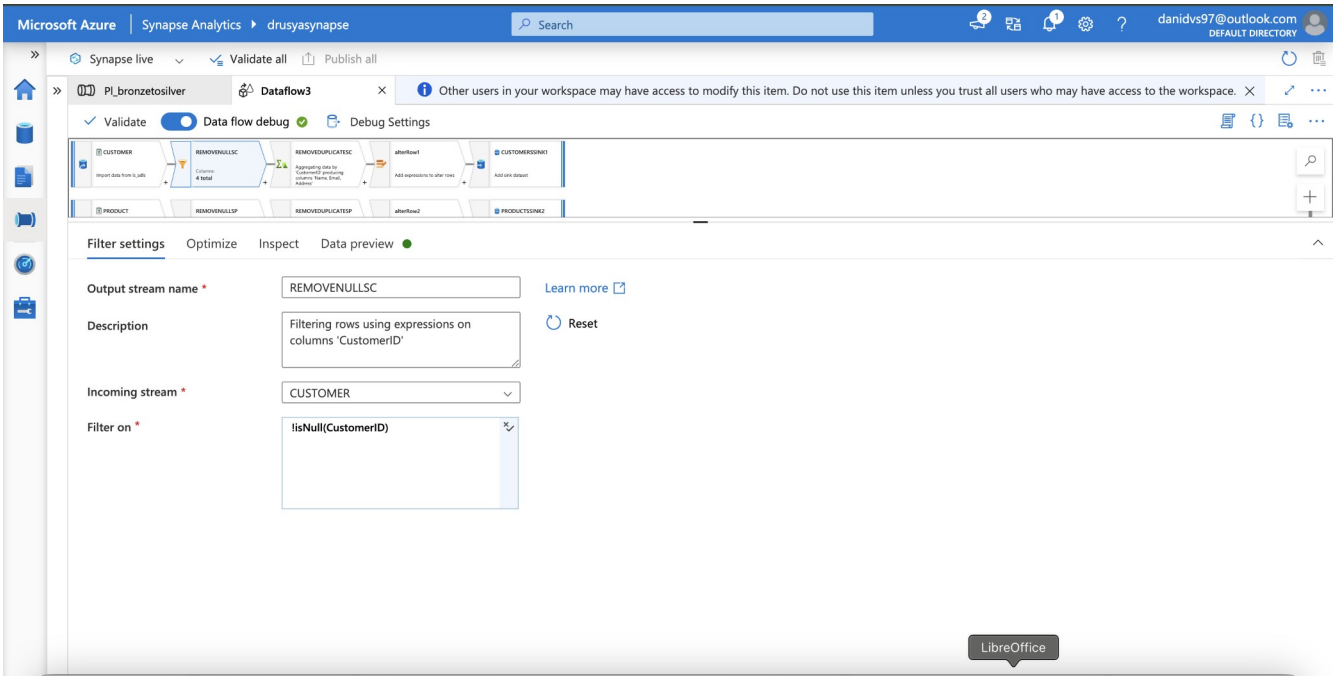




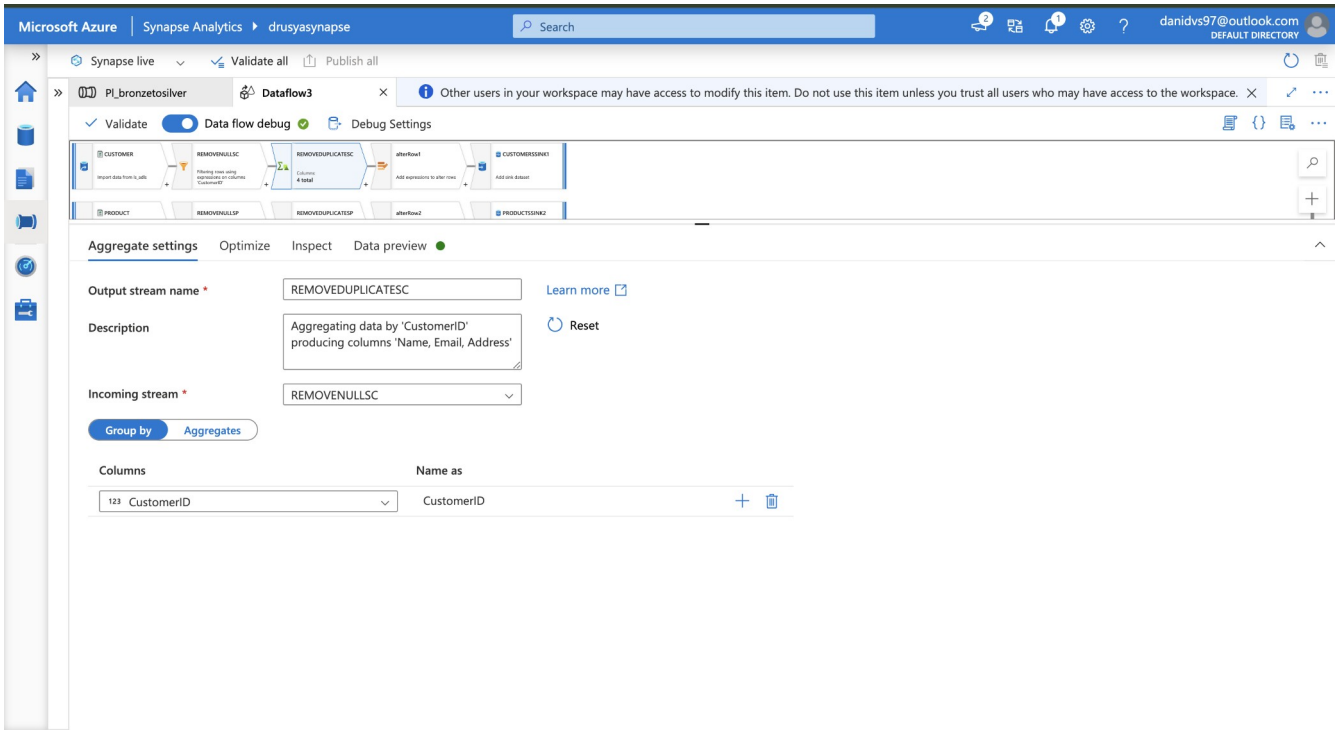
Now in the projection selected the import schema option.



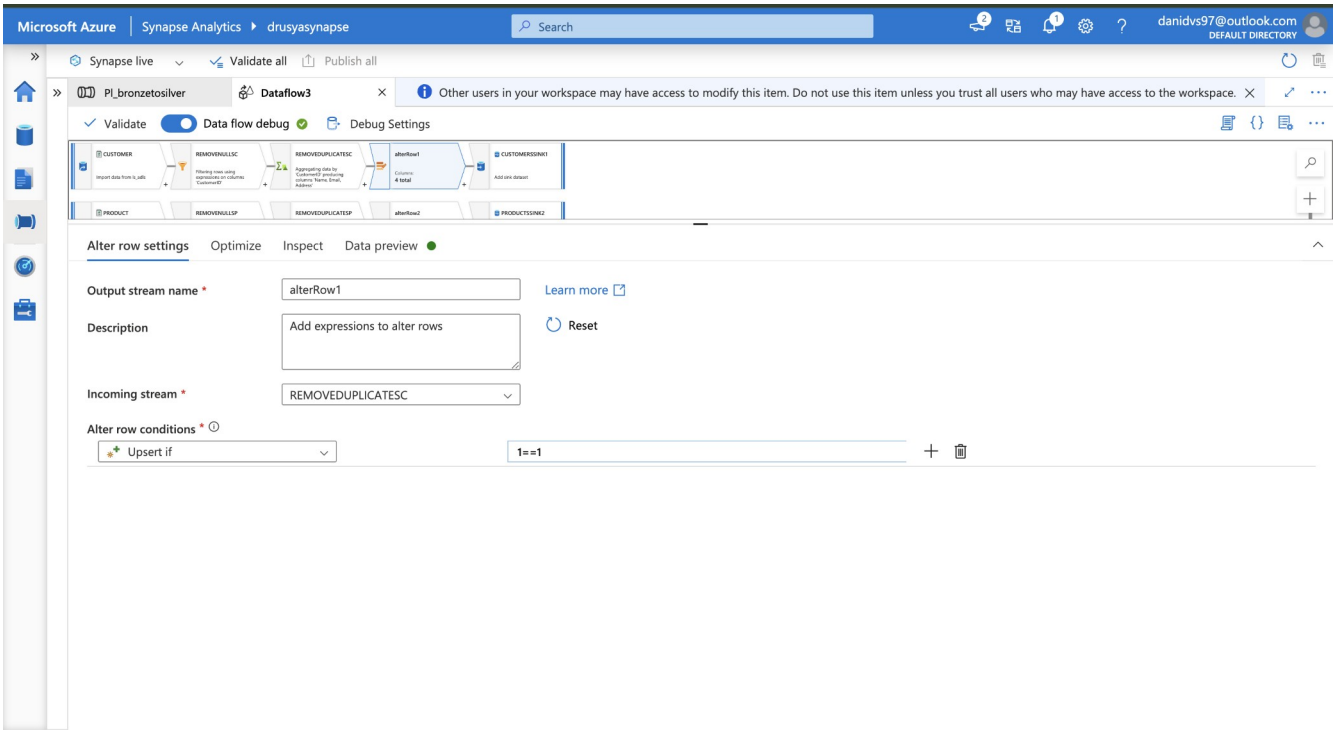
Next added a filter transformation to remove the nulls.



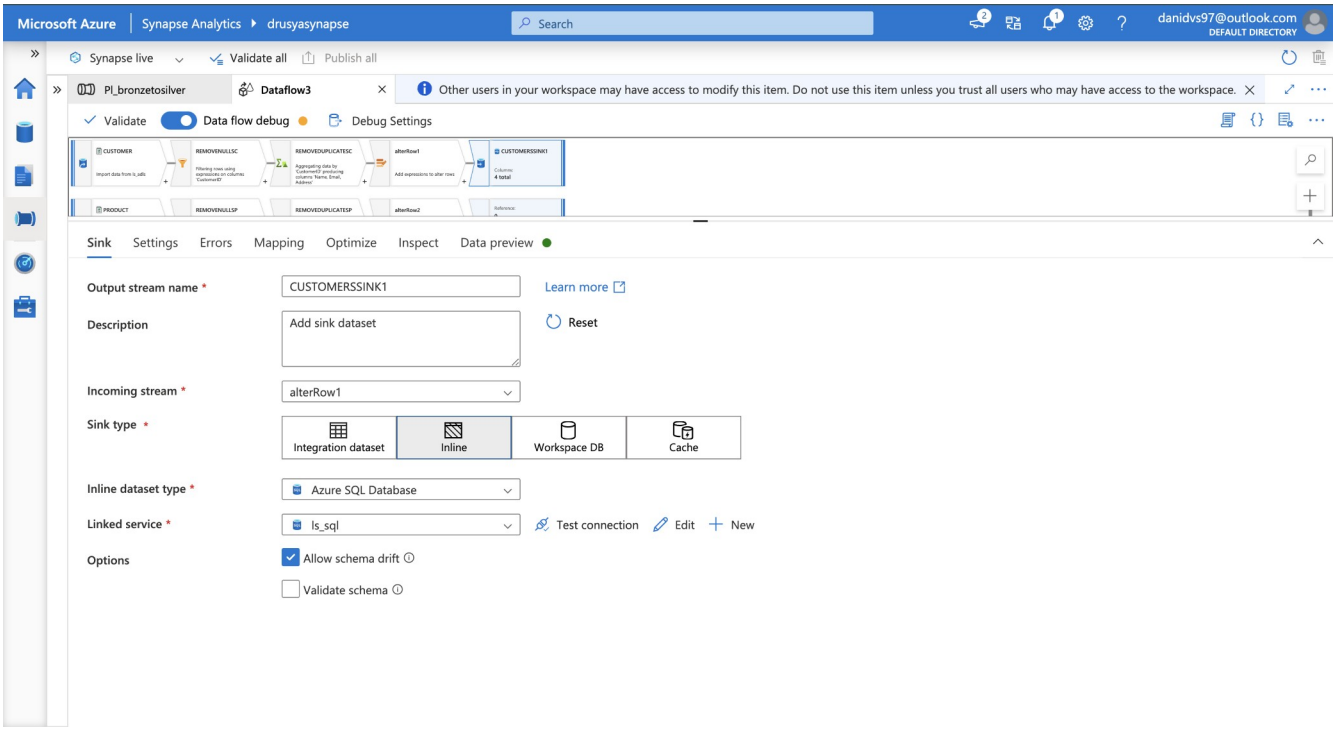
Next added aggregate transformation to remove duplicates.

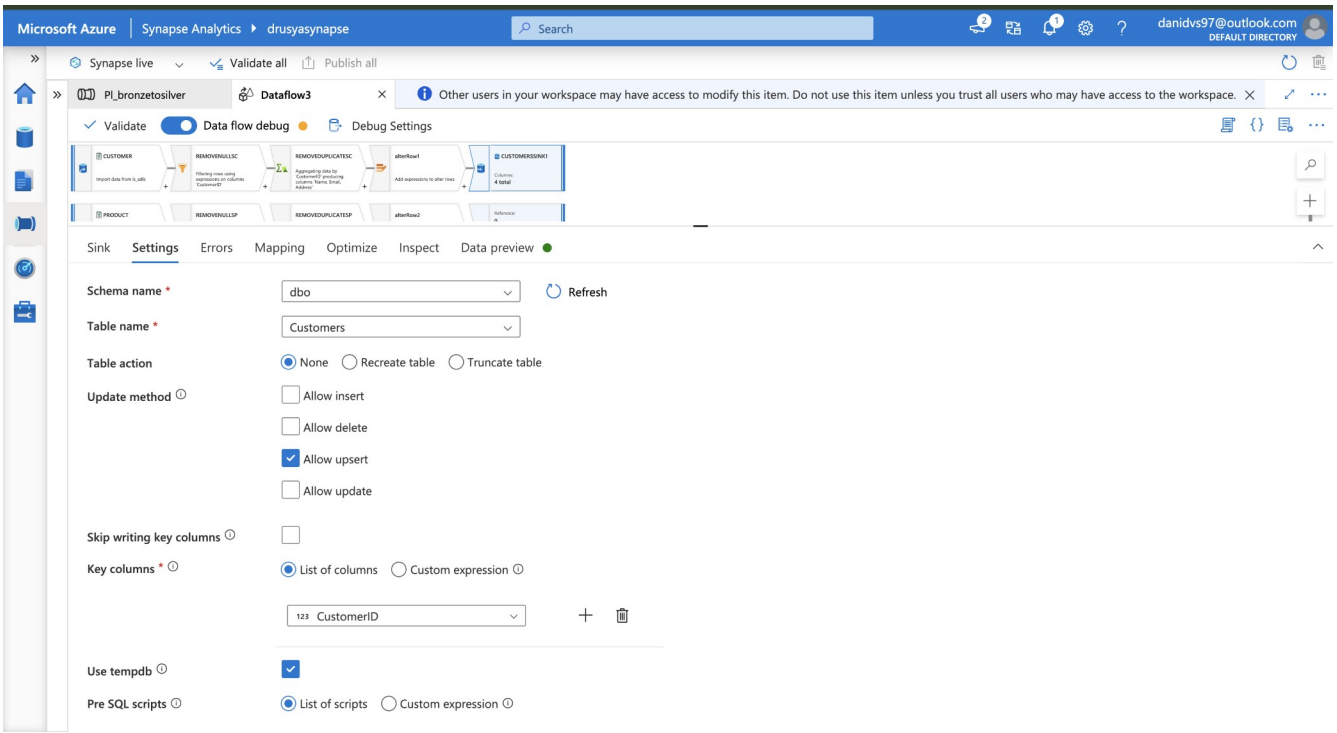


Next added alter row transformation.



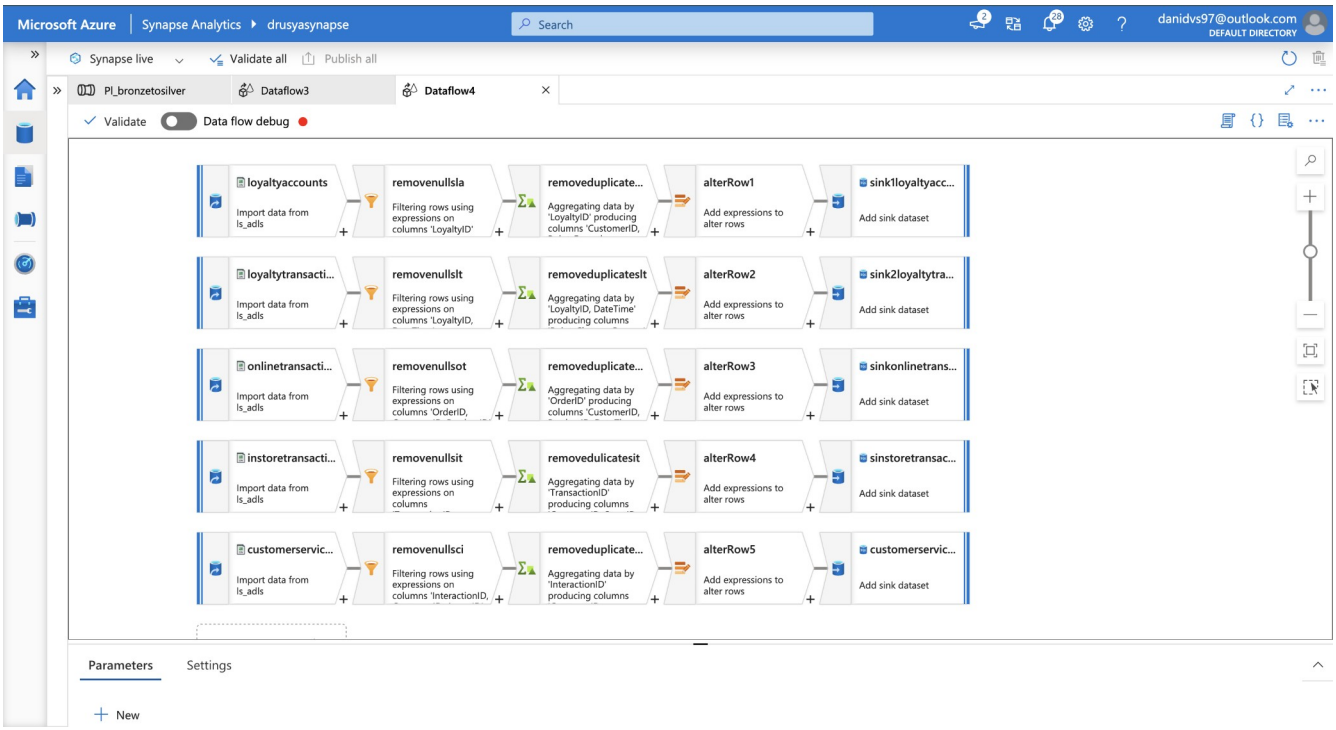
Now added the sink as the SQL table through inline option and gave the key column as the primary key. Also selected the update method as upsert.



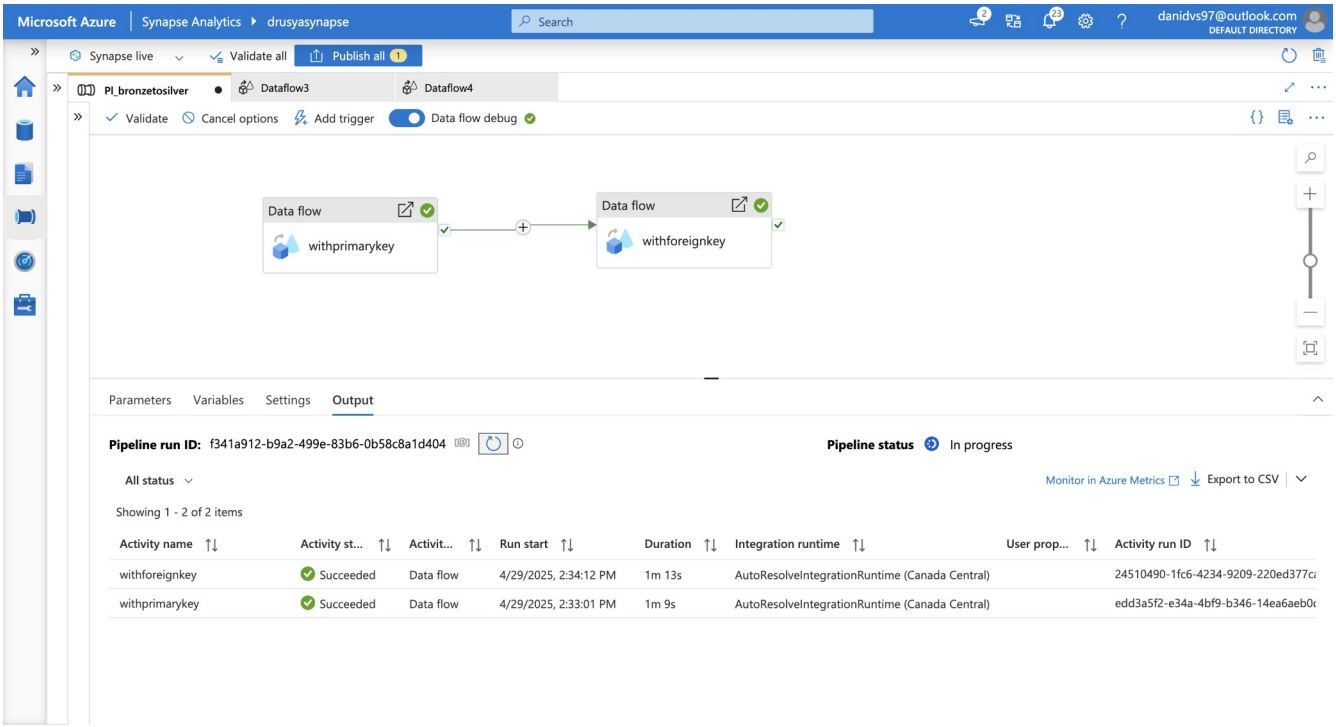


Repeated the same step for all the other files.

Data flow 2: For inserting into tables with foreign keys. In this transformations I gave the primary keys as key columns in the sink.

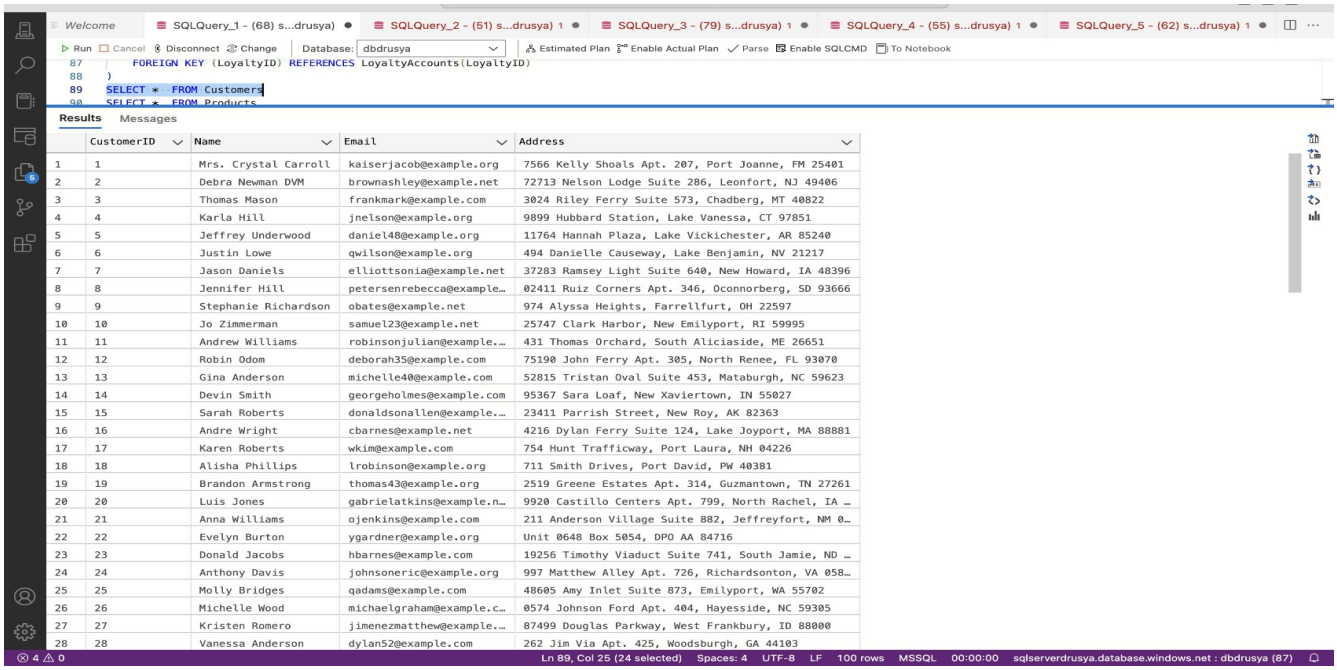


Now I successfully run the pipeline.

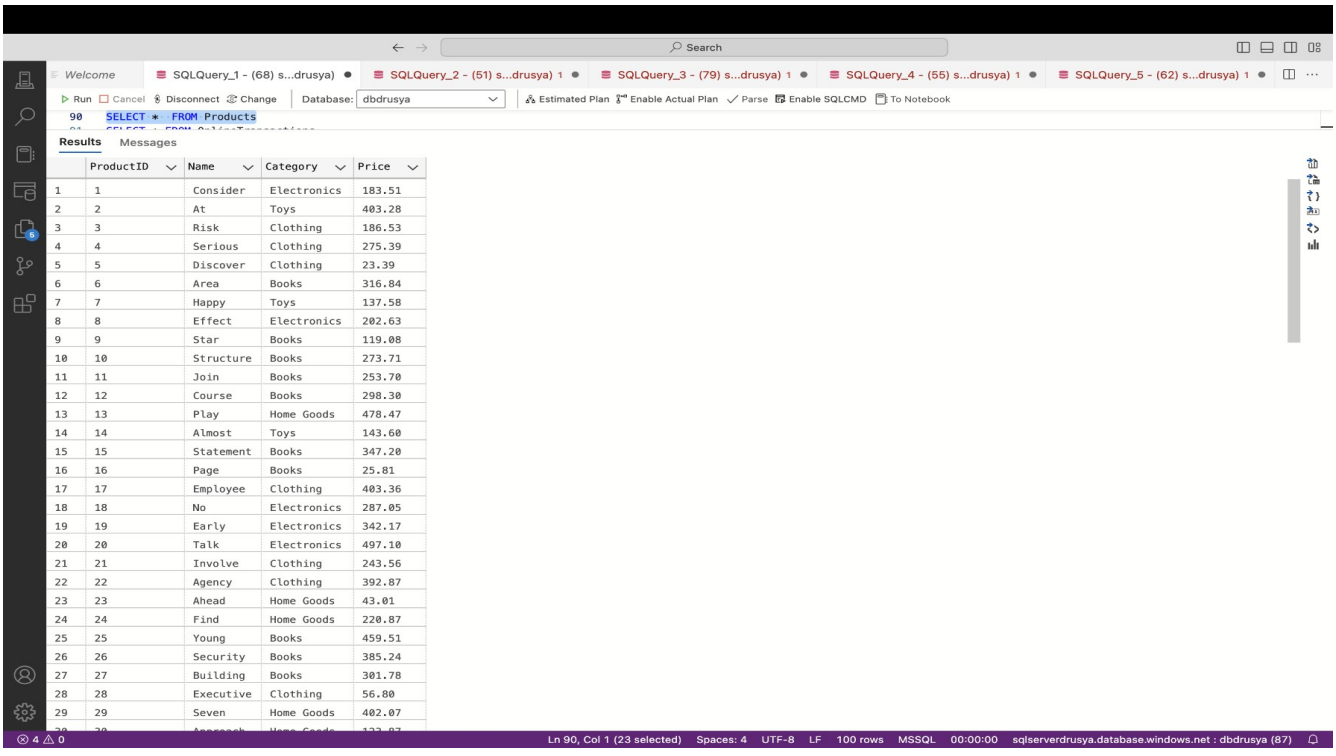


Outputs in data studio are as follows:

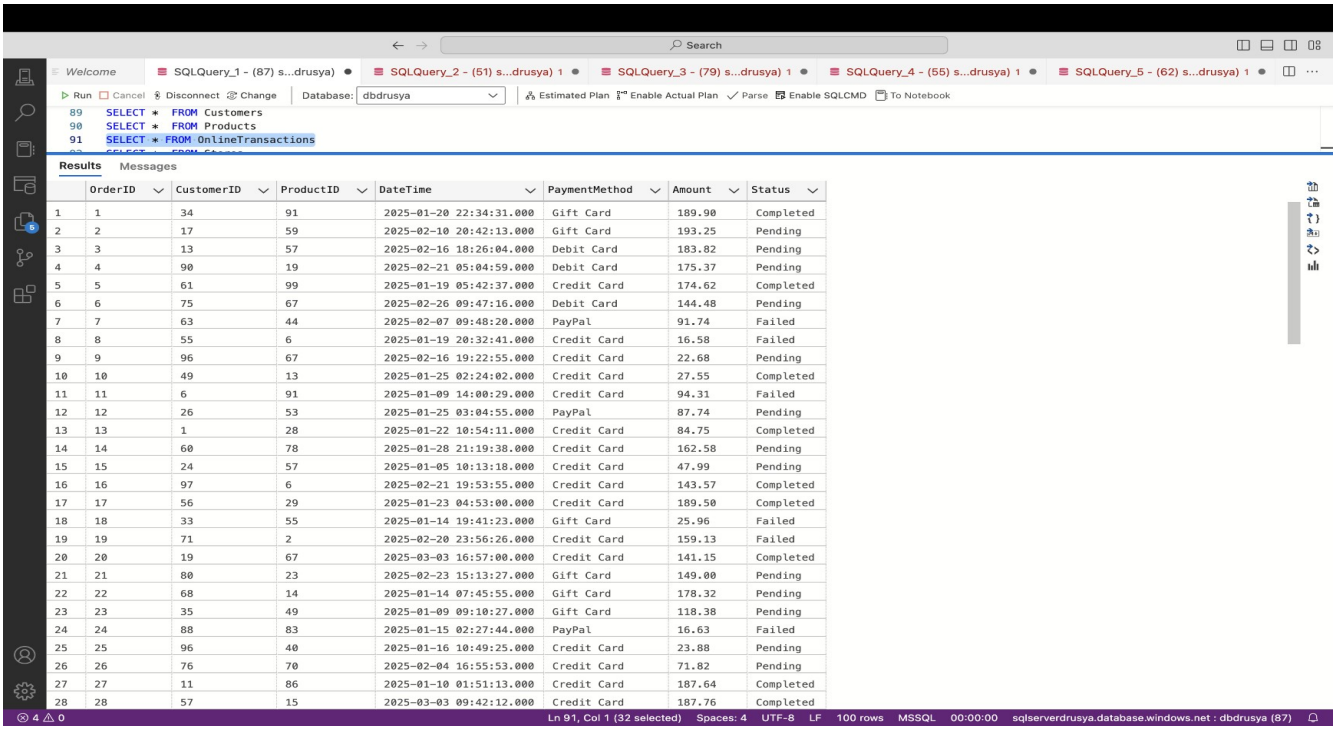
Customers table:



Products table:



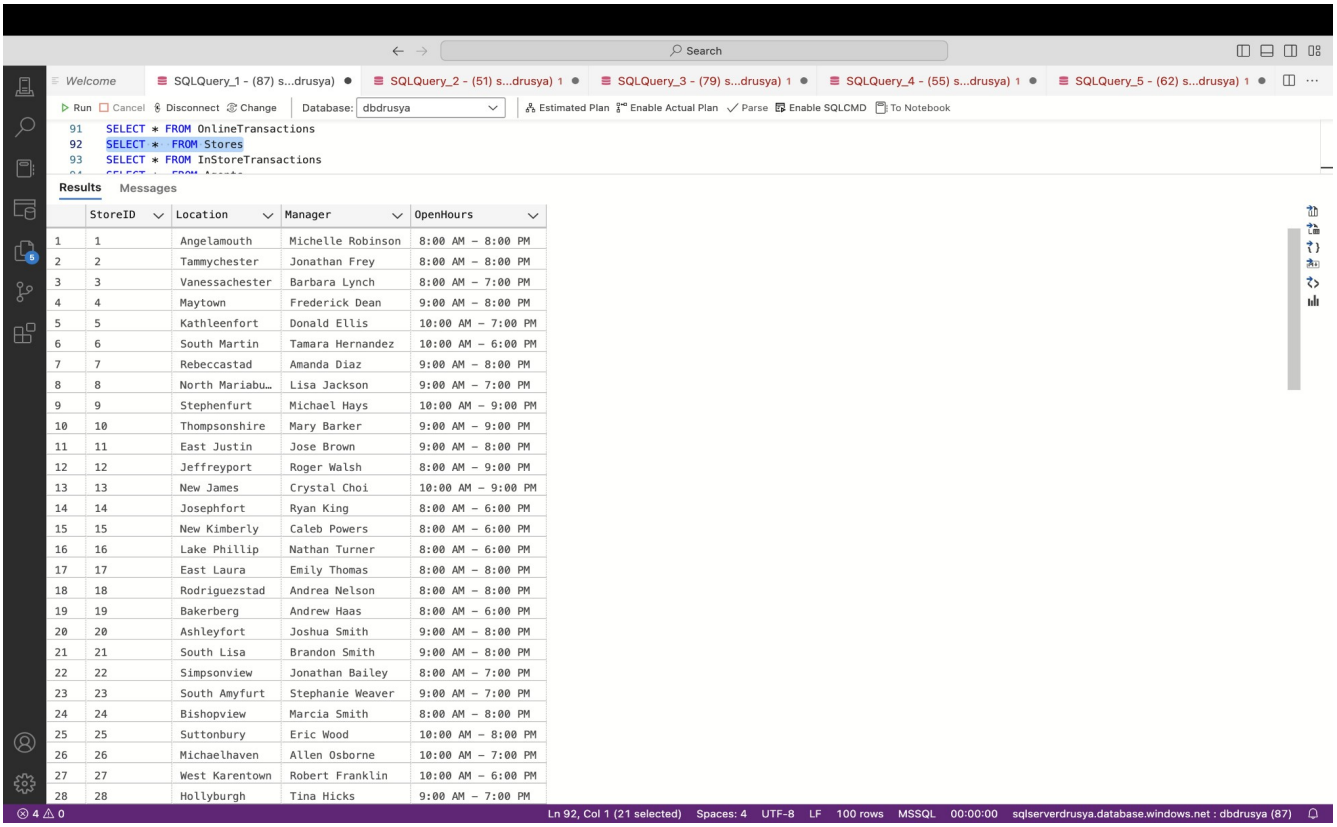
Online transactions table:



```
SELECT * FROM Customers
SELECT * FROM Products
SELECT * FROM OnlineTransactions
```

OrderID	CustomerID	ProductID	DateTime	PaymentMethod	Amount	Status
1	34	91	2025-01-20 22:34:31.000	Gift Card	189.90	Completed
2	17	59	2025-02-10 20:42:13.000	Gift Card	193.25	Pending
3	13	57	2025-02-16 18:26:04.000	Debit Card	183.82	Pending
4	90	19	2025-02-21 05:04:59.000	Debit Card	175.37	Pending
5	61	99	2025-01-19 05:42:37.000	Credit Card	174.62	Completed
6	75	67	2025-02-26 09:47:16.000	Debit Card	144.48	Pending
7	63	44	2025-02-07 09:48:20.000	PayPal	91.74	Failed
8	55	6	2025-01-19 20:32:41.000	Credit Card	16.58	Failed
9	96	67	2025-02-16 19:22:55.000	Credit Card	22.68	Pending
10	49	13	2025-01-25 02:24:02.000	Credit Card	27.55	Completed
11	6	91	2025-01-09 14:00:29.000	Credit Card	94.31	Failed
12	26	53	2025-01-25 03:04:55.000	PayPal	87.74	Pending
13	1	28	2025-01-22 10:54:11.000	Credit Card	84.75	Completed
14	60	78	2025-01-28 21:19:38.000	Credit Card	162.58	Pending
15	24	57	2025-01-05 10:13:18.000	Credit Card	47.99	Pending
16	97	6	2025-02-21 19:53:55.000	Credit Card	143.57	Completed
17	56	29	2025-01-23 04:53:00.000	Credit Card	189.50	Completed
18	33	55	2025-01-14 19:41:23.000	Gift Card	25.96	Failed
19	71	2	2025-02-20 23:56:26.000	Credit Card	159.13	Failed
20	19	67	2025-03-03 16:57:00.000	Credit Card	141.15	Completed
21	80	23	2025-02-23 15:13:27.000	Gift Card	149.00	Pending
22	68	14	2025-01-14 07:45:55.000	Gift Card	178.32	Pending
23	35	49	2025-01-09 09:10:27.000	Gift Card	118.38	Pending
24	88	83	2025-01-15 02:27:44.000	PayPal	16.63	Failed
25	96	40	2025-01-16 10:49:25.000	Credit Card	23.88	Pending
26	76	70	2025-02-04 16:55:53.000	Credit Card	71.82	Pending
27	11	86	2025-01-10 01:51:13.000	Credit Card	187.64	Completed
28	57	15	2025-03-03 09:42:12.000	Credit Card	187.76	Completed

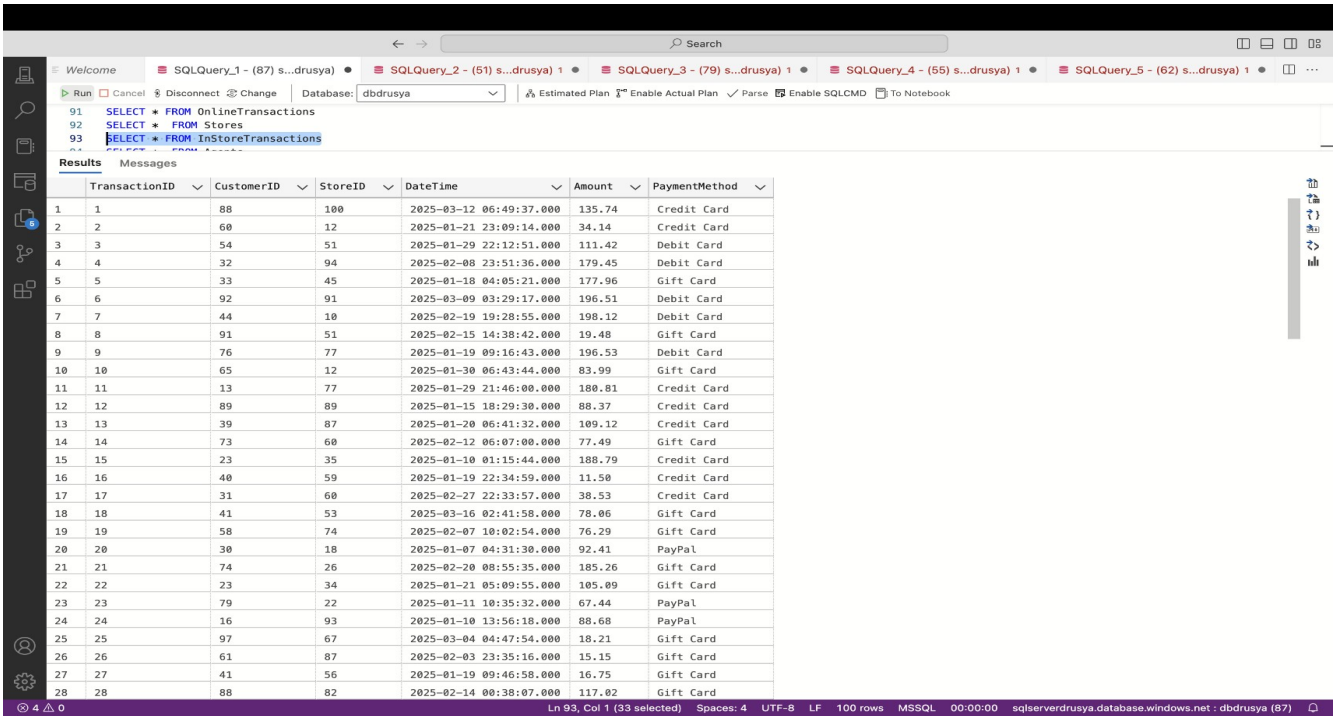
Stores table:



```
SELECT * FROM OnlineTransactions
SELECT * FROM Stores
SELECT * FROM InStoreTransactions
```

StoreID	Location	Manager	OpenHours
1	Angelamouth	Michelle Robinson	8:00 AM – 8:00 PM
2	Tammychester	Jonathan Frey	8:00 AM – 8:00 PM
3	Vanessachester	Barbara Lynch	8:00 AM – 7:00 PM
4	Maytown	Frederick Dean	9:00 AM – 8:00 PM
5	Kathleenfort	Donald Ellis	10:00 AM – 7:00 PM
6	South Martin	Tamara Hernandez	10:00 AM – 6:00 PM
7	Rebeccastad	Amanda Diaz	9:00 AM – 8:00 PM
8	North Mariabu…	Lisa Jackson	9:00 AM – 7:00 PM
9	Stephenfurt	Michael Hays	10:00 AM – 9:00 PM
10	Thompsonshire	Mary Barker	9:00 AM – 9:00 PM
11	East Justin	Jose Brown	9:00 AM – 8:00 PM
12	Jeffreyport	Roger Walsh	8:00 AM – 9:00 PM
13	New James	Crystal Choi	10:00 AM – 9:00 PM
14	Josephfort	Ryan King	8:00 AM – 6:00 PM
15	New Kimberly	Caleb Powers	8:00 AM – 6:00 PM
16	Lake Phillip	Nathan Turner	8:00 AM – 6:00 PM
17	East Laura	Emily Thomas	8:00 AM – 8:00 PM
18	Rodriguezstad	Andrea Nelson	8:00 AM – 8:00 PM
19	Bakerberg	Andrew Haas	8:00 AM – 6:00 PM
20	Ashleyfort	Joshua Smith	9:00 AM – 8:00 PM
21	South Lisa	Brandon Smith	9:00 AM – 8:00 PM
22	Simpsonview	Jonathan Bailey	8:00 AM – 7:00 PM
23	South Amyfurt	Stephanie Weaver	9:00 AM – 7:00 PM
24	Bishopview	Marcia Smith	8:00 AM – 8:00 PM
25	Suttonbury	Eric Wood	10:00 AM – 8:00 PM
26	Michaelhaven	Allen Osborne	10:00 AM – 7:00 PM
27	West Karentown	Robert Franklin	10:00 AM – 6:00 PM
28	Hollyburgh	Tina Hicks	9:00 AM – 7:00 PM

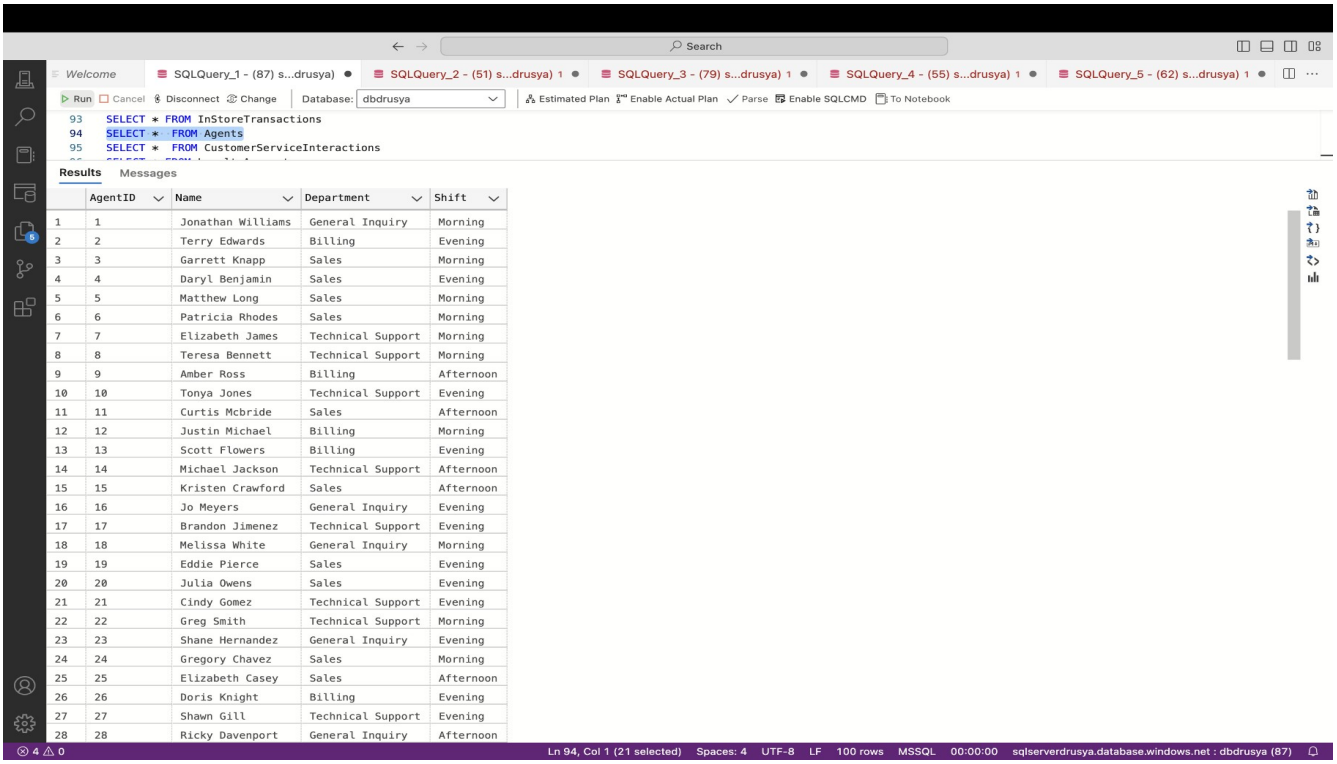
Instore transactions table:



```
SELECT * FROM OnlineTransactions
SELECT *  FROM Stores
SELECT * FROM InStoreTransactions
```

	TransactionID	CustomerID	StoreID	DateTime	Amount	PaymentMethod
1	1	88	100	2025-03-12 06:49:37.000	135.74	Credit Card
2	2	60	12	2025-01-21 23:09:14.000	34.14	Credit Card
3	3	54	51	2025-01-29 22:12:51.000	111.42	Debit Card
4	4	32	94	2025-02-08 23:51:36.000	179.45	Debit Card
5	5	33	45	2025-01-18 04:05:21.000	177.96	Gift Card
6	6	92	91	2025-03-09 03:29:17.000	196.51	Debit Card
7	7	44	10	2025-02-19 19:28:55.000	198.12	Debit Card
8	8	91	51	2025-02-15 14:38:42.000	19.48	Gift Card
9	9	76	77	2025-01-19 09:16:43.000	196.53	Debit Card
10	10	65	12	2025-01-30 06:43:44.000	83.99	Gift Card
11	11	13	77	2025-01-29 21:46:00.000	180.81	Credit Card
12	12	89	89	2025-01-15 18:29:30.000	88.37	Credit Card
13	13	39	87	2025-01-20 06:41:32.000	109.12	Credit Card
14	14	73	60	2025-02-12 06:07:00.000	77.49	Gift Card
15	15	23	35	2025-01-10 01:15:44.000	188.79	Credit Card
16	16	40	59	2025-01-19 22:34:59.000	11.50	Credit Card
17	17	31	60	2025-02-27 22:33:57.000	38.53	Credit Card
18	18	41	53	2025-03-16 02:41:58.000	78.06	Gift Card
19	19	58	74	2025-02-07 10:02:54.000	76.29	Gift Card
20	20	30	18	2025-01-07 04:31:30.000	92.41	PayPal
21	21	74	26	2025-02-20 08:55:35.000	185.26	Gift Card
22	22	23	34	2025-01-21 05:09:55.000	105.09	Gift Card
23	23	79	22	2025-01-11 10:35:32.000	67.44	PayPal
24	24	16	93	2025-01-10 13:56:18.000	88.68	PayPal
25	25	97	67	2025-03-04 04:47:54.000	18.21	Gift Card
26	26	61	87	2025-02-03 23:35:16.000	15.15	Gift Card
27	27	41	56	2025-01-19 09:46:58.000	16.75	Gift Card
28	28	88	82	2025-02-14 00:38:07.000	117.02	Gift Card

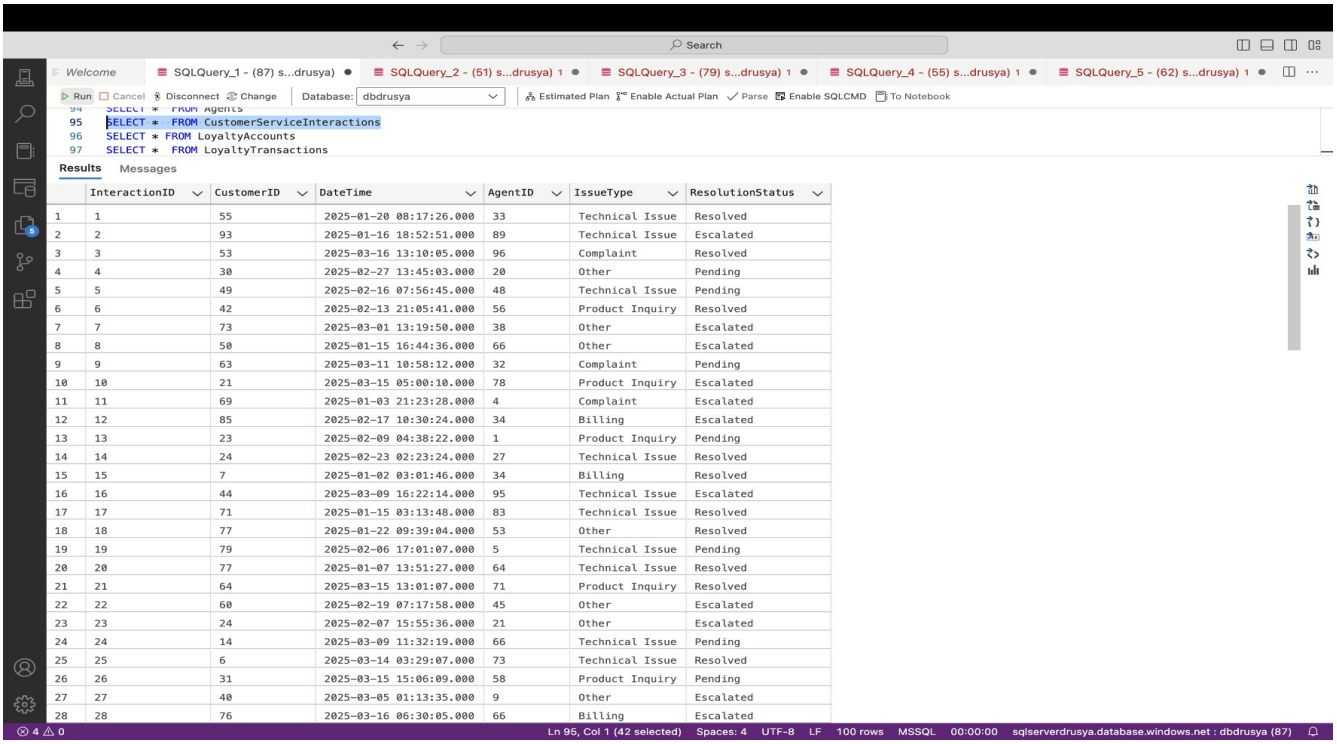
Agents table:



```
SELECT * FROM InStoreTransactions
SELECT *  FROM Agents
SELECT *  FROM CustomerServiceInteractions
```

	AgentID	Name	Department	Shift
1	1	Jonathan Williams	General Inquiry	Morning
2	2	Terry Edwards	Billing	Evening
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 Support	Morning
8	8	Teresa Bennett	Technical Support	Morning
9	9	Amber Ross	Billing	Afternoon
10	10	Tonya Jones	Technical Support	Evening
11	11	Curtis Mcbride	Sales	Afternoon
12	12	Justin Michael	Billing	Morning
13	13	Scott Flowers	Billing	Evening
14	14	Michael Jackson	Technical Support	Afternoon
15	15	Kristen Crawford	Sales	Afternoon
16	16	Jo Meyers	General Inquiry	Evening
17	17	Brandon Jimenez	Technical Support	Evening
18	18	Melissa White	General Inquiry	Morning
19	19	Eddie Pierce	Sales	Evening
20	20	Julia Owens	Sales	Evening
21	21	Cindy Gomez	Technical Support	Evening
22	22	Greg Smith	Technical Support	Morning
23	23	Shane Hernandez	General Inquiry	Evening
24	24	Gregory Chavez	Sales	Morning
25	25	Elizabeth Casey	Sales	Afternoon
26	26	Doris Knight	Billing	Evening
27	27	Shawn Gill	Technical Support	Evening
28	28	Ricky Davenport	General Inquiry	Afternoon

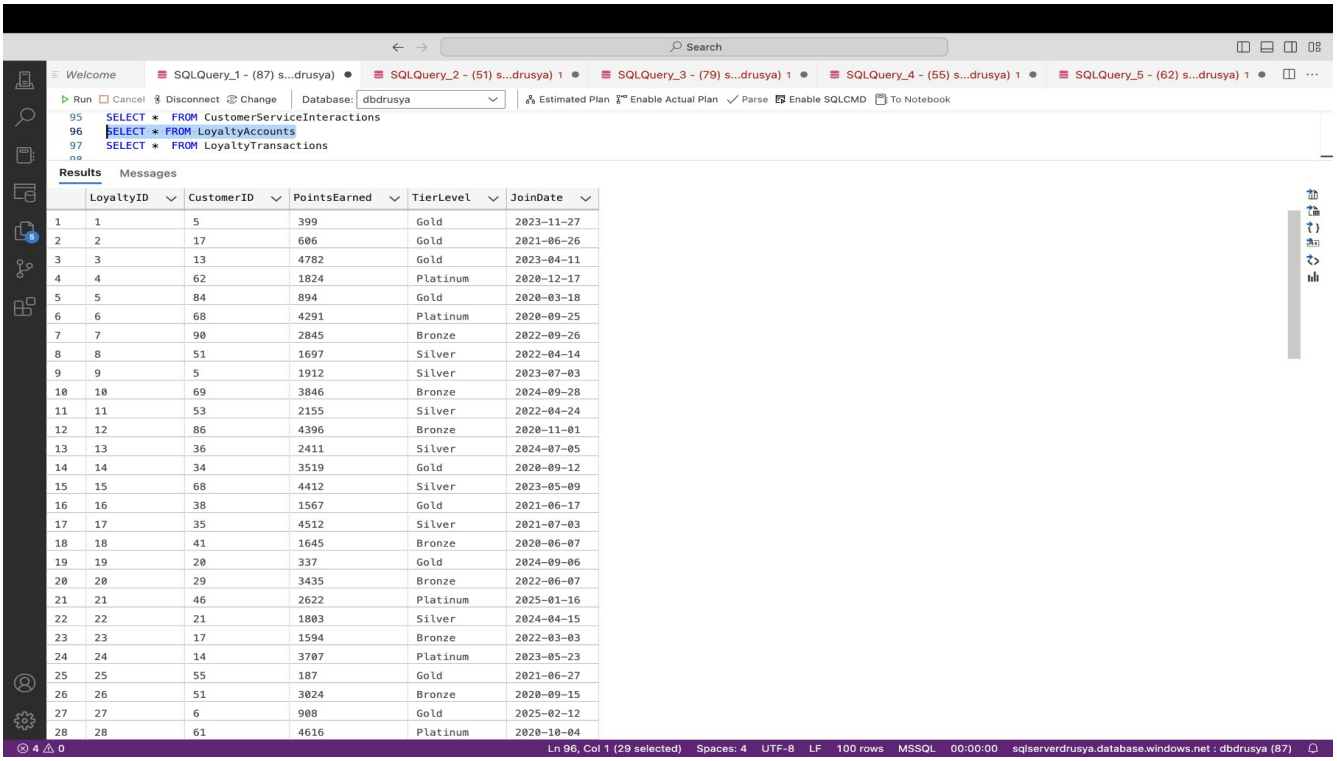
Customer service interactions table:



```
SELECT * FROM CustomerServiceInteractions
SELECT * FROM LoyaltyAccounts
SELECT * FROM LoyaltyTransactions
```

	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
13	13	23	2025-02-09 04:38:22.000	1	Product Inquiry	Pending
14	14	24	2025-02-23 02:23:24.000	27	Technical Issue	Resolved
15	15	7	2025-01-02 03:01:46.000	34	Billing	Resolved
16	16	44	2025-03-09 16:22:14.000	95	Technical Issue	Escalated
17	17	71	2025-01-15 03:13:48.000	83	Technical Issue	Resolved
18	18	77	2025-01-22 09:39:04.000	53	Other	Resolved
19	19	79	2025-02-06 17:01:07.000	5	Technical Issue	Pending
20	20	77	2025-01-07 13:51:27.000	64	Technical Issue	Resolved
21	21	64	2025-03-15 13:01:07.000	71	Product Inquiry	Resolved
22	22	60	2025-02-19 07:17:58.000	45	Other	Escalated
23	23	24	2025-02-07 15:55:36.000	21	Other	Escalated
24	24	14	2025-03-09 11:32:19.000	66	Technical Issue	Pending
25	25	6	2025-03-14 03:29:07.000	73	Technical Issue	Resolved
26	26	31	2025-03-15 15:06:09.000	58	Product Inquiry	Pending
27	27	40	2025-03-05 01:13:35.000	9	Other	Escalated
28	28	76	2025-03-16 06:30:05.000	66	Billing	Escalated

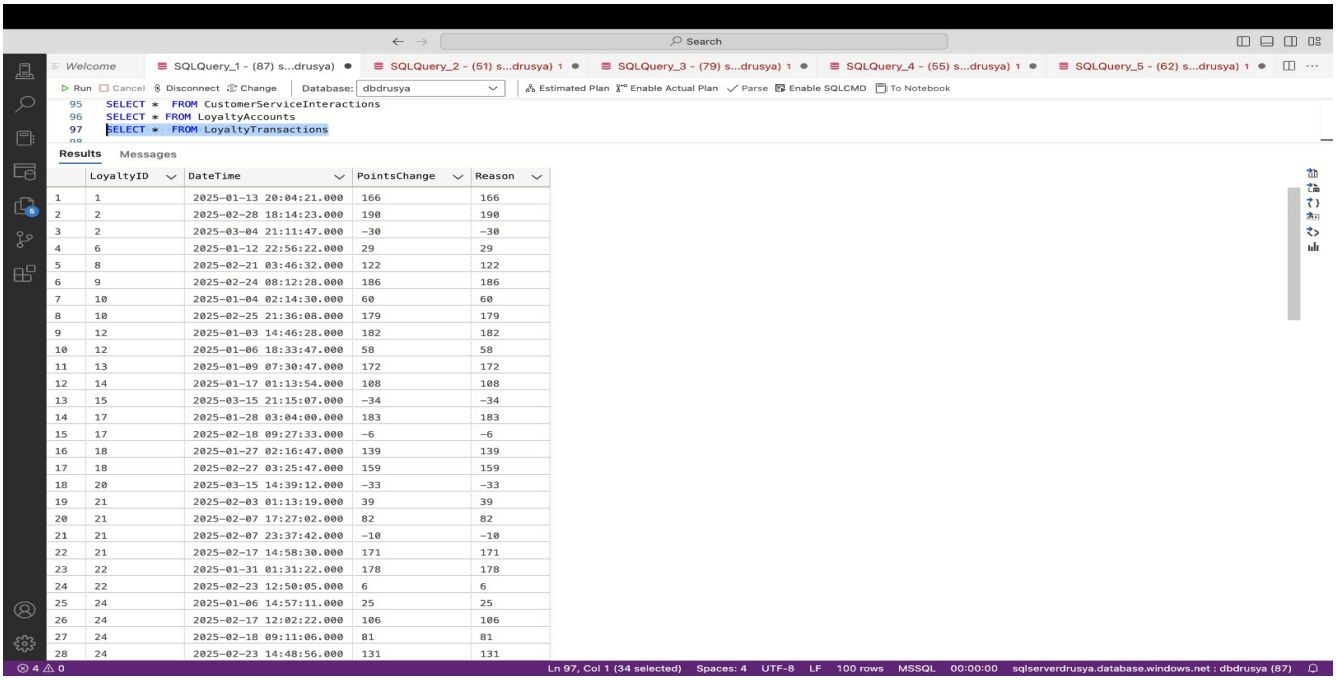
Loyalty accounts:



```
SELECT * FROM CustomerServiceInteractions
SELECT * FROM LoyaltyAccounts
SELECT * FROM LoyaltyTransactions
```

	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
13	13	36	2411	Silver	2024-07-05
14	14	34	3519	Gold	2020-09-12
15	15	68	4412	Silver	2023-05-09
16	16	38	1567	Gold	2021-06-17
17	17	35	4512	Silver	2021-07-03
18	18	41	1645	Bronze	2020-06-07
19	19	20	337	Gold	2024-09-06
20	20	29	3435	Bronze	2022-06-07
21	21	46	2622	Platinum	2025-01-16
22	22	21	1803	Silver	2024-04-15
23	23	17	1594	Bronze	2022-03-03
24	24	14	3707	Platinum	2023-05-23
25	25	55	187	Gold	2021-06-27
26	26	51	3024	Bronze	2020-09-15
27	27	6	908	Gold	2025-02-12
28	28	61	4616	Platinum	2020-10-04

Loyalty transactions:



	LoyaltyID	DateTime	PointsChange	Reason
1	1	2025-01-13 20:04:21.000	166	166
2	2	2025-02-28 18:14:23.000	190	190
3	2	2025-03-04 21:11:47.000	-30	-30
4	6	2025-01-12 22:56:22.000	29	29
5	8	2025-02-21 03:46:32.000	122	122
6	9	2025-02-24 08:12:28.000	186	186
7	10	2025-01-04 02:14:30.000	60	60
8	10	2025-02-25 21:36:08.000	179	179
9	12	2025-01-03 14:46:28.000	182	182
10	12	2025-01-06 18:33:47.000	58	58
11	13	2025-01-09 07:30:47.000	172	172
12	14	2025-01-17 01:13:54.000	108	108
13	15	2025-03-15 21:15:07.000	-34	-34
14	17	2025-01-28 03:04:00.000	183	183
15	17	2025-02-18 09:27:33.000	-6	-6
16	18	2025-01-27 02:16:47.000	139	139
17	18	2025-02-27 03:25:47.000	159	159
18	20	2025-03-15 14:39:12.000	-33	-33
19	21	2025-02-03 01:13:19.000	39	39
20	21	2025-02-07 17:27:02.000	82	82
21	21	2025-02-07 23:37:42.000	-10	-10
22	21	2025-02-17 14:58:30.000	171	171
23	22	2025-01-31 01:31:22.000	178	178
24	22	2025-02-23 12:50:05.000	6	6
25	24	2025-01-06 14:57:11.000	25	25
26	24	2025-02-17 12:02:22.000	106	106
27	24	2025-02-18 09:11:06.000	81	81
28	24	2025-02-23 14:48:56.000	131	131

STEP 4:

Now we have successfully cleaned the data and loaded into SQL tables which completes the Silver layer. Now we have to create views on top of these table based on business requirements.

VIEW 1: FOR AVERAGE ORDER VALUE

I used the following code for creating the view.

```
CREATE VIEW View_AverageOrderValue AS
SELECT
p.ProductID,
p.Name AS ProductName,
p.Category,
s.Location AS StoreLocation,
COUNT(ot.OrderID) AS TotalOrders,
SUM(ot.Amount) AS TotalAmount,
AVG(ot.Amount) 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;
```

Explanation of the code:

CREATE VIEW View_AverageOrderValue AS

This starts the creation of a **view** named `View_AverageOrderValue`.

- A **view** is a saved query that acts like a virtual table.
- Users can query it just like a regular table, but it doesn't store data itself—it's based on the underlying tables.

SELECT

p.ProductID,

p.Name AS ProductName,

p.Category,

s.Location AS StoreLocation,

COUNT(ot.OrderID) AS TotalOrders,

SUM(ot.Amount) AS TotalAmount,

AVG(ot.Amount) AS AverageOrderValue

This selects columns for output and calculates **aggregates**:

- `ProductID`, `ProductName`, `Category`: Information from the `Products` table.
- `StoreLocation`: Location from the `Stores` table (via in-store transactions).
- `TotalOrders`: Number of online transactions for the product.
- `TotalAmount`: Total sales value of those transactions.
- `AverageOrderValue`: Average value of each online transaction.

FROM OnlineTransactions ot

- This indicates that the **base table** for the query is `OnlineTransactions`, alias `ot`.
- All records and calculations begin with online transaction data.

JOIN Products p ON ot.ProductID = p.ProductID

- Joins `OnlineTransactions` to `Products` based on `ProductID`.
- This allows the view to include product details (name and category).

LEFT JOIN InStoreTransactions IST ON ot.CustomerID = IST.CustomerID

- A **LEFT JOIN** means:

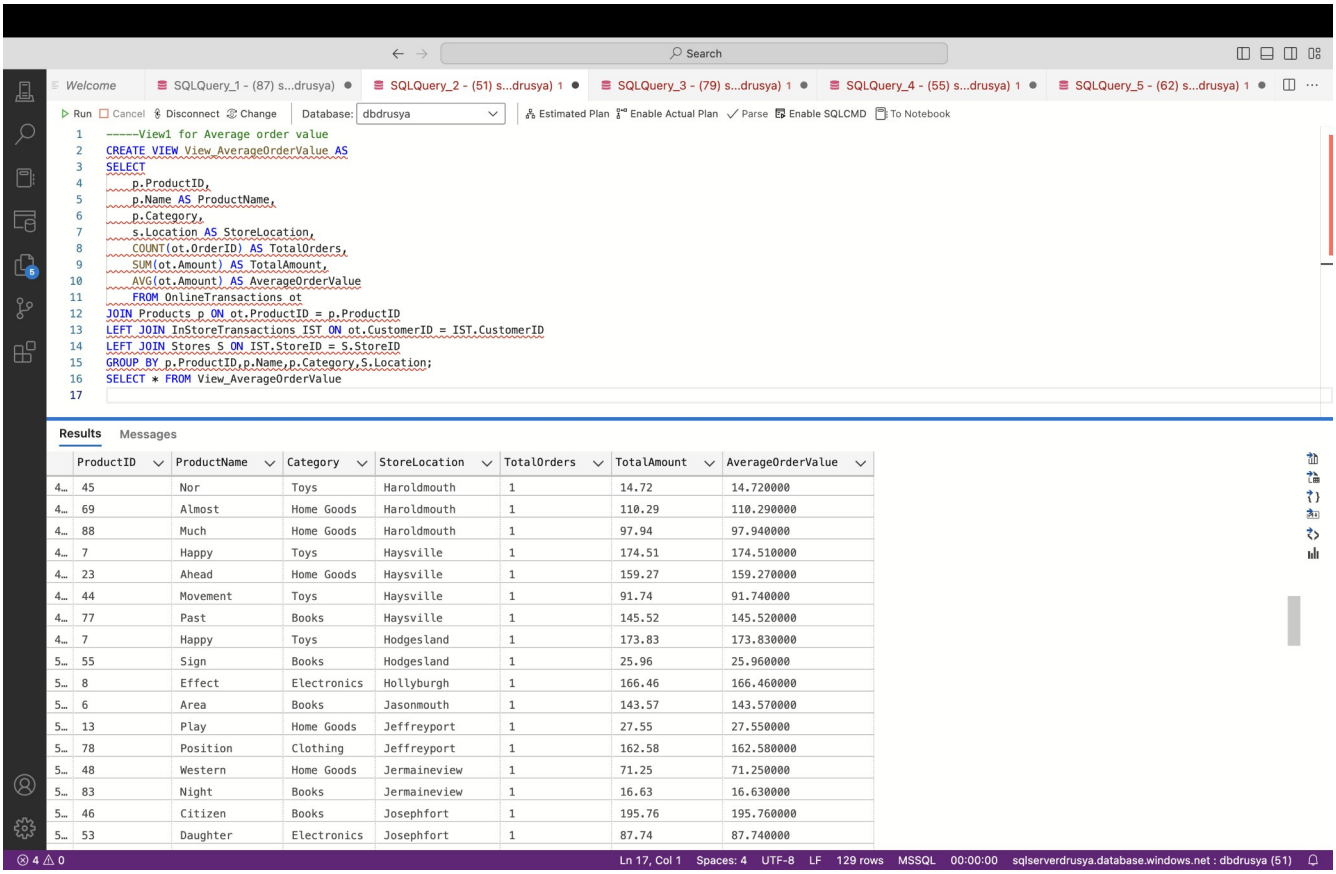
- Keep all online transactions, even if the customer has no in-store transactions.
- If the customer has in-store transactions, this join adds that data.
- The goal is to **link customers' online orders to their in-store purchases**.

LEFT JOIN Stores S ON IST.StoreID = S.StoreID

- This gets the **store location** from the store where the customer made in-store purchases.
- Also a **LEFT JOIN** — if the customer never made an in-store purchase, the `StoreLocation` will be `NULL`.

GROUP BY p.ProductID, p.Name, p.Category, S.Location;

- Groups the output **by product and store location**.
- Required so that aggregate functions like `COUNT`, `SUM`, and `AVG` work properly per group.
- Each row in the view represents a unique combination of a **product and a store location**.



VIEW 2: For categorizing customers based on spending,frequency of purchase and purchase tier

```
CREATE VIEW View_CustomerSegments AS
SELECT
c.CustomerID,
ISNULL(SUM(ot.Amount), 0) + ISNULL(SUM(ist.Amount), 0) AS TotalSpend,
ISNULL(COUNT(ot.OrderID), 0) + ISNULL(COUNT(ist.TransactionID), 0) AS PurchaseFrequency,
la.TierLevel,
CASE
WHEN (ISNULL(SUM(ot.Amount), 0) + ISNULL(SUM(ist.Amount), 0)) >= 1000 THEN 'High-Value
Customer'
WHEN (ISNULL(COUNT(ot.OrderID), 0) + ISNULL(COUNT(ist.TransactionID), 0)) = 1 THEN
'One-Time Buyer'
WHEN (ISNULL(COUNT(ot.OrderID), 0) + ISNULL(COUNT(ist.TransactionID), 0)) >= 5 AND
la.TierLevel IN ('Gold', 'Platinum') THEN 'Loyalty Champion'
ELSE 'Regular Customer'
END AS Segment
FROM Customers c
LEFT JOIN OnlineTransactions ot ON c.CustomerID = ot.CustomerID AND ot.Status = 'Completed'
LEFT JOIN InStoreTransactions ist ON c.CustomerID = ist.CustomerID
LEFT JOIN LoyaltyAccounts la ON c.CustomerID = la.CustomerID
GROUP BY c.CustomerID, la.TierLevel;
```

Explanation of the code:
ISNULL(SUM(ot.Amount), 0) + ISNULL(SUM(ist.Amount), 0) AS TotalSpend,
Calculates the **total amount spent** by the customer.

- Sums:
- `ot.Amount`: from **online transactions**
- `ist.Amount`: from **in-store transactions**
- Uses `ISNULL(..., 0)` to handle cases where the customer has **no transactions**, returning `0` instead of `NULL`.

ISNULL(COUNT(ot.OrderID), 0) + ISNULL(COUNT(ist.TransactionID), 0) AS PurchaseFrequency,
Counts **how many purchases** the customer made:

    - Online (`OrderID`)
    - In-store (`TransactionID`)
- Again, `ISNULL` ensures that customers with no transactions still get a count of `0`.

la.TierLevel,
Includes the customer's **loyalty tier** (e.g., Silver, Gold, Platinum) from the `LoyaltyAccounts` table.

```
CASE
  WHEN (TotalSpend) >= 1000 THEN 'High-Value Customer'
```

WHEN (PurchaseFrequency) = 1 THEN 'One-Time Buyer'
WHEN (PurchaseFrequency) >= 5 AND la.TierLevel IN ('Gold', 'Platinum') THEN 'Loyalty Champion'
ELSE 'Regular Customer'
END AS Segment
This **CASE expression** classifies each customer into a segment:

1. **High-Value Customer**: Spent **$1000 or more** total.
2. **One-Time Buyer**: Only **1 purchase**.
3. **Loyalty Champion**: Made **5 or more purchases** AND is in a **Gold or Platinum** tier.
4. **Regular Customer**: All others.

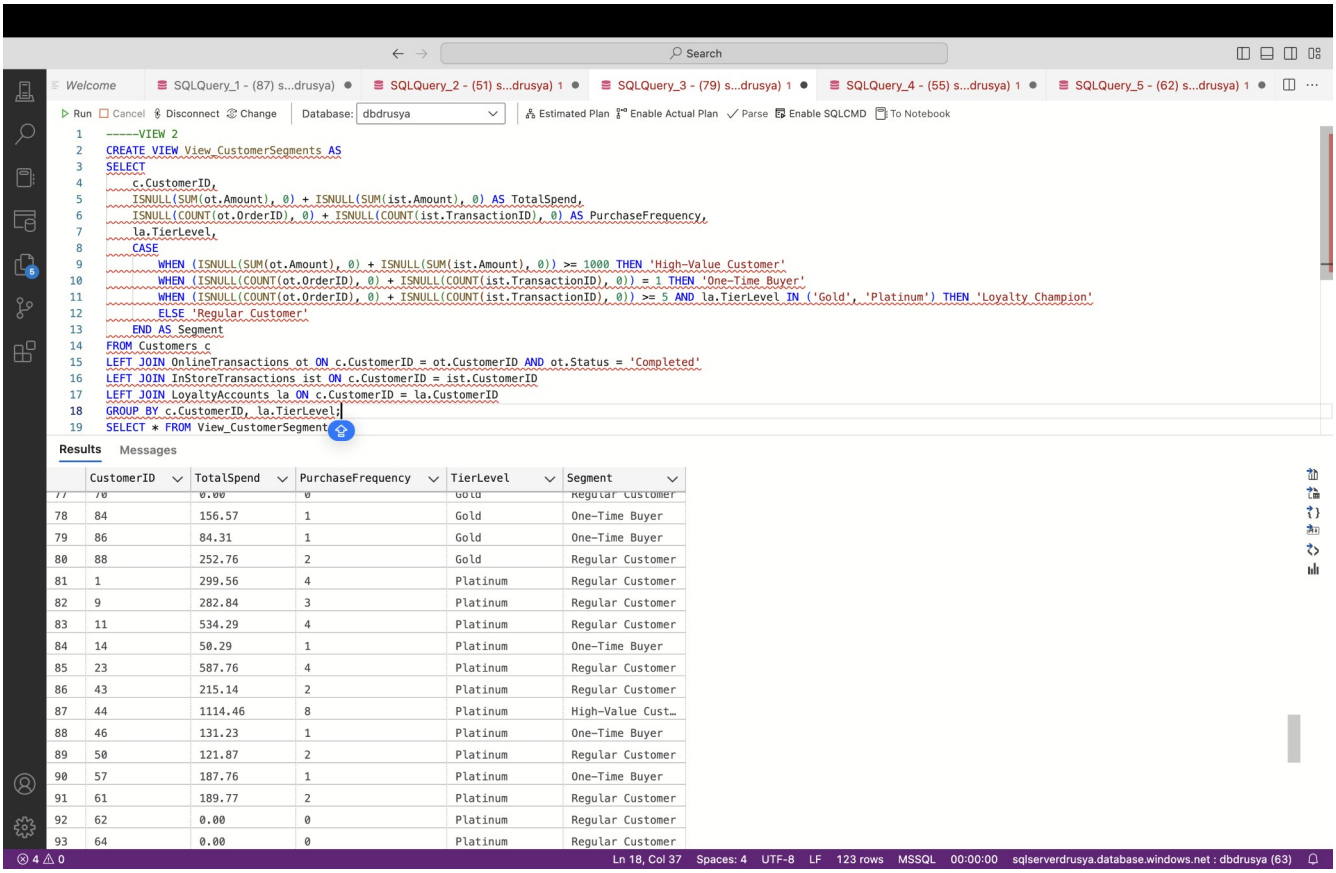
Note: Although `TotalSpend` and `PurchaseFrequency` were calculated earlier, they're **re-calculated inline** in the `CASE` expression for accuracy, not reused as aliases.

FROM Customers c
LEFT JOIN OnlineTransactions ot ON c.CustomerID = ot.CustomerID AND ot.Status = 'Completed'
LEFT JOIN InStoreTransactions ist ON c.CustomerID = ist.CustomerID
LEFT JOIN LoyaltyAccounts la ON c.CustomerID = la.CustomerID
Starts from the `Customers` table and **LEFT JOINs**:

- `OnlineTransactions` (only **completed** orders).
- `InStoreTransactions`
- `LoyaltyAccounts`
- **LEFT JOIN** ensures that even customers with **no purchases or no loyalty account** are still included.

GROUP BY c.CustomerID, la.TierLevel;
Groups the data **per customer and loyalty tier**, allowing aggregates like `SUM` and `COUNT` to work correctly.
Each row in `View_CustomerSegments` represents:

- A customer
- Their total spend across all channels
- Their total number of purchases
- Their loyalty tier (if any)
- Their assigned **segment label** (e.g., *High-Value Customer*)



VIEW 3: For Analyze DateTime to find peak days and times in-store vs. Online.

```
CREATE VIEW View_PeakTransactionTimes AS
SELECT
'Online' AS TransactionType,
DATENAME(WEEKDAY, DateTime) AS DayOfWeek,
DATEPART(HOUR, DateTime) AS HourOfDay,
COUNT(OrderID) AS TransactionCount
FROM OnlineTransactions
WHERE Status = 'Completed'
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);
```

Explanation of the code:

This SQL script creates a **view** named `View_PeakTransactionTimes` that helps analyze **when** (day and hour) transactions peak, for both **online** and **in-store** purchases.

```
SELECT
  'Online' AS TransactionType,
  DATENAME(WEEKDAY, DateTime) AS DayOfWeek,
  DATEPART(HOUR, DateTime) AS HourOfDay,
  COUNT(OrderID) AS TransactionCount
FROM OnlineTransactions
WHERE Status = 'Completed'
GROUP BY DATENAME(WEEKDAY, DateTime), DATEPART(HOUR, DateTime)
```

Pulls data from the `OnlineTransactions` table.

- Filters only **completed** orders.
- Groups transactions by:
  - **Day of the week** (`DATENAME(WEEKDAY, DateTime)`, e.g., Monday, Tuesday)
  - **Hour of the day** (`DATEPART(HOUR, DateTime)`, values from 0–23)
- Counts the number of transactions per (day, hour) combination.
- Adds a column `'Online'` as `TransactionType` to label the source.

```
UNION ALL
```

Combines the **online** and **in-store** parts of the query into one result set.

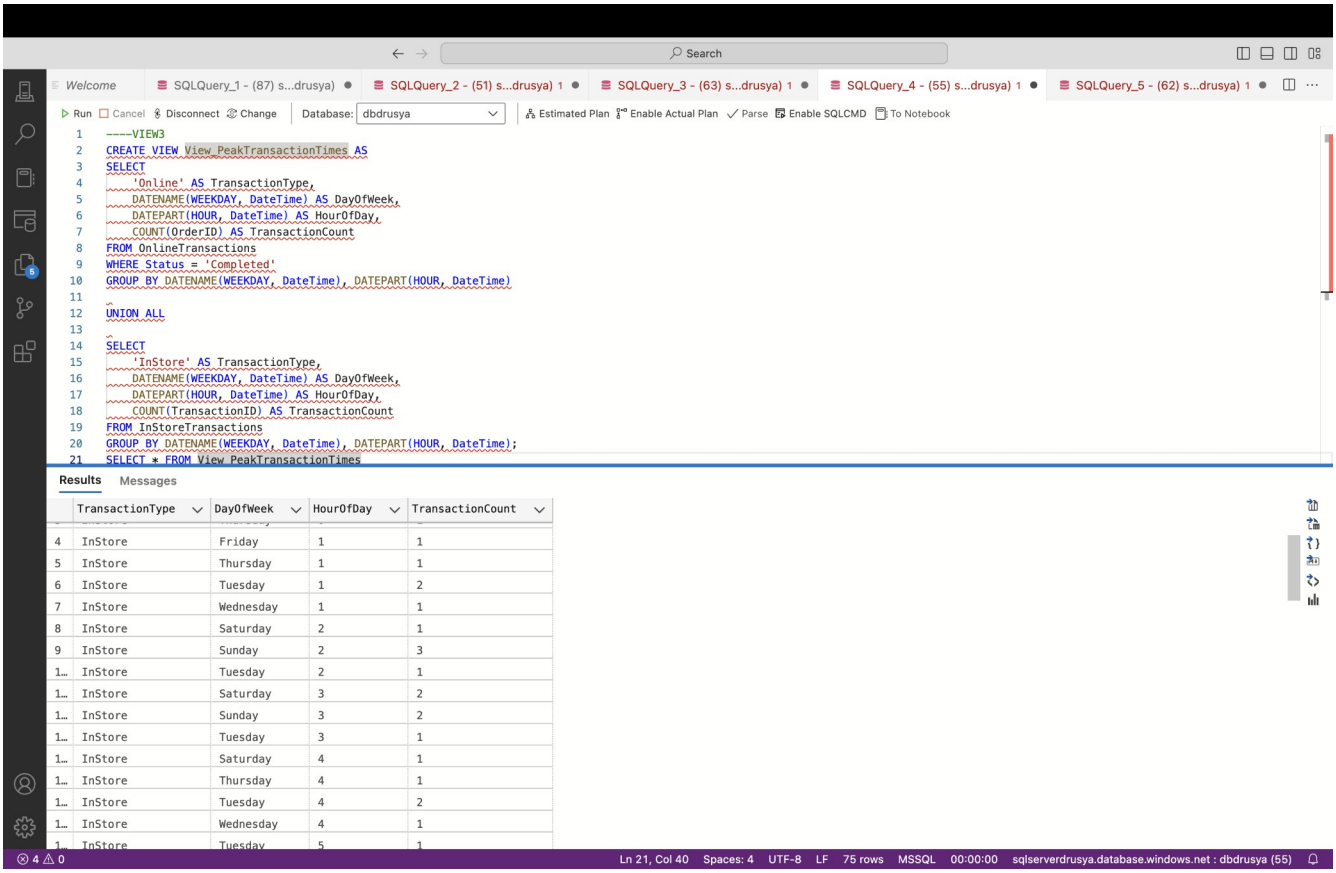
- `UNION ALL` keeps **all rows**, including duplicates if any (which is fine here because each part is already grouped).

```
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);

Pulls data from the `InStoreTransactions` table.

- No status filter—**all in-store transactions** are counted.
- Again, groups by day of week and hour of day.
- Counts number of in-store transactions per (day, hour) slot.
- Labels this row as `'InStore'` in the `TransactionType` column.



VIEW 4: For Number of interactions and resolution success rates per agent

CREATE VIEW View_AgentResolutionStats AS
SELECT
AgentID,
COUNT(*) AS TotalInteractions,
SUM(CASE WHEN ResolutionStatus = 'Resolved' THEN 1 ELSE 0 END) AS ResolvedCount,
CAST(
100.0 * SUM(CASE WHEN ResolutionStatus = 'Resolved' THEN 1 ELSE 0 END) /
NULLIF(COUNT(*), 0)

```
AS DECIMAL(5, 2)
) AS ResolutionSuccessRate
FROM CustomerServiceInteractions
GROUP BY AgentID;
```

Explanation of the code:
SELECT
AgentID,
The view groups results **per customer service agent**.

- `AgentID` identifies the employee.

COUNT(*) AS TotalInteractions,
Counts **all rows** (i.e., interactions) for that agent.

- This shows the **workload** of each agent.

SUM(CASE WHEN ResolutionStatus = 'Resolved' THEN 1 ELSE 0 END) AS ResolvedCount,
Uses a **conditional sum** to count how many interactions were marked as `'Resolved'`.

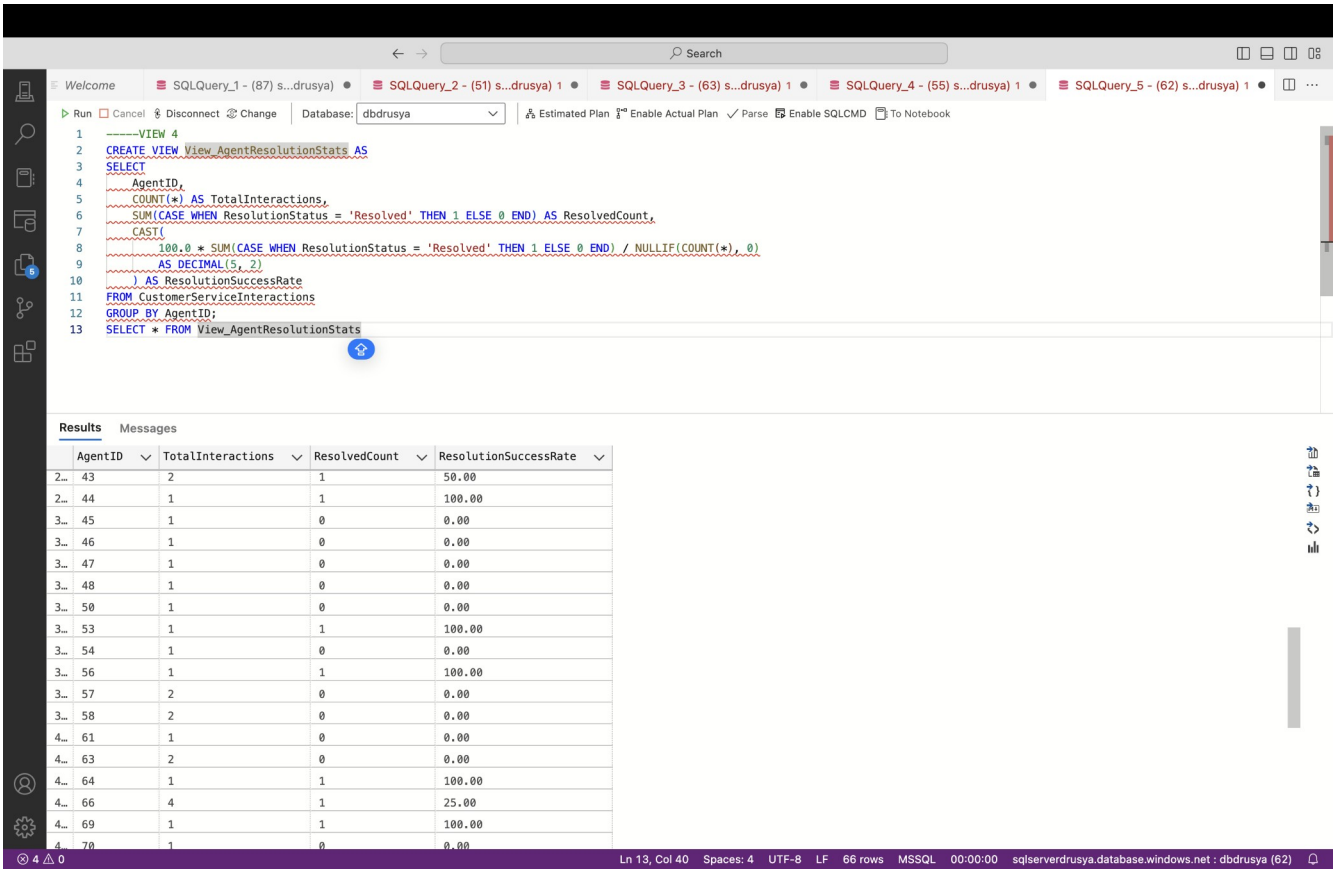
- This gives the total number of **successfully resolved cases** per agent.

```
CAST(
  100.0 * SUM(CASE WHEN ResolutionStatus = 'Resolved' THEN 1 ELSE 0 END)
  / NULLIF(COUNT(*), 0)
  AS DECIMAL(5, 2)
) AS ResolutionSuccessRate
```
Calculates the **percentage of resolved interactions** out of the total.

- Multiplies by `100.0` to convert to a percentage.
- Uses `NULLIF(COUNT(*), 0)` to avoid division by zero if an agent has no interactions.
- The result is **cast to a decimal** with two decimal places (e.g., `85.71`%).

This view helps us to identify:

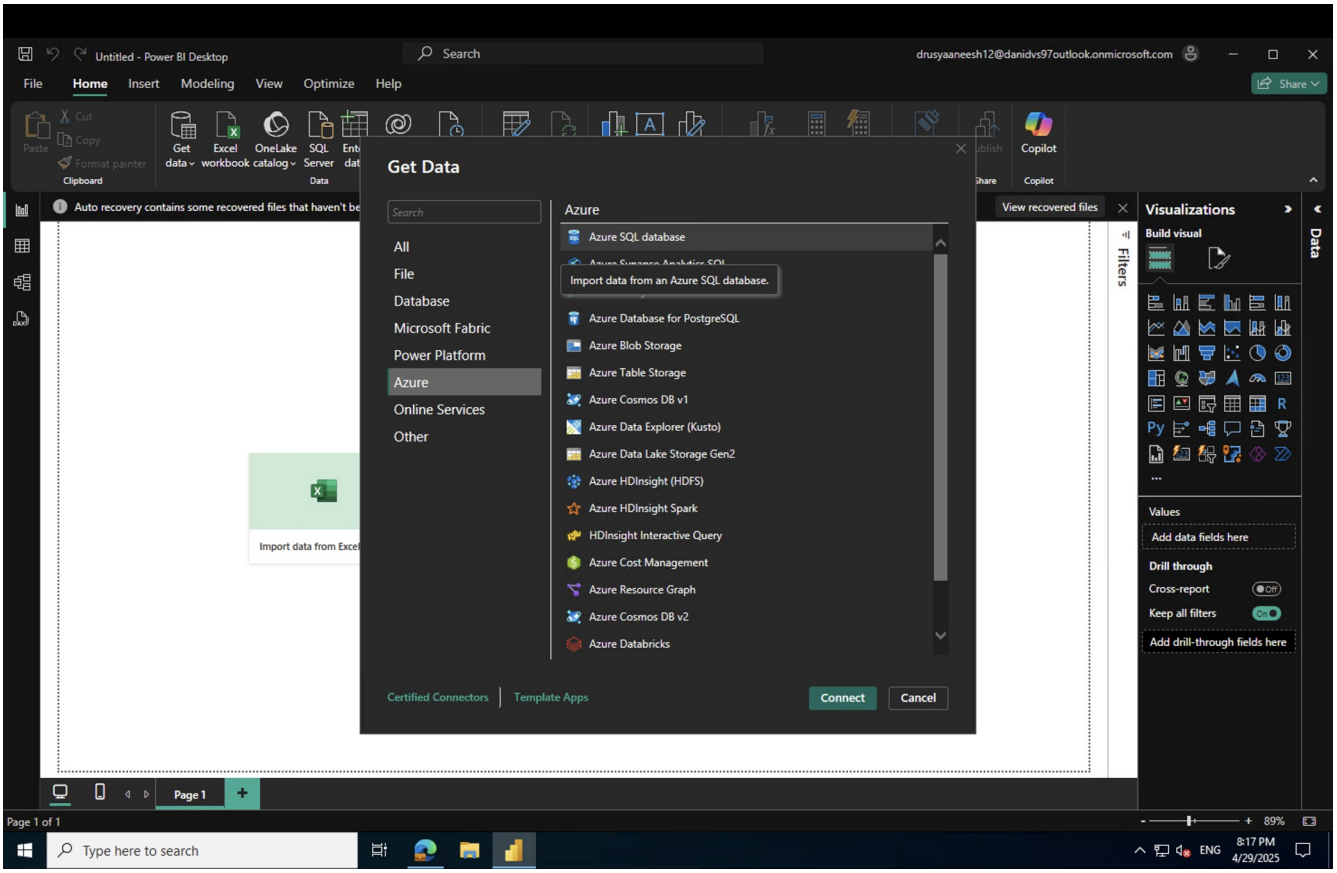
- Which agents resolve the most cases
- Who has the highest (or lowest) resolution success rate

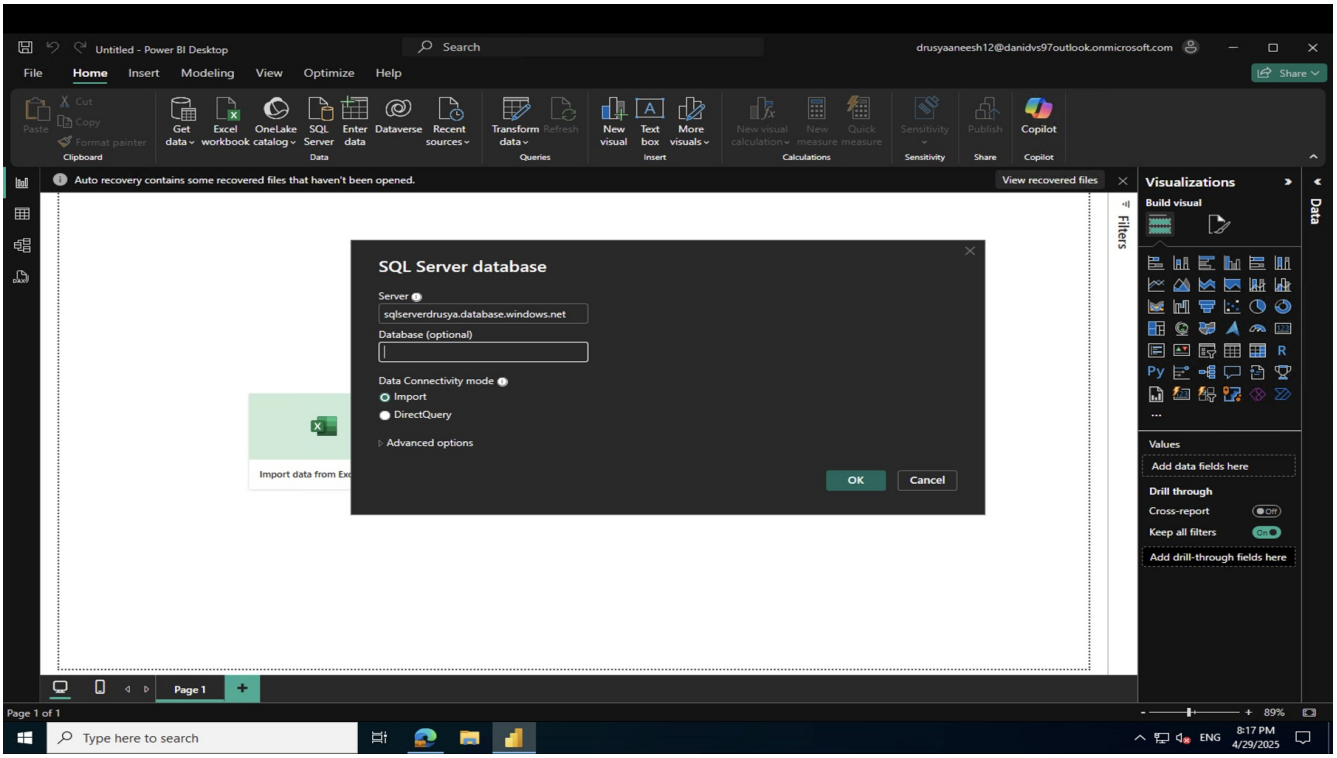


STEP 5:

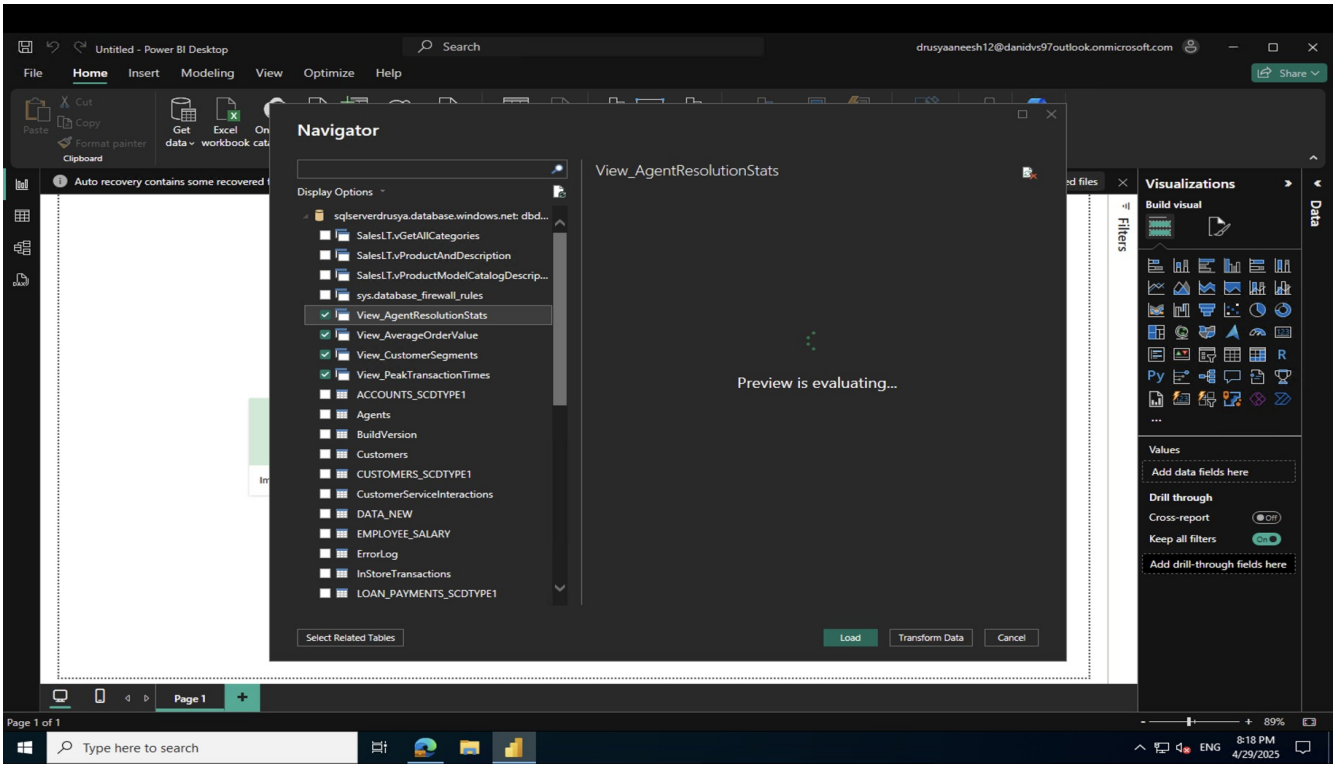
Now we have created the views in SQL database. We have to visualize the views with the help of Power BI and we have to publish the report in Fabric. For that we will create a connection with the power BI desktop and our sql server.

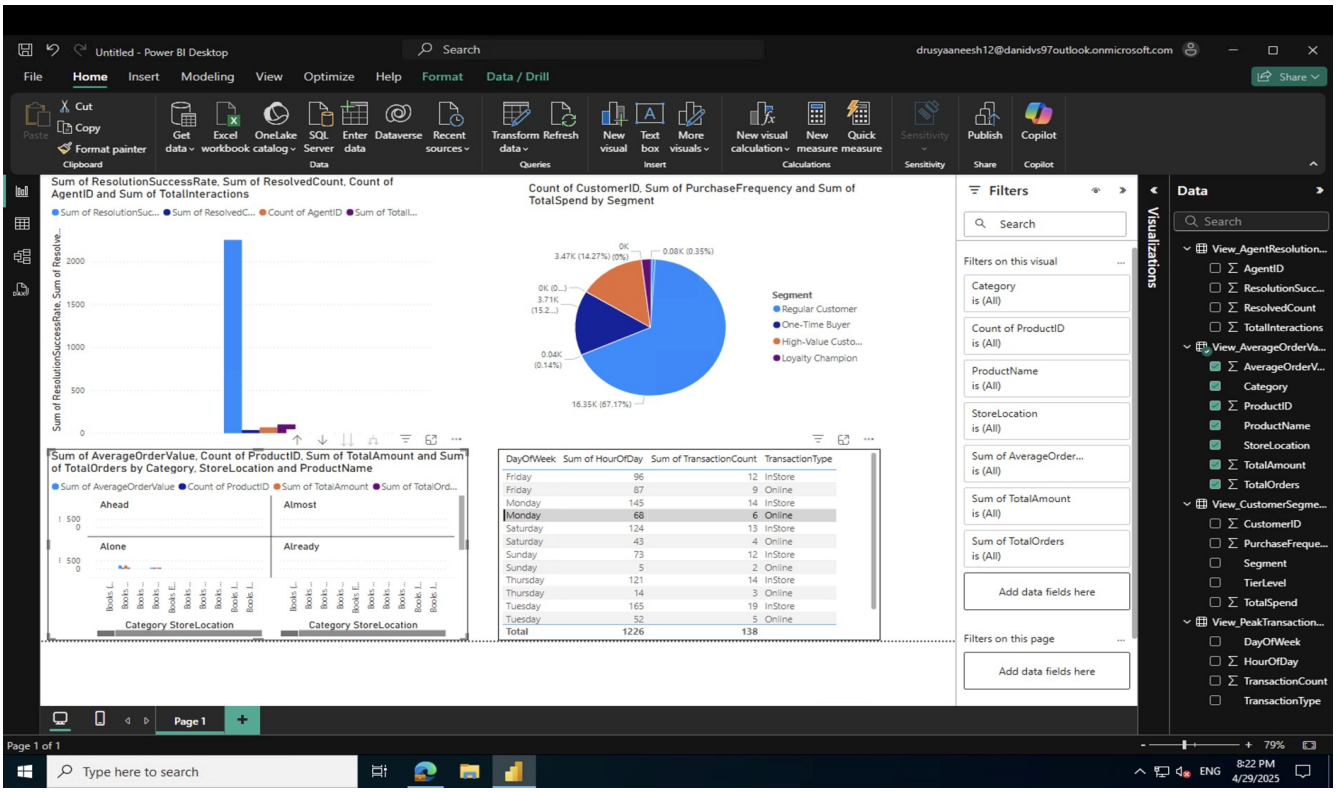
Open power BI → New blank report → Get data → Azure → Azure sql


Untitled - Power BI Desktop
Search
drusyaaneesh12@danidvs97outlook.onmicrosoft.com
File
Home
Insert
Modeling
View
Optimize
Help
Share
Get data
Excel workbook
OneLake catalog
SQL Server
Copilot
Get Data
Search
All
File
Database
Microsoft Fabric
Power Platform
Azure
Online Services
Other
Azure
Azure SQL database
Import data from an Azure SQL database.
Azure Database for PostgreSQL
Azure Blob Storage
Azure Table Storage
Azure Cosmos DB v1
Azure Data Explorer (Kusto)
Azure Data Lake Storage Gen2
Azure HDInsight (HDFS)
Azure HDInsight Spark
HDInsight Interactive Query
Azure Cost Management
Azure Resource Graph
Azure Cosmos DB v2
Azure Databricks
Certified Connectors
Template Apps
Connect
Cancel
Auto recovery contains some recovered files that haven't be
View recovered files
Visualizations
Build visual
Values
Add data fields here
Drill through
Cross-report
Keep all filters
Add drill-through fields here
Filters
Data
Import data from Excel
Page 1
Page 1 of 1
89%
Type here to search
ENG
8:17 PM
4/29/2025

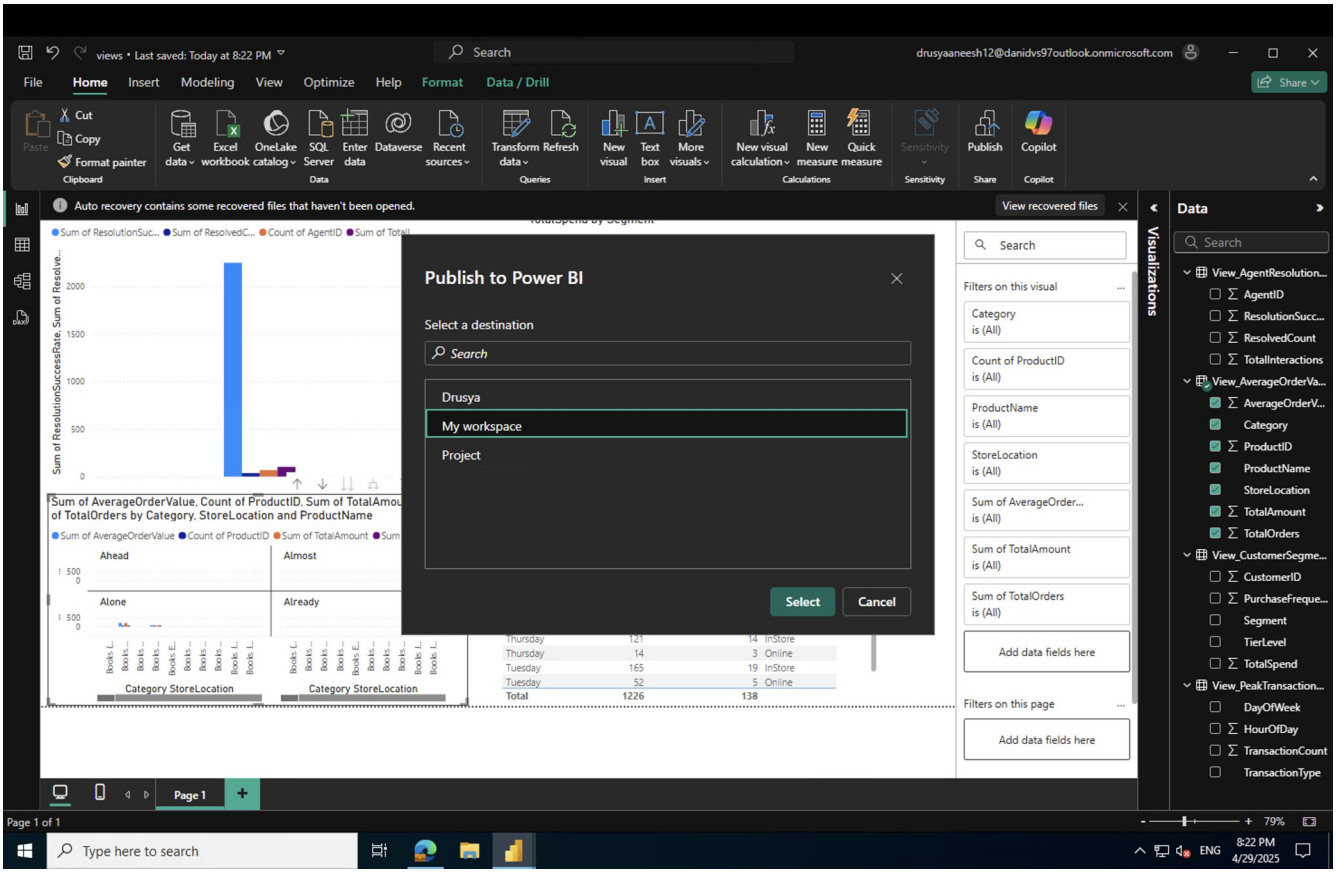

Untitled - Power BI Desktop
Search
drusyaaneesh12@danidvs97outlook.onmicrosoft.com
File
Home
Insert
Modeling
View
Optimize
Help
Share
Get data
Excel workbook
OneLake catalog
SQL Server
Enter data
Dataverse
Recent sources
Transform data
Refresh
New visual
Text box
More visuals
New visual calculation
New measure
Quick measure
Sensitivity
Publish
Copilot
SQL Server database
Server
sqlserverdrusya.database.windows.net
Database (optional)
Data Connectivity mode
Import
DirectQuery
Advanced options
OK
Cancel
Auto recovery contains some recovered files that haven't been opened.
View recovered files
Visualizations
Build visual
Values
Add data fields here
Drill through
Cross-report
Keep all filters
Add drill-through fields here
Filters
Data
Import data from Excel
Page 1
Page 1 of 1
89%
Type here to search
ENG
8:17 PM
4/29/2025

After connecting select the tables we want to visualize.

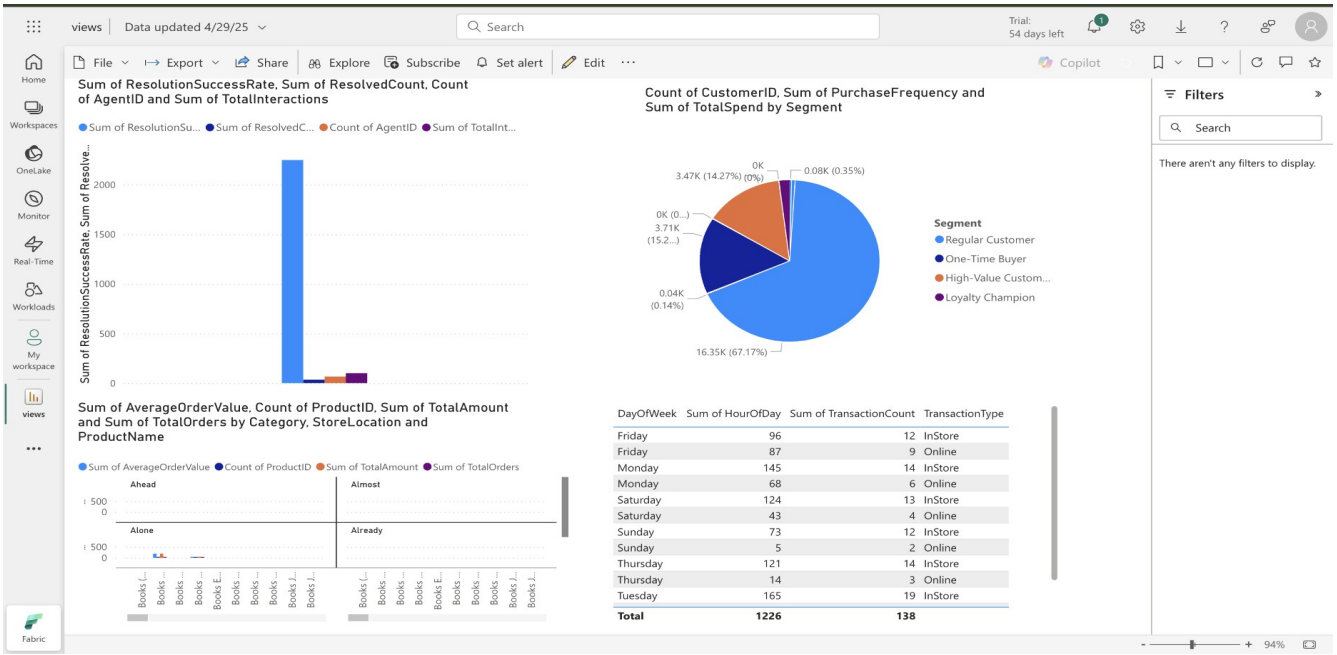




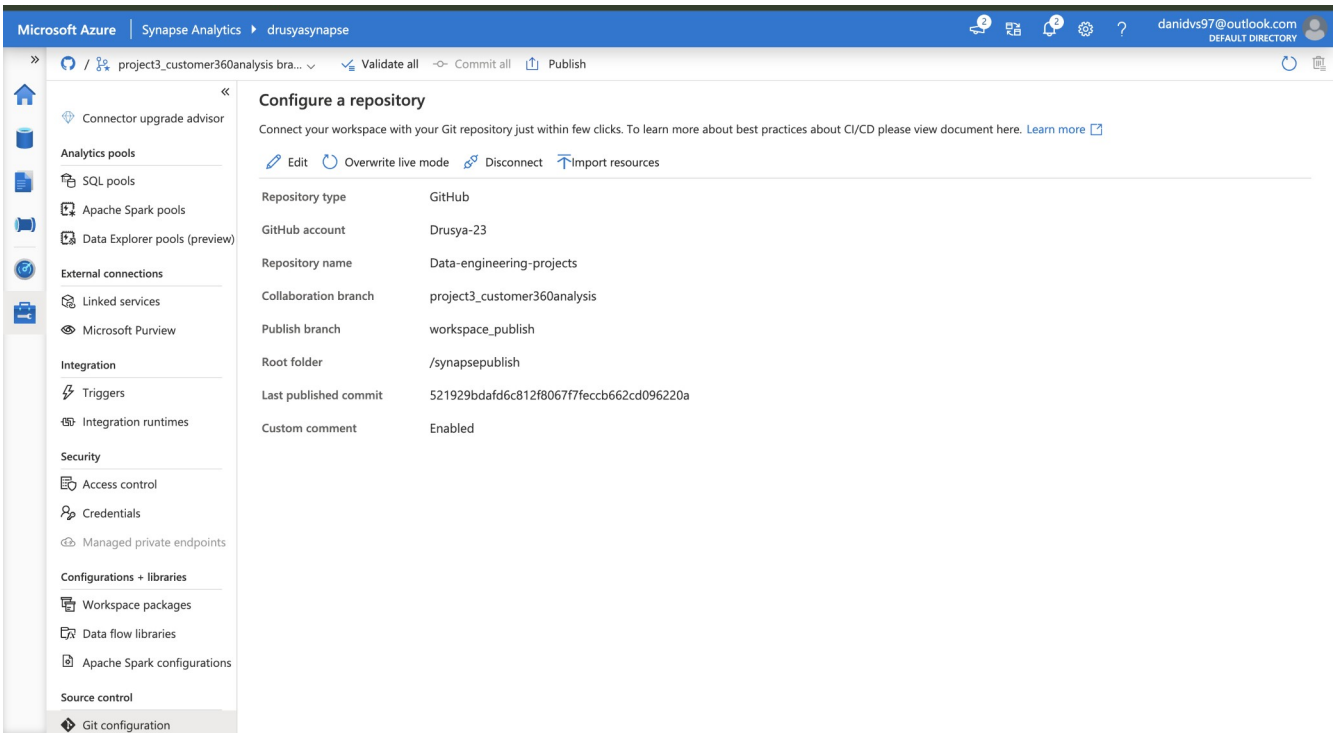
Pubish the report and save to fabric workspace.



Open Fabric and open the published file.



GIT HUB


Microsoft Azure
Synapse Analytics
drusyasynapse
danidvs97@outlook.com
DEFAULT DIRECTORY
project3_customer360analysis bra...
Validate all
Commit all
Publish
Connector upgrade advisor
Analytics pools
SQL pools
Apache Spark pools
Data Explorer pools (preview)
External connections
Linked services
Microsoft Purview
Integration
Triggers
Integration runtimes
Security
Access control
Credentials
Managed private endpoints
Configurations + libraries
Workspace packages
Data flow libraries
Apache Spark configurations
Source control
Git configuration
Configure a repository
Connect your workspace with your Git repository just within few clicks. To learn more about best practices about CI/CD please view document here. Learn more
Edit
Overwrite live mode
Disconnect
Import resources
Repository type
GitHub
GitHub account
Drusya-23
Repository name
Data-engineering-projects
Collaboration branch
project3_customer360analysis
Publish branch
workspace_publish
Root folder
/synapsepublish
Last published commit
521929bdafd6c812f8067f7feccb662cd096220a
Custom comment
Enabled