

BOOTCAMP PROJECT 3

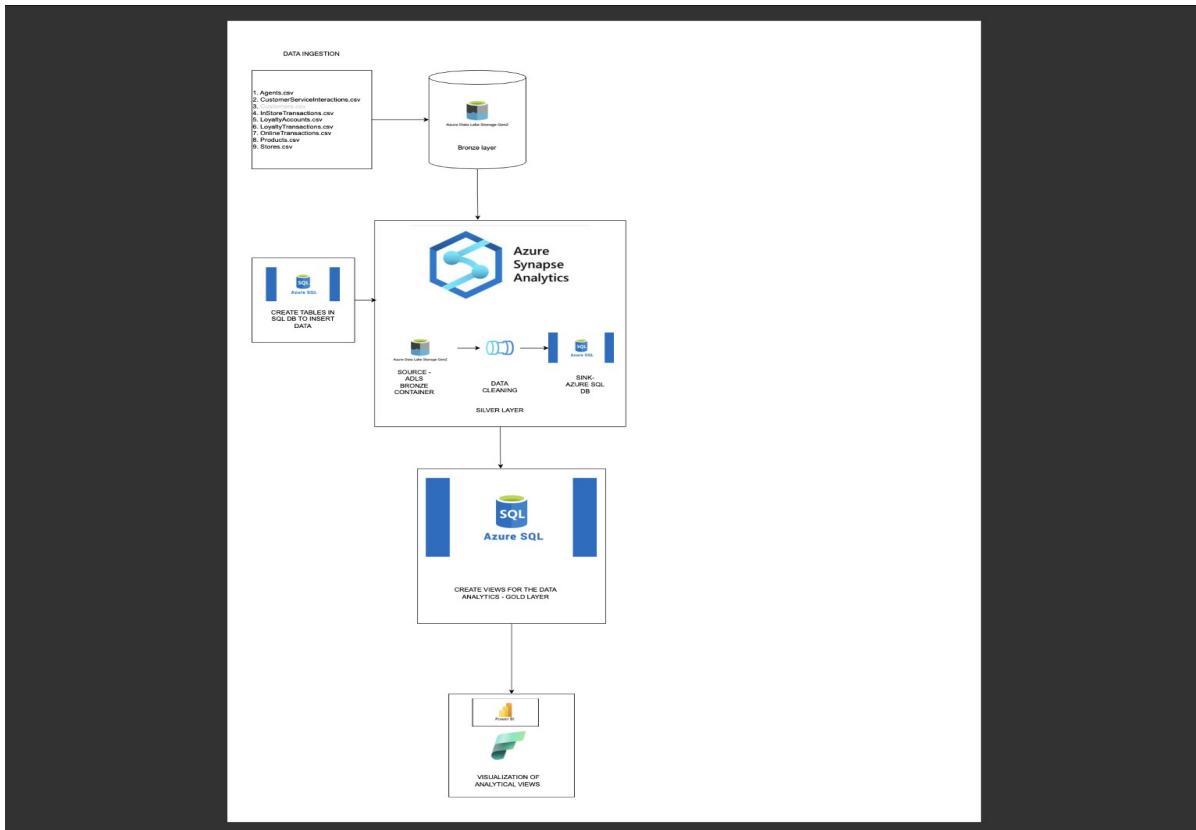
CUSTOMER 360 DATA INTEGRATION

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

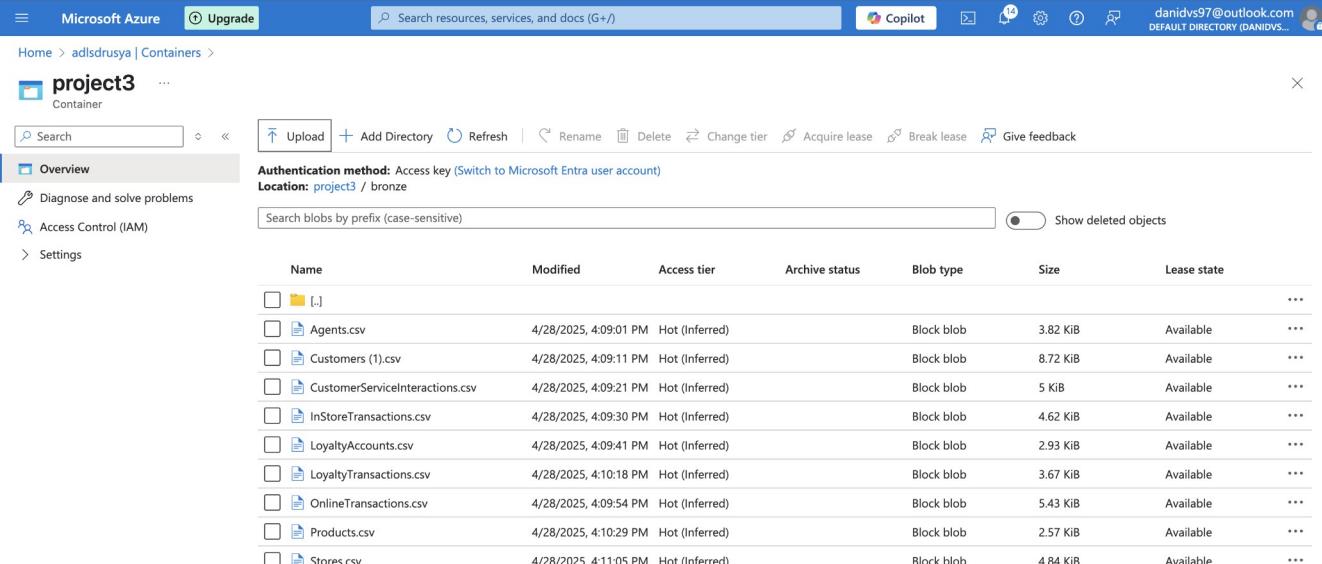
STEPS INVOLVED

- 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](#).



The screenshot shows the Microsoft Azure Storage Explorer interface. The left sidebar displays the 'Overview' tab for the 'project3' container. The main area lists several CSV files uploaded to the container, including 'Agents.csv', 'Customers (1).csv', 'CustomerServiceInteractions.csv', 'InStoreTransactions.csv', 'LoyaltyAccounts.csv', 'LoyaltyTransactions.csv', 'OnlineTransactions.csv', 'Products.csv', and 'Stores.csv'. Each file entry includes columns for Name, Modified, Access tier, Archive status, Blob type, Size, and Lease state. The 'Modified' column shows dates ranging from April 28, 2025, to April 29, 2025. The 'Access tier' column indicates most files are 'Hot (Inferred)'. The 'Archive status' column shows all files as 'Not yet archived'. The 'Blob type' column shows all files as 'Block blob'. The 'Size' column shows file sizes such as 3.82 KiB, 8.72 KiB, 5 KiB, etc. The 'Lease state' column shows all files as 'Available'. The 'Actions' column on the far right contains three dots for each file entry.

Add or remove favorites by pressing Cmd+Shift+F

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)

)

The screenshot shows the SSMS interface with a query window containing the following SQL code:

```
1  ----Create table customers
2  CREATE TABLE Customers
3  (
4      CustomerID INT PRIMARY KEY,
5      Name VARCHAR(100),
6      Email VARCHAR(100),
7      Address VARCHAR(250)
8  )
9  -----Create table Products
10 CREATE TABLE Products
11 (
12     ProductID INT PRIMARY KEY,
13     Name VARCHAR(100),
14     Category VARCHAR(50),
15     Price DECIMAL(10,2)
16 )
17 -----Create table Online Transactions
18 CREATE TABLE OnlineTransactions
19 (
20     OrderID INT PRIMARY KEY,
21     CustomerID INT,
22     ProductID INT,
23     DateTime DATETIME,
24     PaymentMethod VARCHAR(50),
25     Amount DECIMAL(10,2),
26     Status VARCHAR(20),
27     FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
28     FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
29 )
30 -----Create table Stores
31 CREATE TABLE Stores
32 (
33     StoreID INT PRIMARY KEY,
34     Location VARCHAR(100),
35     Manager VARCHAR(100)
```

The 'Messages' pane at the bottom shows the execution results:

```
4:52:13 PM Started executing query at Line 80
Commands completed successfully.
Total execution time: 00:00:00.045
```

SQL Server Management Studio (SSMS) screenshot showing the creation of several tables:

```

29 )
30 -----Create table Stores
31 CREATE TABLE Stores
32 (
33     StoreID INT PRIMARY KEY,
34     Location VARCHAR(100),
35     Manager VARCHAR(100),
36     OpenHours VARCHAR(50)
37 )
38 -----Create table InStoreTransactions
39 CREATE TABLE InStoreTransactions
40 (
41     TransactionID INT PRIMARY KEY,
42     CustomerID INT,
43     StoreID INT,
44     DateTime DATETIME,
45     Amount DECIMAL(10,2),
46     PaymentMethod VARCHAR(50),
47     FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
48     FOREIGN KEY (StoreID) REFERENCES Stores(StoreID)
49 )
50 -----Create table Agents
51 CREATE TABLE Agents(
52     AgentID INT PRIMARY KEY,
53     Name VARCHAR(100),
54     Department VARCHAR(50),
55     Shift VARCHAR(50)
56 )
57 -----Create table CustomerServiceInteractions
58 CREATE TABLE CustomerServiceInteractions
59 (
60     InteractionID INT PRIMARY KEY,
61     CustomerID INT,
62     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)

SQL Server Management Studio (SSMS) screenshot showing the creation of several tables:

```

56 )
57 -----Create table CustomerServiceInteractions
58 CREATE TABLE CustomerServiceInteractions
59 (
60     InteractionID INT PRIMARY KEY,
61     CustomerID INT,
62     DateTime DATETIME,
63     AgentID INT,
64     IssueType VARCHAR(50),
65     ResolutionStatus VARCHAR(50),
66     FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
67     FOREIGN KEY (AgentID) REFERENCES Agents(AgentID)
68 )
69 -----Create table LoyaltyAccounts
70 CREATE TABLE LoyaltyAccounts
71 (
72     LoyaltyID INT PRIMARY KEY,
73     CustomerID INT,
74     PointsEarned INT,
75     TierLevel VARCHAR(20),
76     JoinDate DATE,
77     FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
78 )
79 -----Create table LoyaltyTransactions
80 CREATE TABLE LoyaltyTransactions
81 (
82     LoyaltyID INT,
83     DateTime DATETIME,
84     PointsSpent INT,
85     Reason VARCHAR(100),
86     PRIMARY KEY (LoyaltyID,DateTime),
87     FOREIGN KEY (LoyaltyID) REFERENCES LoyaltyAccounts(LoyaltyID)
88 )

```

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)

The screenshot shows the SSMS interface with a query window open. The query window title is "SQLQuery_1 - (77) s...drusya". The code pane contains the following SELECT statements:

```
85 )  
89 SELECT * FROM Customers  
90 SELECT * FROM Products  
91 SELECT * FROM OnlineTransactions  
92 SELECT * FROM Stores  
93 SELECT * FROM InStoreTransactions  
94 SELECT * FROM Agents  
95 SELECT * FROM CustomerServiceInteractions  
96 SELECT * FROM LoyaltyAccounts  
97 SELECT * FROM LoyaltyTransactions
```

The results pane displays four tables:

- CustomerID**: Shows columns CustomerID, Name, Email, Address.
- ProductID**: Shows columns ProductID, Name, Category, Price.
- OrderID**: Shows columns OrderID, CustomerID, ProductID, DateTime, PaymentMethod, Amount, Status.
- StoreID**: Shows columns StoreID, Location, Manager, OpenHours.

The status bar at the bottom indicates: Ln 97, Col 34 (261 selected), Spaces: 4, UTF-8, LF, 0 rows, MSSQL, 00:00:00, sqlserverdrusya.database.windows.net : dbdrusya (85).

The screenshot shows the SSMS interface with a query window open. The query window title is "SQLQuery_1 - (77) s...drusya". The code pane contains the following SELECT statements:

```
85 )  
86 SELECT * FROM Customers  
87 SELECT * FROM Products  
88 SELECT * FROM OnlineTransactions  
89 SELECT * FROM Stores  
90 SELECT * FROM InStoreTransactions  
91 SELECT * FROM Agents  
92 SELECT * FROM CustomerServiceInteractions  
93 SELECT * FROM LoyaltyAccounts  
94 SELECT * FROM LoyaltyTransactions
```

The results pane displays four tables:

- CustomerID**: Shows columns CustomerID, Name, Email, Address.
- ProductID**: Shows columns ProductID, Name, Category, Price.
- OrderID**: Shows columns OrderID, CustomerID, ProductID, DateTime, PaymentMethod, Amount, Status.
- StoreID**: Shows columns StoreID, Location, Manager, OpenHours.

The status bar at the bottom indicates: 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:

The screenshot shows the 'Linked services' section in the Microsoft Azure Synapse Analytics workspace 'drusynapsee'. On the left, there's a sidebar with various navigation options like 'Connector upgrade advisor', 'Analytics pools', 'External connections', etc. The main area shows a list of existing linked services: 'AzureKeyVault1', 'drusynapsee-WorkspaceDefaultSqlServer', 'drusynapsee-WorkspaceDefaultStorage', 'ls_adls', and 'ls_sql'. To the right, the details for 'ls_adls' are being edited. The 'Name' field is set to 'ls_adls'. Under 'Type', 'Azure Key Vault' is selected. The 'Connect via integration runtime' dropdown has 'AutoResolveIntegrationRuntime' selected. The 'Authentication type' is set to 'Account key'. The 'URL' field contains 'https://adlsdrusya.dfs.core.windows.net/'. The 'Storage account key' field is set to 'Azure Key Vault'. The 'Annotations' section has a note: 'Linked services are much like connection strings, which define the connection information needed to access external data sources.' At the bottom, there are 'Save' and 'Cancel' buttons, and a 'Test connection' button.

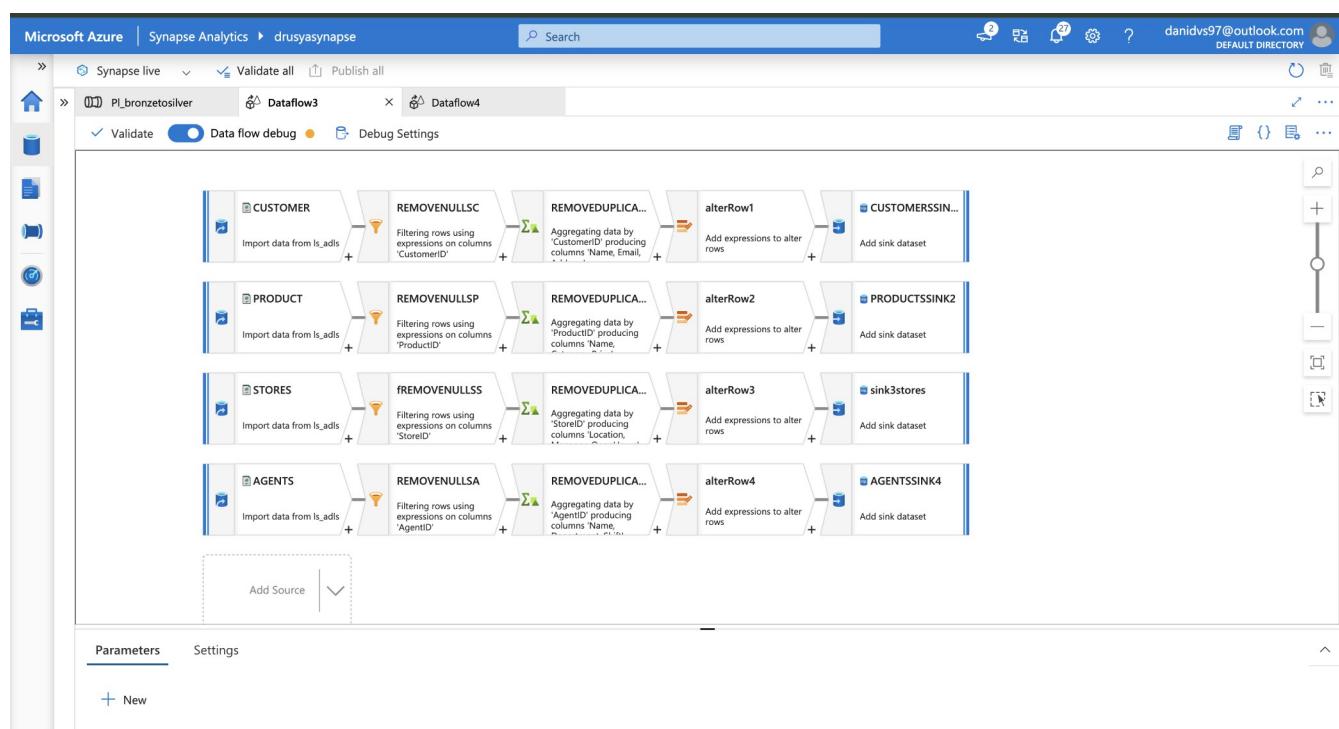
For SQL server used Azure key vault integration.

The screenshot shows the Microsoft Azure Synapse Analytics interface. On the left, there's a sidebar with various navigation options like 'Synapse live', 'Validate all', 'Publish all', 'Linked services', 'External connections', 'Integration', 'Security', 'Configurations + libraries', 'Source control', etc. The main area is titled 'Edit linked service' for an 'Azure SQL Database'. It shows a dropdown menu with 'AutoResolveIntegrationRuntime' selected. The configuration includes:

- Fully qualified domain name ***: sqldrusya.database.windows.net
- Database name ***: dbdrusya
- Authentication type ***: SQL authentication
- User name ***: dbdrusya
- Password**: Azure Key Vault (selected)
- AKV linked service ***: AzureKeyVault1
- Secret name ***: sqldr
- Secret version**: (dropdown menu)

At the bottom, there are 'Save' and 'Cancel' buttons, and a 'Test connection' link.

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.

The screenshot shows the Microsoft Azure Synapse Analytics Dataflow3 settings page. The top navigation bar includes 'Microsoft Azure', 'Synapse Analytics', 'drusyaynsynapse', 'Search' (with a magnifying glass icon), and user information ('danidvs97@outlook.com', 'DEFAULT DIRECTORY'). Below the navigation is a toolbar with icons for 'Validate', 'Data flow debug', 'Debug Settings', and more.

The main area displays a data flow diagram with three parallel streams: 'CUSTOMER', 'PRODUCT', and 'STORES'. Each stream starts with an 'Import data from ls_adls' source, followed by a 'REMOVENULLS' component, then a 'REMOVEDUPICATES' component, and finally an 'afterRow' component that adds a 'sink' dataset. The 'CUSTOMER' and 'PRODUCT' streams both have an 'Add one dataset' action after 'sink1' and 'sink2' respectively.

The 'Source settings' tab is active, showing the following configuration:

- Output stream name:** CUSTOMER
- Description:** Import data from ls_adls
- Source type:** Integration dataset Inline Workspace DB
- Inline dataset type:** DelimitedText
- Linked service:** ls_adls
- Skip line count:** (empty input field)
- Sampling:** Enable Disable

The screenshot shows the Microsoft Azure Synapse Analytics Dataflow3 settings page with the 'Source options' tab active. The top navigation and toolbar are identical to the previous screenshot.

The main area displays the same data flow diagram as the previous screenshot. The 'Source options' tab is active, showing the following configuration:

- File settings** (under 'Source options'):
 - File mode:** File Wildcard
 - File path:** project3 / bronze / Customers (1).csv
 - Allow no files found:**
 - Change data capture:**
 - Compression type:** No compression
 - Encoding:** Default(UTF-8)
 - Column delimiter:** Comma (,)
 - Row delimiter:** Default (\r,\n, or \r\n)
 - Quote character:** Double quote (")
 - Escape character:** Backslash (\)
 - First row as header:**
 - Null value:** (empty input field)

Now in the projection selected the import schema option.

The screenshot shows the Microsoft Azure Synapse Analytics Dataflow3 interface. The 'Projection' tab is selected. Under 'Import schema', there is a table showing column mappings:

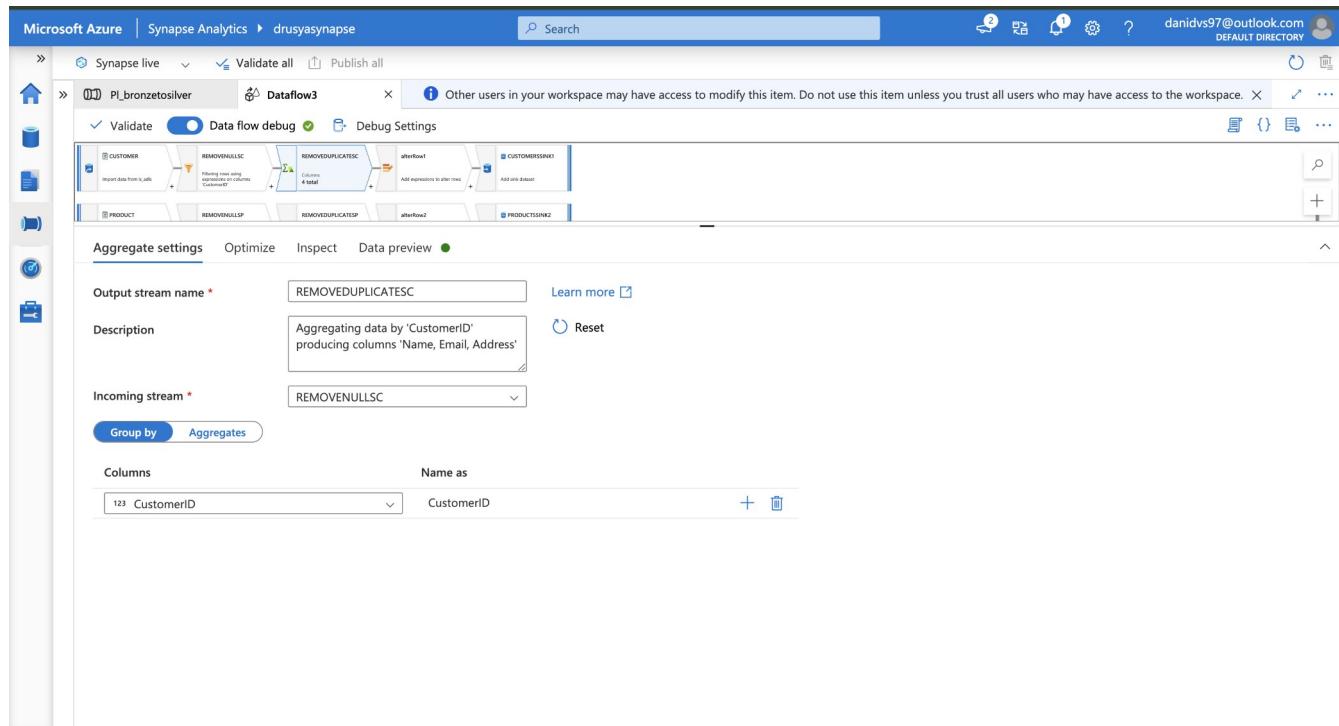
Column name	Type	Format
CustomerID	integer	Specify format
Name	string	Specify format
Email	string	Specify format
Address	string	Specify format

Next added a filter transformation to remove the nulls.

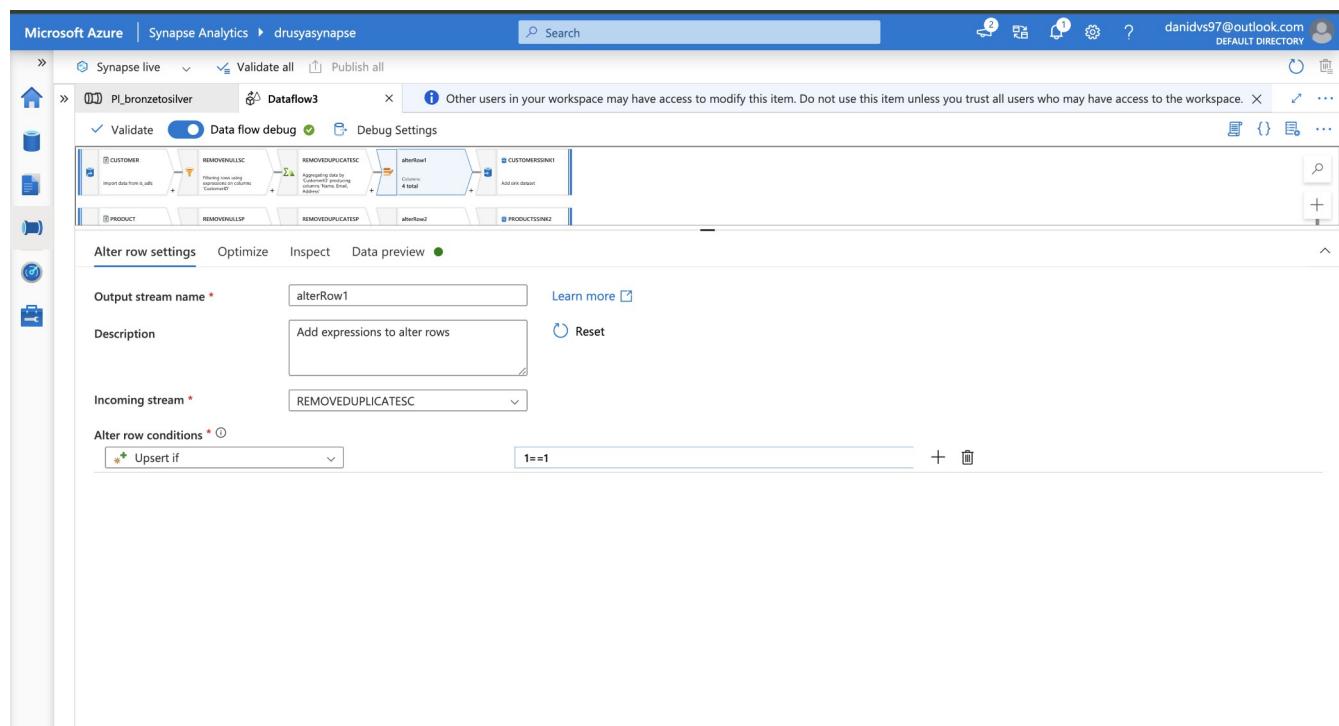
The screenshot shows the Microsoft Azure Synapse Analytics Dataflow3 interface. The 'Filter settings' tab is selected. The configuration is as follows:

- Output stream name: REMOVNULLSC
- Description: Filtering rows using expressions on columns 'CustomerID'
- Incoming stream: CUSTOMER
- Filter on: `isNull(CustomerID)`

Next added aggregate transformation to remove duplicates.



Next added alter row transformation.



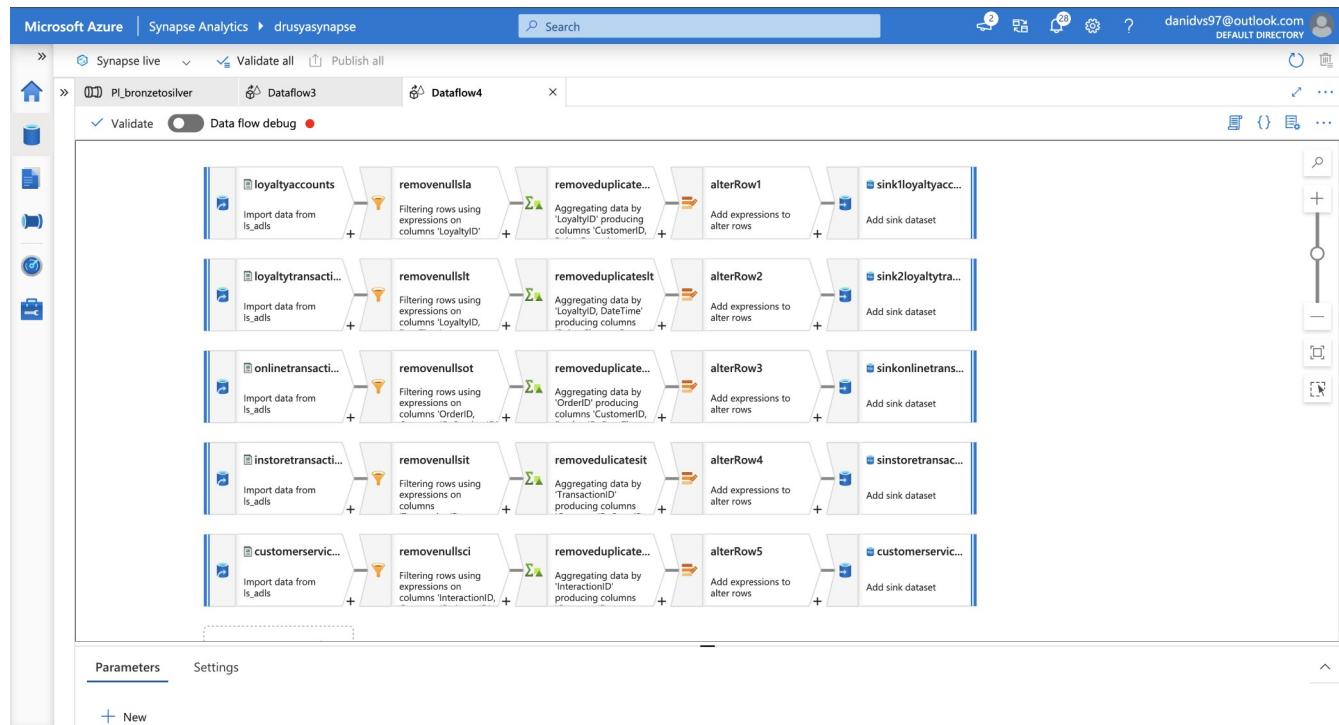
Now added the sink as the SQL table through inline option and gave the key column as the primary key. Also selected the update method as upsert.

The screenshot shows the Microsoft Azure Synapse Analytics Dataflow3 settings page. The sink configuration is set to 'CUSTOMERSINK1' with a description 'Add sink dataset'. The incoming stream is 'alterRow1'. The sink type is 'Inline'. The linked service is 'ls_sql'. Under options, 'Allow schema drift' is checked, while 'Validate schema' is not.

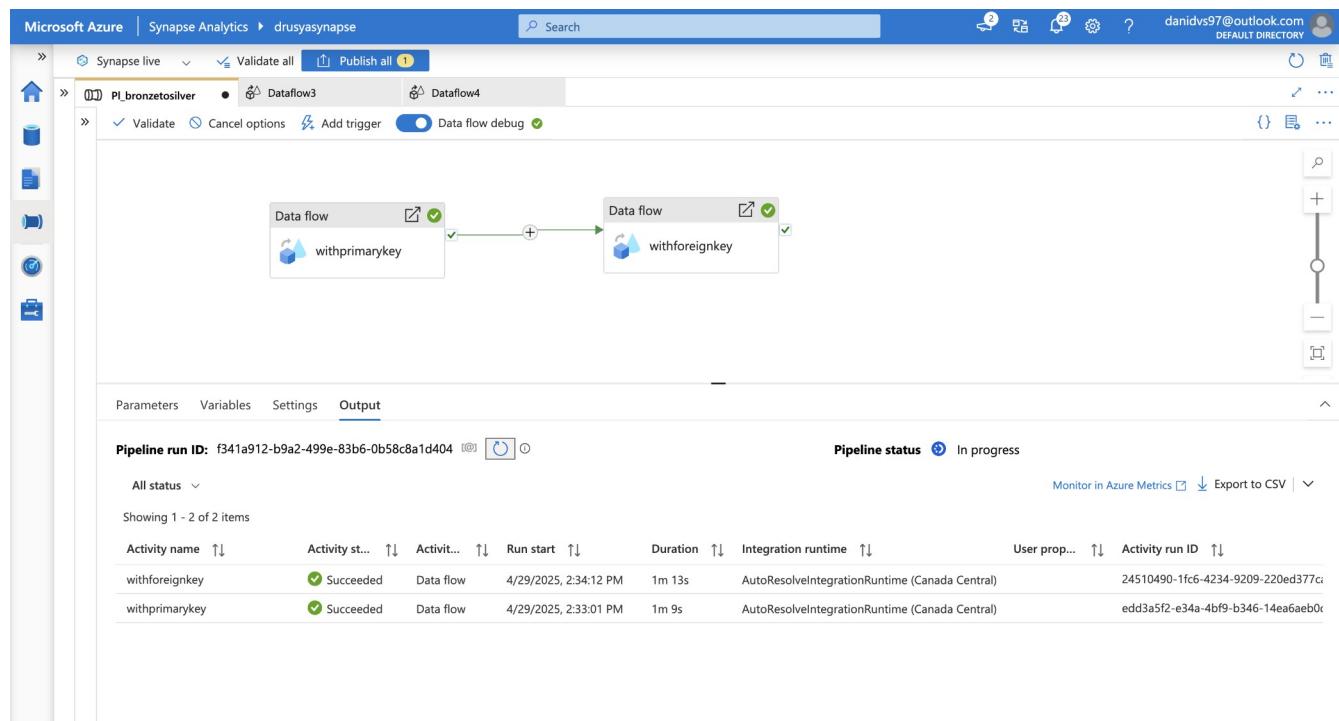
The screenshot shows the Microsoft Azure Synapse Analytics Dataflow3 settings page. The sink configuration is set to 'dbo' with a table name 'Customers'. The table action is 'None'. The update method is set to 'Allow upsert'. The key columns are defined as 'CustomerID'. The 'Use tempdb' option is checked. The pre SQL scripts option is set to 'List of scripts'.

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.

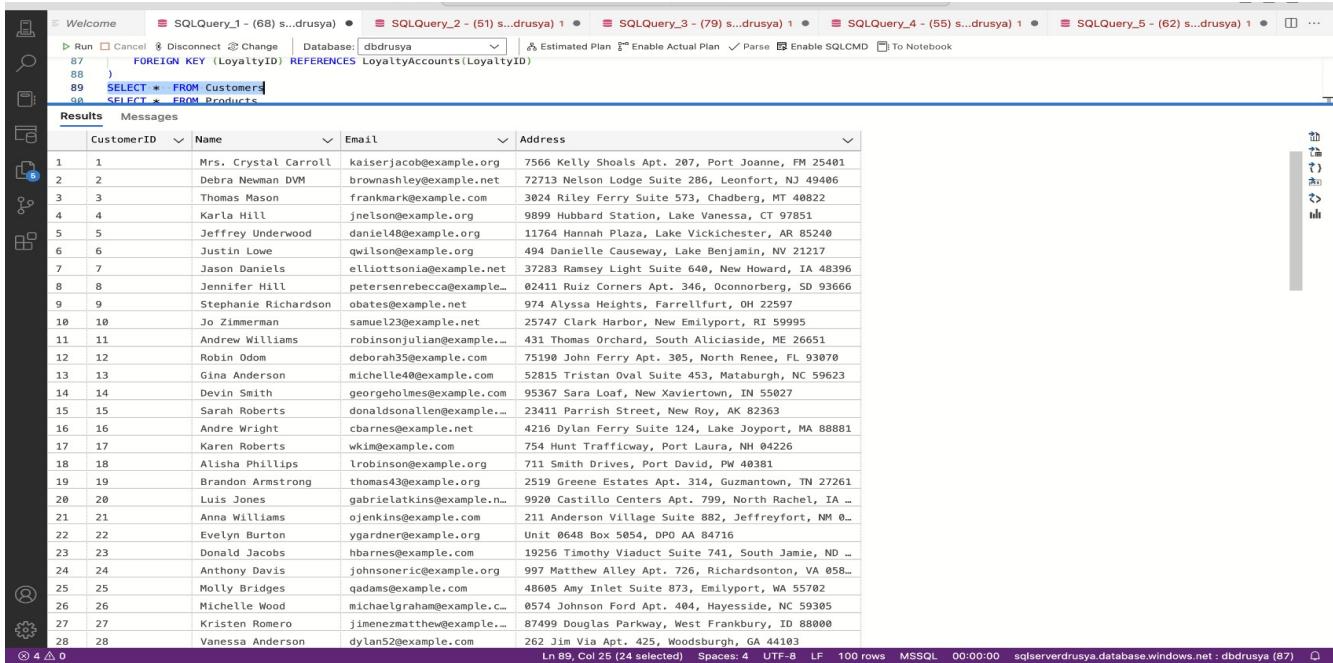


Now I successfully run the pipeline.



Outputs in data studio are as follows:

Customers table:



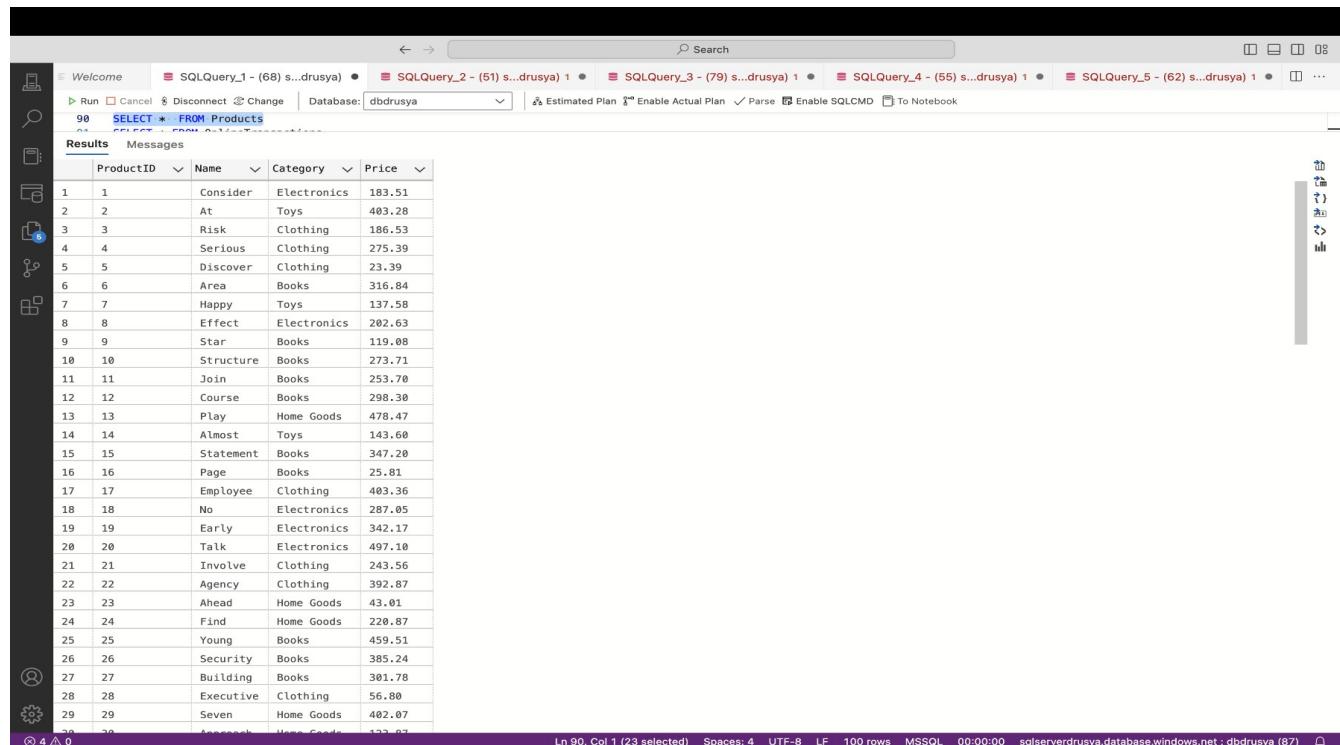
```

87 ) FOREIGN KEY (LoyaltyID) REFERENCES LoyaltyAccounts(LoyaltyID)
88 )
89 SELECT * FROM Customers
90 SELECT * FROM Products

```

	CustomerID	Name	Email	Address
1	1	Mrs. Crystal Carroll	kaiserjacob@example.org	7566 Kelly Shoals Apt. 207, Port Joanne, FM 25401
2	2	Debra Newman DVM	brownashley@example.net	72713 Nelson Lodge Suite 286, Leonfort, NJ 49406
3	3	Thomas Mason	frankmark@example.com	3024 Riley Ferry Suite 573, Chadberg, MT 40822
4	4	Karla Hill	jnelson@example.org	9899 Hubbard Station, Lake Vanessa, CT 97851
5	5	Jeffrey Underwood	daniel48@example.org	11764 Hannah Plaza, Lake Vickchester, AR 85240
6	6	Justin Lowe	qwilson@example.org	494 Danielle Causeway, Lake Benjamin, NV 21217
7	7	Jason Daniels	elliottsonia@example.net	37283 Ramsey Light Suite 640, New Howard, IA 48396
8	8	Jennifer Hill	petersenrebecca@example.com	02411 Ruiz Corners Apt. 346, OConnorberg, SD 93666
9	9	Stephanie Richardson	obates@example.net	974 Alyssa Heights, Farrelfurt, OH 23597
10	10	Jo Zimmerman	samuel123@example.net	25747 Clark Harbor, New Emilyport, RI 59995
11	11	Andrew Williams	robinsonjulian@example.com	431 Thomas Orchard, South Alciaside, ME 26651
12	12	Robin Odon	deborah35@example.com	7519 John Ferry Apt. 305, North Renee, FL 93070
13	13	Gina Anderson	michelle40@example.com	52815 Tristan Oval Suite 453, Mataburgh, NC 59623
14	14	Devin Smith	georgeholmes@example.com	95367 Sara Loaf, New Xaviertown, IN 55027
15	15	Sarah Roberts	donaldsonallen@example.com	23411 Parrish Street, New Roy, AK 82363
16	16	Andre Wright	cbarnes@example.net	4216 Dylan Ferry Suite 124, Lake Joyport, MA 88881
17	17	Karen Roberts	wkim@example.com	754 Hunt Trafficway, Port Laura, NH 04226
18	18	Alisha Phillips	lrobinson@example.org	711 Smith Drives, Port David, PR 40381
19	19	Brandon Armstrong	thomas43@example.org	2519 Greene Estates Apt. 314, Guzmantown, TN 27261
20	20	Luis Jones	gabrielatkins@example.com	9920 Castillo Centers Apt. 799, North Rachel, IA ..
21	21	Anna Williams	ojenkins@example.com	211 Anderson Village Suite 882, Jeffreyfort, NM ..
22	22	Evelyn Burton	ygardner@example.com	Unit 0648 Box 5054, DPO AA 84716
23	23	Donald Jacobs	hbarnes@example.com	19256 Timothy Viaduct Suite 741, South Jamie, ND ..
24	24	Anthony Davis	johnsoneric@example.org	997 Matthew Alley Apt. 726, Richardonton, VA 058..
25	25	Molly Bridges	qadams@example.com	48605 Amy Inlet Suite 873, Emilyport, WA 55702
26	26	Michelle Wood	michaelgraham@example.com	0574 Johnson Fort Apt. 404, Hayesside, NC 59305
27	27	Kristen Romero	jimenezmatthew@example.com	87499 Douglas Parkway, West Frankbury, ID 88000
28	28	Vanessa Anderson	dylan52@example.com	262 Jim Via Apt. 425, Woodsburgh, GA 44103

Products table:



```

90 SELECT * FROM Products

```

	ProductID	Name	Category	Price
1	1	Consider	Electronics	183.51
2	2	At	Toys	483.28
3	3	Risk	Clothing	186.53
4	4	Serious	Clothing	275.39
5	5	Discover	Clothing	23.39
6	6	Area	Books	316.84
7	7	Happy	Toys	137.58
8	8	Effect	Electronics	282.63
9	9	Star	Books	119.08
10	10	Structure	Books	273.71
11	11	Join	Books	253.70
12	12	Course	Books	298.30
13	13	Play	Home Goods	478.47
14	14	Almost	Toys	143.60
15	15	Statement	Books	347.20
16	16	Page	Books	25.81
17	17	Employee	Clothing	483.36
18	18	No	Electronics	287.05
19	19	Early	Electronics	342.17
20	20	Talk	Electronics	497.10
21	21	Involve	Clothing	243.56
22	22	Agency	Clothing	392.87
23	23	Ahead	Home Goods	43.01
24	24	Find	Home Goods	220.87
25	25	Young	Books	459.51
26	26	Security	Books	385.24
27	27	Building	Books	301.78
28	28	Executive	Clothing	56.80
29	29	Seven	Home Goods	402.07
30	30	Approach	Home Goods	122.07

Online transactions table:

The screenshot shows the SQL Server Management Studio interface with the following details:

- Query Editor:** The query window displays the following T-SQL code:


```
89  SELECT * FROM Customers
90  SELECT * FROM Products
91  SELECT * FROM OnlineTransactions
92  SELECT * FROM Stores
93  SELECT * FROM InStoreTransactions
```
- Results Grid:** The results grid shows the data from the OnlineTransactions table. The columns are OrderID, CustomerID, ProductID, DateTime, PaymentMethod, Amount, and Status. There are 28 rows of data.
- Status Bar:** The status bar at the bottom indicates "Ln 91, Col 1 (32 selected) Spaces: 4 UTF-8 LF 100 rows MSSQL 00:00:00 sqlserverdrusya.database.windows.net : dbdrusya (87)".

Stores table:

The screenshot shows the SQL Server Management Studio interface with the following details:

- Query Editor:** The query window displays the following T-SQL code:


```
91  SELECT * FROM OnlineTransactions
92  SELECT * FROM Stores
93  SELECT * FROM InStoreTransactions
```
- Results Grid:** The results grid shows the data from the Stores table. The columns are StoreID, Location, Manager, and OpenHours. There are 28 rows of data.
- Status Bar:** The status bar at the bottom indicates "Ln 92, Col 1 (21 selected) Spaces: 4 UTF-8 LF 100 rows MSSQL 00:00:00 sqlserverdrusya.database.windows.net : dbdrusya (87)".

Instore transactions table:

The screenshot shows the SQL Server Management Studio interface with the following details:

- Query Editor:** The query window displays the following T-SQL code:


```

91  SELECT * FROM OnlineTransactions
92  SELECT * FROM Stores
93  SELECT * FROM InStoreTransactions
      
```
- Results Grid:** The results grid shows the data from the InStoreTransactions table. The columns are: TransactionID, CustomerID, StoreID, DateTime, Amount, and PaymentMethod. The data consists of 28 rows of transaction records.
- Status Bar:** The status bar at the bottom indicates: Ln 93, Col 1 (33 selected), Spaces: 4, UTF-8, LF, 100 rows, MSSQL, 00:00:00, sqlserverdrusya.database.windows.net : dbdrusya (87).

Agents table:

The screenshot shows the SQL Server Management Studio interface with the following details:

- Query Editor:** The query window displays the following T-SQL code:


```

93  SELECT * FROM InStoreTransactions
94  SELECT * FROM Agents
95  SELECT * FROM CustomerServiceInteractions
      
```
- Results Grid:** The results grid shows the data from the Agents table. The columns are: AgentID, Name, Department, and Shift. The data consists of 28 rows of agent records.
- Status Bar:** The status bar at the bottom indicates: Ln 94, Col 1 (21 selected), Spaces: 4, UTF-8, LF, 100 rows, MSSQL, 00:00:00, sqlserverdrusya.database.windows.net : dbdrusya (87).

Customer service interactions table:

SQL Server Management Studio (SSMS) screenshot showing the results of a query against the CustomerServiceInteractions table. The table has columns: InteractionID, CustomerID, DateTime, AgentID, IssueType, and ResolutionStatus. Data rows show various interactions like Technical Issues, Complaints, and Billing issues from January to March 2025.

	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	38	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	58	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	68	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:

SQL Server Management Studio (SSMS) screenshot showing the results of a query against the LoyaltyAccounts table. The table has columns: LoyaltyID, CustomerID, PointsEarned, TierLevel, and JoinDate. Data rows show customer loyalty levels and joining dates from 2021 to 2025.

	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	2028-12-17
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	1883	Silver	2024-04-15
23	23	17	1594	Bronze	2022-03-03
24	24	14	3787	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 21:31:47.000	-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	186	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 **NUL**L.

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**.

```

1 -----View1 for Average order value
2 CREATE VIEW View_AverageOrderValue AS
3 SELECT
4     p.ProductID,
5     p.Name AS ProductName,
6     p.Category,
7     s.Location AS Storelocation,
8     COUNT(ot.OrderID) AS TotalOrders,
9     SUM(ot.Amount) AS TotalAmount,
10    AVG(ot.Amount) AS AverageOrderValue
11   FROM OnlineTransactions ot
12  JOIN Products p ON ot.ProductID = p.ProductID
13  LEFT JOIN InstoreTransactions IST ON ot.CustomerID = IST.CustomerID
14  LEFT JOIN Stores S ON IST.StoreID = S.StoreID
15 GROUP BY p.ProductID,p.Name,p.Category,S.Location;
16 SELECT * FROM View_AverageOrderValue
17

```

Results

ProductID	ProductName	Category	Storelocation	TotalOrders	TotalAmount	AverageOrderValue
4...	Nor	Toys	Haroldmouth	1	14.72	14.720000
4...	Almost	Home Goods	Haroldmouth	1	110.29	110.290000
4...	Much	Home Goods	Haroldmouth	1	97.94	97.940000
4...	Happy	Toys	Haysville	1	174.51	174.510000
4...	Ahead	Home Goods	Haysville	1	159.27	159.270000
4...	Movement	Toys	Haysville	1	91.74	91.740000
4...	Past	Books	Haysville	1	145.52	145.520000
4...	Happy	Toys	Hodgesland	1	173.83	173.830000
5...	Sign	Books	Hodgesland	1	25.96	25.960000
5...	Effect	Electronics	Hollyburgh	1	166.46	166.460000
5...	Area	Books	Jasomouth	1	143.57	143.570000
5...	Play	Home Goods	Jeffreypoint	1	27.55	27.550000
5...	Position	Clothing	Jeffreypoint	1	162.58	162.580000
5...	Western	Home Goods	Jermainevie	1	71.25	71.250000
5...	Night	Books	Jermainevie	1	16.63	16.630000
5...	Citizen	Books	Josephfort	1	195.76	195.760000
5...	Daughter	Electronics	Josephfort	1	87.74	87.740000

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 **NUL**L.

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*)

```

1   ----VIEW 2
2   CREATE VIEW View_CustomerSegments AS
3   SELECT
4       c.CustomerID,
5       ISNULL(SUM(ot.Amount), 0) + ISNULL(SUM(iist.Amount), 0) AS TotalSpend,
6       ISNULL(COUNT(ot.OrderID), 0) + ISNULL(COUNT(iist.TransactionID), 0) AS PurchaseFrequency,
7       la.TierLevel,
8       CASE
9           WHEN (ISNULL(GUM(ot.Amount), 0) + ISNULL(GUM(iist.Amount), 0)) >= 1000 THEN 'High-Value Customer'
10          WHEN (ISNULL(COUNT(ot.OrderID), 0) + ISNULL(COUNT(iist.TransactionID), 0)) = 1 THEN 'One-Time Buyer'
11          WHEN (ISNULL(COUNT(ot.OrderID), 0) + ISNULL(COUNT(iist.TransactionID), 0)) >= 5 AND la.TierLevel IN ('Gold', 'Platinum') THEN 'Loyalty Champion'
12          ELSE 'Regular Customer'
13      END AS Segment
14  FROM Customers c
15  LEFT JOIN OnlineTransactions ot ON c.CustomerID = ot.CustomerID AND ot.Status = 'Completed'
16  LEFT JOIN InStoreTransactions ist ON c.CustomerID = ist.CustomerID
17  LEFT JOIN LoyaltyAccounts la ON c.CustomerID = la.CustomerID
18  GROUP BY c.CustomerID, la.TierLevel
19  SELECT * FROM View_CustomerSegment

```

Results Messages

	CustomerID	TotalSpend	PurchaseFrequency	TierLevel	Segment
77	70	0.00	0	Gold	Regular Customer
78	84	156.57	1	Gold	One-Time Buyer
79	86	84.31	1	Gold	One-Time Buyer
80	88	252.76	2	Gold	Regular Customer
81	1	299.56	4	Platinum	Regular Customer
82	9	282.84	3	Platinum	Regular Customer
83	11	534.29	4	Platinum	Regular Customer
84	14	50.29	1	Platinum	One-Time Buyer
85	23	587.76	4	Platinum	Regular Customer
86	43	215.14	2	Platinum	Regular Customer
87	44	1114.46	8	Platinum	High-Value Cust...
88	46	131.23	1	Platinum	One-Time Buyer
89	50	121.87	2	Platinum	Regular Customer
90	57	187.76	1	Platinum	One-Time Buyer
91	61	189.77	2	Platinum	Regular Customer
92	62	0.00	0	Platinum	Regular Customer
93	64	0.00	0	Platinum	Regular Customer

Ln 18, Col 37 Spaces: 4 UTF-8 LF 123 rows MSSQL 00:00:00 sqlserverdrusya.database.windows.net : dbdrusya (63)

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.

The screenshot shows a SQL Server Management Studio (SSMS) interface. The top bar has tabs for 'Welcome' and several other query windows. The main area contains a T-SQL script for creating a view named 'View_PeakTransactionTimes'. The script includes two SELECT statements: one for 'Online' transactions and one for 'InStore' transactions, both grouped by weekday and hour. The results pane below shows the output of the 'InStore' query, which lists transaction counts for each day of the week and hour of the day. The results table has columns: TransactionType, DayOfWeek, HourOfDay, and TransactionCount. The data shows various counts for different days and hours, with some rows being repeated (e.g., multiple rows for Tuesday at hour 1).

	TransactionType	DayOfWeek	HourOfDay	TransactionCount
4	InStore	Friday	1	1
5	InStore	Thursday	1	1
6	InStore	Tuesday	1	2
7	InStore	Wednesday	1	1
8	InStore	Saturday	2	1
9	InStore	Sunday	2	3
10	InStore	Tuesday	2	1
11	InStore	Saturday	3	2
12	InStore	Sunday	3	2
13	InStore	Tuesday	3	1
14	InStore	Saturday	4	1
15	InStore	Thursday	4	1
16	InStore	Tuesday	4	2
17	InStore	Wednesday	4	1
18	InStore	Tuesday	5	1

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**.

- **Agent ID** 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

```

1   ----VIEW 4
2   CREATE VIEW View_AgentResolutionStats AS
3   SELECT
4       AgentID,
5           COUNT(*) AS TotalInteractions,
6           SUM(CASE WHEN ResolutionStatus = 'Resolved' THEN 1 ELSE 0 END) AS ResolvedCount,
7           CAST(
8               100.0 * SUM(CASE WHEN ResolutionStatus = 'Resolved' THEN 1 ELSE 0 END) / NULLIF(COUNT(*), 0)
9               AS DECIMAL(5, 2))
10      ) AS ResolutionSuccessRate
11  FROM CustomerServiceInteractions
12  GROUP BY AgentID;
13  SELECT * FROM View_AgentResolutionStats

```

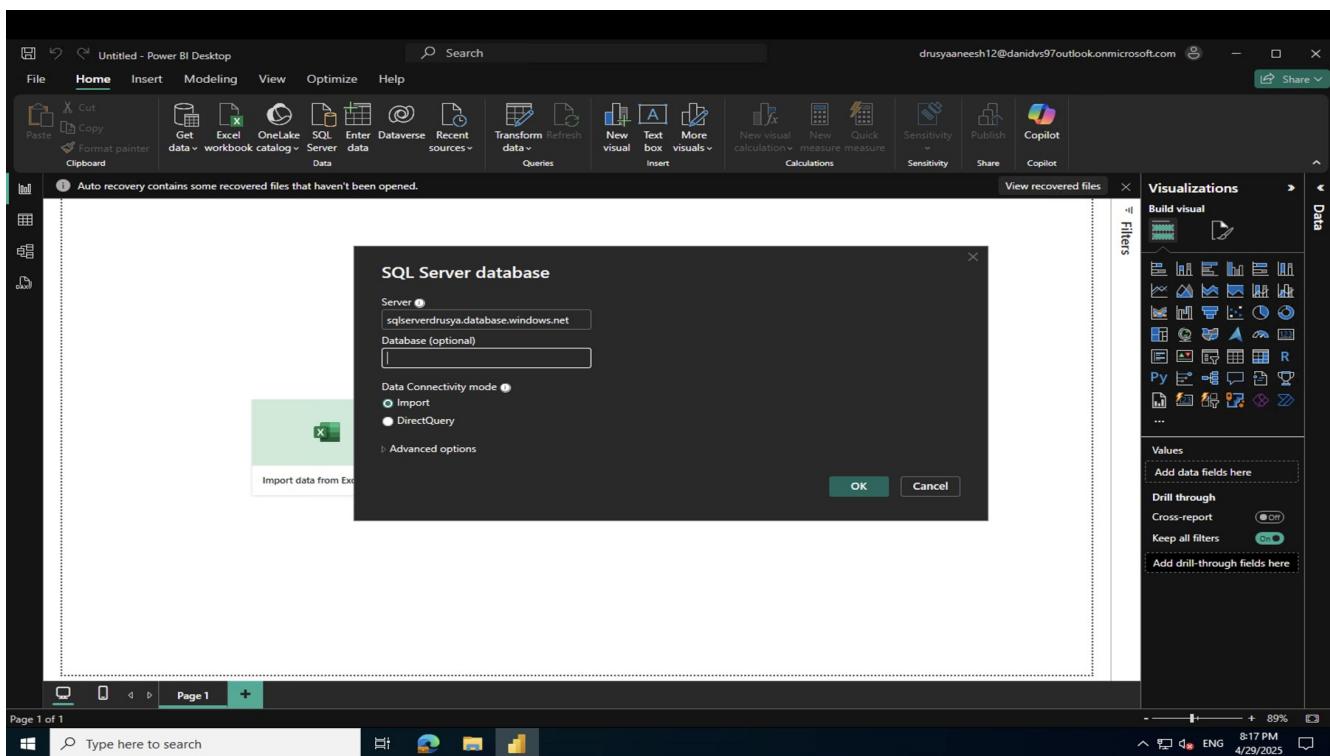
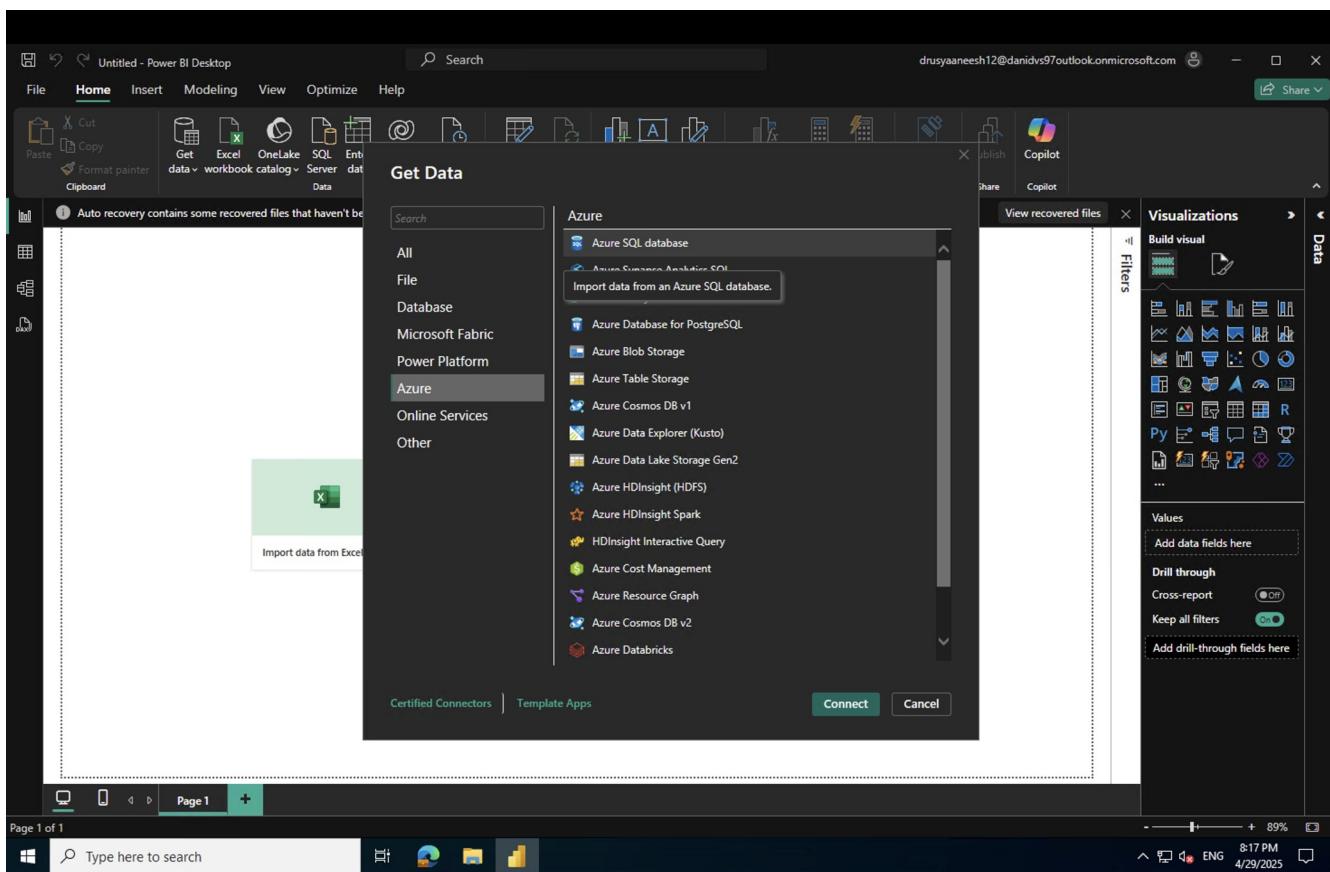
AgentID	TotalInteractions	ResolvedCount	ResolutionSuccessRate
43	2	1	50.00
44	1	1	100.00
45	1	0	0.00
46	1	0	0.00
47	1	0	0.00
48	1	0	0.00
50	1	0	0.00
53	1	1	100.00
54	1	0	0.00
56	1	1	100.00
57	2	0	0.00
58	2	0	0.00
61	1	0	0.00
63	2	0	0.00
64	1	1	100.00
66	4	1	25.00
69	1	1	100.00
70	1	0	0.00

Ln 13, Col 40 Spaces: 4 UTF-8 LF 66 rows MSSQL 00:00:00 sqlserverdrusya.database.windows.net : dbdrusya (62)

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

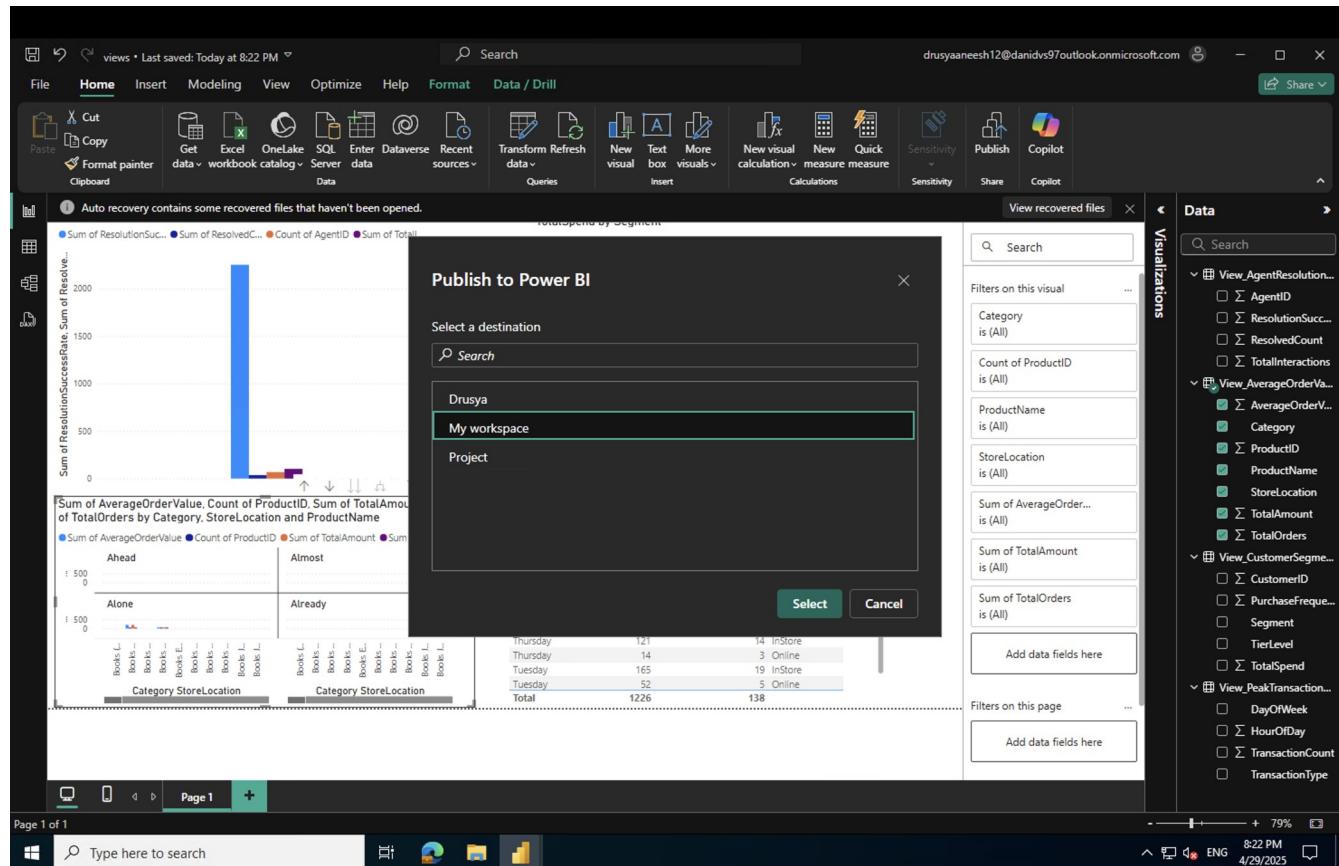


After connecting select the tables we want to visualize.

The screenshot shows the Power BI Desktop interface. In the top navigation bar, the 'Home' tab is selected. The 'Navigator' pane on the left lists various database objects, including tables and views from a 'sqlserverdrusya.database.windows.net' database. One view, 'View_AgentResolutionStats', is selected and highlighted in blue. A preview window on the right displays the content of this view with the message 'Preview is evaluating...'. The bottom status bar shows the date and time as 4/29/2025 at 8:18 PM.

The screenshot shows a report page in Power BI Desktop. It contains several visualizations: a bar chart, a pie chart, and a matrix table. The bar chart displays the 'Sum of ResolutionSuccessRate' across different categories. The pie chart shows the distribution of customer segments. The matrix table provides a detailed breakdown of transaction counts by day of the week and hour of the day. The right side of the screen features the 'Data' and 'Visualizations' panes, which list various data fields and visualizations respectively. The bottom status bar shows the date and time as 4/29/2025 at 8:22 PM.

Publish the report and save to fabric workspace.



Open Fabric and open the published file.

DayOfWeek	Sum of HourOfDay	Sum of TransactionCount	TransactionType
Friday	96	12	InStore
Friday	87	9	Online
Monday	145	14	InStore
Monday	68	6	Online
Saturday	124	13	InStore
Saturday	43	4	Online
Sunday	73	12	InStore
Sunday	5	2	Online
Thursday	121	14	InStore
Thursday	14	3	Online
Tuesday	165	19	InStore
Total	1226	138	

GIT HUB

The screenshot shows the Microsoft Azure Synapse Analytics workspace configuration interface. The left sidebar navigation menu includes:

- 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

The main content area is titled "Configure a repository". It displays the following configuration details:

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	521929bdaf6c812f8067f7feccb662cd096220a
Custom comment	Enabled

At the top of the main content area, there are several buttons: "Edit", "Overwrite live mode", "Disconnect", and "Import resources".