Data Flows: Low Code Data Transformation

Table of Contents

Data Flows

The generalized concept of Data Flows refers to creating a forward flow of data from raw data source to Gold layer of medallion architecture. For more details about medallion architecture, please refer to the section [APPENDIX C: Medallion Architecture]. It includes pipelines to ingest both the streaming as well as non-streaming data in delta tables. Data Flows are basically designed to allow engineers to develop transformation logic without writing single line of code.

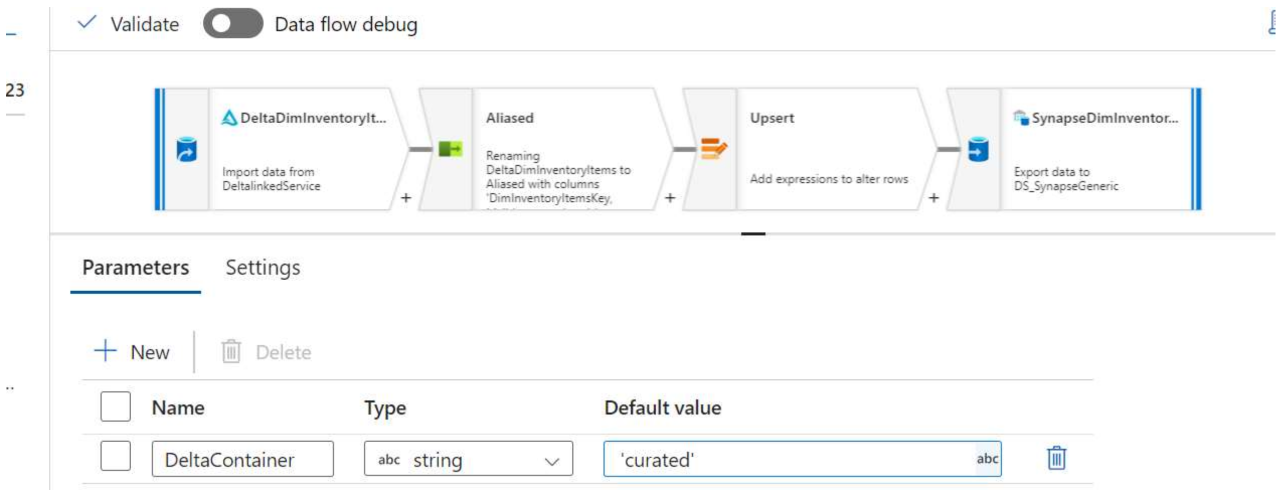
Azure provides Data Flows under Data Factory services and Power services like Power Platform, Power BI etc.

Power BI Data Flows

Data Flows in Power BI is a collection of tables that are created and managed in Power BI workspace. Tables are a state of column or group of columns which are used to store the data, much like a table within the database. It allows us to import data from different data sources with proper schema, different joins, merge operations or other transformations on the imported table to create final tables. Power BI dataflows can be operated via Power BI service only. Please refer to the Microsoft documentation [Link] for more details.

Azure Data Factory Data Flows

Sample ADF Data Flow


Validate
Data flow debug
23
DeltaDimInventoryIt...
Import data from DeltalinkedService
Aliased
Renaming DeltaDimInventoryItems to Aliased with columns 'DimInventoryItemsKey,
Upsert
Add expressions to alter rows
SynapseDimInventor...
Export data to DS_SynapseGeneric
Parameters
Settings
New
Delete

	Name	Type	Default value
☐	DeltaContainer	abc string	'curated'

Azure Data Flows majorly include following activities:

1. Data Flow Graph: Visualize the joins of tables and the flow of data.

2. Data Flow Configuration: Setup the column mappings of tables including primary keys, data types.
3. Transformation Settings: Data transformation logics which contains data source type, Data sink, transformations.
4. Optimize: It enables the creation of efficient data flows by configuring to the partitioning schemes.
5. Inspect: It provides a metadata preview of the stream that is being transformed. Column order, column datatype, column references etc. are displayed in preview for the selected transformation.

Features And Capabilities for Azure Data Flows

1. **Data Movement**
    a. Copy Activity: Move data between supported source and destination data stores.
    b. Data Flow: Transform and clean the data as it moves from source to destination.
2. **Data Transformation**
    a. Wrangling Data Flow: Visually prepare and transform data using a code-free environment.
3. **Orchestration**
    a. Pipeline Orchestration: Create and schedule data-driven workflows (pipelines) that orchestrate data movement and data transformation activities.
4. **Integration Runtimes**
    a. Azure Integration Runtimes: Execute data movement and data transformation activities across different network environments (e.g., on-premises, cloud, virtual network).
5. **Monitoring and Management**
    a. Azure Monitor: Monitor the health, performance, and usage of your data pipelines.
    b. Azure Data Factory Managed Portal: Web-based interface for designing, monitoring, and managing data pipelines.
6. **Security and Compliance**
    a. Azure Active Directory Integration: Integrate with Azure AD for authentication and authorization.
    b. Data Encryption: Ensure data security in transit and at rest.
7. **Data Flow Debugging**

   a. Debugging Data Flows: Identify and resolve issues in your data flow logic with debugging capabilities.

8. **Source and Destination Connectors**
   a. Built-in Connectors: Utilize a variety of built-in connectors for popular data sources and destinations (e.g., Azure Blob Storage, Azure SQL Database, Azure Data Lake Storage, on-premises SQL Server).

9. **Data Flow Monitoring**
   a. Data Flow Monitoring Features: Monitor the data flow executions and identify bottlenecks or performance issues.

10. **Integration with Other Azure Services**
    a. Integration with Azure Services: Integrate with other Azure services like Azure Synapse Analytics, Azure Machine Learning, etc., for comprehensive data processing and analytics.

11. **Data Lineage and Impact Analysis**
    a. Data Lineage**:** Understand and visualize the flow of data from source to destination.
    b. Impact Analysis: Assess the potential impact of changes to your data pipelines.

Benefits for Azure Data Flows

[Reference: Link]

1. **Ease of Use**
   a. Data Flows offer a user-friendly, drag-and-drop interface.
   b. Users can create and execute data transformations without the need for coding skills.

2. **Scalability**
   a. Data Flows are built on Azure Databricks, providing a scalable and distributed computing environment.
   b. Enables the processing of large datasets quickly and efficiently.

3. **Reusability**
   a. Allows the creation of reusable data transformation logic.
   b. This logic can be applied across multiple data sources, saving time and effort in pipeline development.

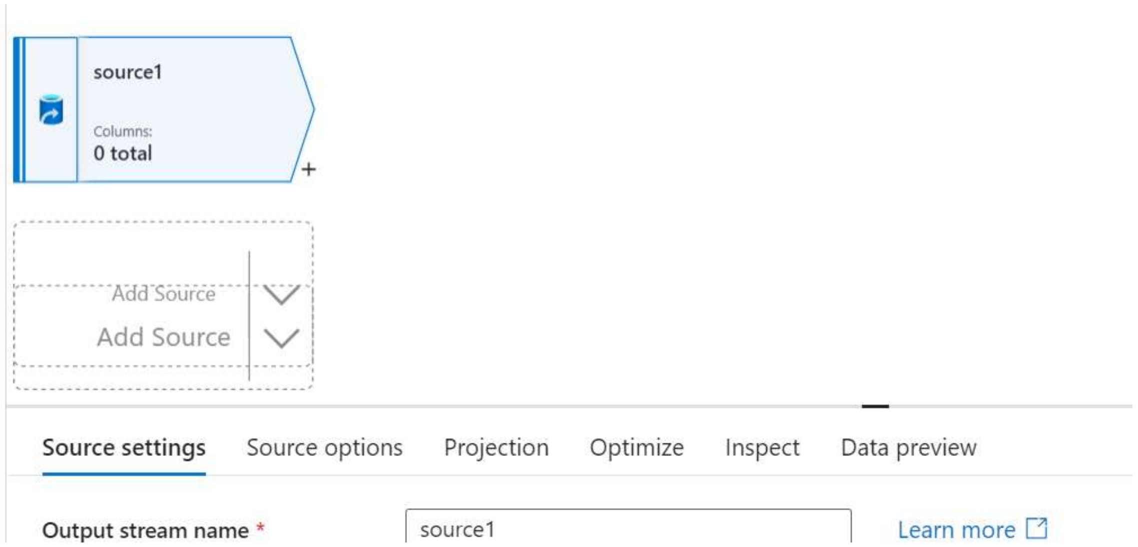
4. **Data Profiling**

    a. Includes built-in data profiling capabilities.
    b. Users can analyze and understand data before applying transformations, helping to identify and address data quality issues.

5. **Integration with other Azure services**
    a. Integrates seamlessly with Azure Synapse Analytics, Azure Data Lake Storage, and Azure SQL Database.
    b. Enables the building of end-to-end data pipelines across multiple Azure services.
6. **Code Generation**
    a. Data Flows generate Apache Spark code for executing data transformations.
    b. The generated code can be used for troubleshooting or automating the creation of Data Flows pipelines.
7. **Monitoring & Management**
    a. Provides a monitoring dashboard for real-time performance monitoring of Data Flows.
    b. Enables developers and data engineers to quickly identify and troubleshoot any issues.
8. **Data Transformation Capabilities**
    a. Offers a comprehensive set of data transformation capabilities.
    b. Includes filtering, sorting, aggregating, pivoting, and joining data from multiple sources.
    c. Facilitates the transformation of data into the desired format for downstream analytics and reporting.

APPENDIX A: Data Flow Components

Data Source Setting



Tab: Source Setting

1. Output Stream Name: Meaningful name for the flow step representing purpose of task.

2. Description: Description of the flow task.

3. **Source Type**
    a. Dataset
        i. Represents a dataset pre-created in advance.
        ii. Connects to allowed data sources in Azure Data Factory (ADF) or Azure Synapse, including ADLS Gen2, Oracle, AWS S3, etc.
        iii. Serves as a named view abstracting the physical format and location of the data.
        iv. Represents the data structure, schema, and location, whether in files (e.g., CSV, Parquet) or database tables.
        v. Compatible with a variety of data sources, ensuring flexibility in connectivity.
    b. Inline
        i. In ADF or Synapse, "inline" refers to inline data within the pipeline definition.
        ii. Inline data involves providing data directly within the pipeline.
        iii. This approach is particularly useful for small, static datasets.
        iv. Eliminates the need for external storage when incorporating data directly into the pipeline.
        v. Provides a convenient method for handling data within the pipeline without relying on external repositories.

4. Sampling: If enabled, this option allows to select sample of the dataset for testing purpose. It asks for **Row Limits** representing how much row counts should be selected in the sample.

Tab: Source Options

1. Input: This option allows to select data either from a Table or via a Query or using a Stored Procedure.

2. Change Data Capture: This option allows to ingest only new data using CDC column.
    a. If an incremental data column is already defined then we can select Incremental Column and based on that allow data flow to ingest data incrementally
    b. Select SQL Server based CDC if it is enabled.

3. Run Option
    a. Based on CDC selected it allows to ingest incremental every time or full ingest first time and then incremental ingest.

    b. **Incremental Column**
        i. Purpose: The **Incremental Column** approach is used to detect changed records automatically during data flows. It applies not only to databases but also to files.

        ii. How It Works?
            1. When extracting data, the system uses a designated **incremental column** to identify changes.
            2. For files, the **last modified timestamp** is often used to detect changes.
            3. You select a specific column (the **incremental column**) that will be used to track changes.
            4. When developer wants to extract **delta data** (i.e., only the changed records), the system compares the values in this column to determine what has been added, updated, or deleted.

        iii. Use Case: This approach is particularly useful when you need to perform incremental data loads, ensuring that only the changed data is processed.

    c. SQL Server Change Data Capture (CDC)
        i. Purpose: CDC is a technology supported by data stores such as **Azure SQL Managed Instances (MI)** and **SQL Server**.

        ii. How It Works?
            1. CDC captures changes made to tables (inserts, updates, and deletes).

2. It maintains a record of these changes in special CDC tables.
3. These tables store information about the changed rows, including the type of change (insert, update, or delete) and the timestamp.
4. Azure Data Factory leverages CDC to identify and extract only the changed data.

iii. Use Case: CDC is commonly used for **incremental data extraction** scenarios, where you want to load only the data that has changed since the last extraction.

4. Schema Name: It refers to the name of schema in which table is present.

5. Table Name: Name of the table to ingest data.

6. Isolation Level: It provides Isolation levels for SQL server read.
    a. Read Uncommitted**:** Allows transactions to read uncommitted and potentially inconsistent data, providing the highest concurrency.
    b. Read Committed: Transactions can only read committed data, balancing concurrency, and consistency in database transactions.
    c. Serializable: Highest isolation level, ensuring transactions are completely isolated, minimizing concurrency for maximum data consistency.
    d. None: For no isolation applied.

Tab: Projection

1. This tab allows to see the mapping of columns of source table selected. Including column name, column datatype. Data Flows also allow us to import projections when the source is selected.
2. It provides two more options to **Clear Schema** or **Schema Options** to modify the schema mapping detected from the table.

Tab: Optimize

This window allows partitioning options for optimization.

1. **Use Current Partitioning**

    a. Purpose: By default, this option is selected. It instructs the service to keep the current output partitioning of the transformation.

    b. How It Works?

        ▪ When reading data from a source (except for Azure SQL Database), data flows automatically partition the data evenly based on the size of the data.

        ▪ A new partition is created for approximately every **128 MB** of data.

        ▪ As your data size increases, the number of partitions also increases.

- Custom partitioning happens after Spark reads in the data and can negatively affect data flow performance.

c. Recommendation: Use this option in most scenarios unless you specifically need to repartition your data after certain transformations.

- Repartitioning data takes time, so it's generally recommended to stick with the current partitioning.
- Custom partitioning should be considered only if you understand the shape and cardinality of your data well.

2. **Single Partition**

a. Purpose: This option groups all the results into a single output group.

b. Use Case: Rarely used, as it's almost certainly **not** recommended.

c. Example: If you're writing data to a single file, you might choose this option.

d. Caution: Be cautious when using this option, especially for large datasets, as it can lead to performance bottlenecks.

3. **Set Partitioning**

a. Purpose: Allows to re-group the sink data based on a **key column**.

b. How It Works?

- You can specify a column that defines the partitioning scheme for your data.
- Useful when you want to control how data is grouped in the output.

c. Use Case: When you need to customize the partitioning behavior based on specific business requirements.

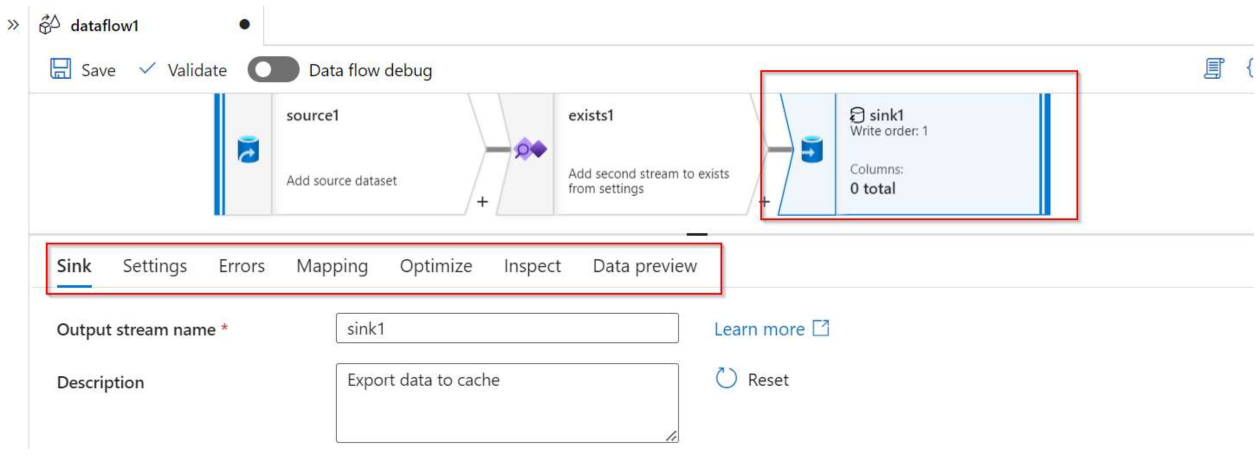
Tab: Inspect

This tab allows developers to inspect the schema again. Please refer to the section [APPENDIX B: Creating a Simple Data Flow in ADFAPPENDIX B: Creating a Simple Data Flow in] step 14 for sample snapshot.

Tab: Data Preview

Data to be ingested can be previewed here for column/value mappings.

Sink



Output stream name * sink1

Description Export data to cache

Tab: Sink

1. Output Stream Name: Meaningful name for the flow step representing purpose of sink.

2. Description: Description of the tasks.

3. Incoming Stream: Select one of the Source streams defined in the flow which will be input for sink activity.

4. Sink Type: It offers type **Dataset**, **Inline** & **Cache**. For Dataset and Inline details, refer to the section [Tab: Source Options] for more details.

    a. Cache Sink

        i. Purpose: The cache sink is used to store intermediate results during data flow execution. It acts as an in-memory cache, improving performance by avoiding redundant computations.

        ii. How It Works?
            1. On adding a cache sink, it stores the data in memory (within the data flow).
            2. Unlike other sink types (which write to external stores), the cache sink doesn't require selecting a dataset or linked service.
            3. One can use the cache sink to store data temporarily and then perform further transformations on it.

        iii. Key Columns
            1. Optionally, developers can specify **key columns** for the cache sink. These columns define the uniqueness of records stored in the cache.
            2. Key columns help optimize lookups and joins within the data flow.

   b. **Use Cases**
      i. Cached Lookups: You can use the cached data for efficient lookups using functions like lookup, mlookup, and output.
      ii. Intermediate Results: Store intermediate results during complex transformations to avoid recomputing them.
      iii. Parameterized Sinks: If your sink is heavily parameterized, the cache sink allows you to avoid creating unnecessary “dummy” objects.

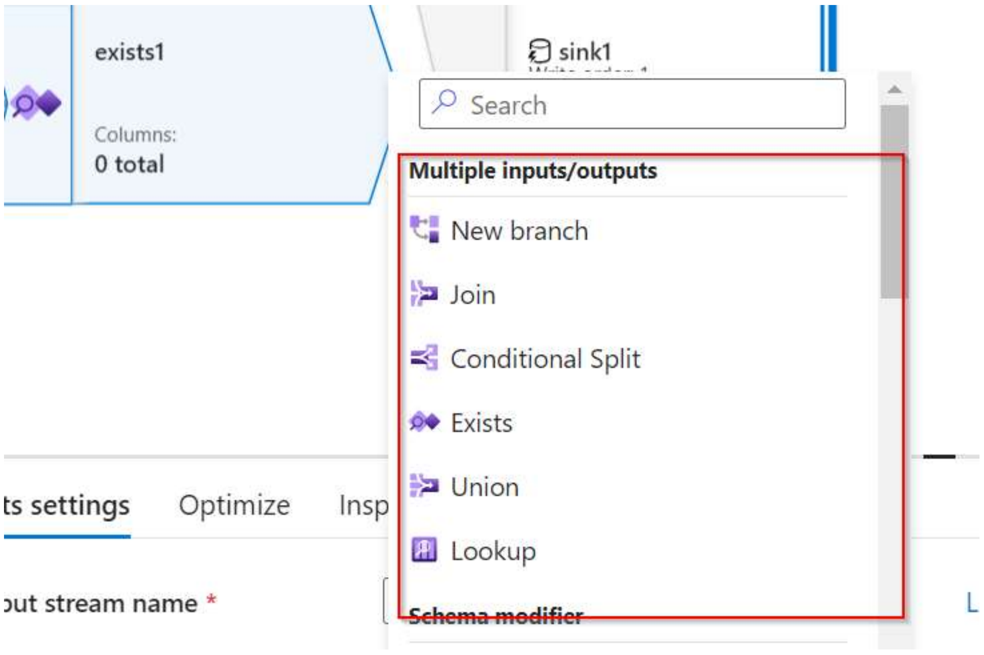
   c. **Workspace DB (Synapse Workspaces Only)**
      i. In Azure Synapse workspaces, you have an additional option to sink data directly into a database type within your Synapse workspace.
      ii. This eliminates the need for additional linked services or datasets for those databases.

5. Options: Based on selected Sink Type, it provides **Options.**
   a. If Sink Type is Dataset or Inline, it provides two options. **Allow Schema Drift** option allows flexibility if Columns are changing very frequently. Opposite to this, **Validate Schema** option will cause data flow to fail if any column and type defined in the projection does not match the discovered schema of the source data.

   b. If Sink Type is Cached, then it provides **Write to Activity Output** options.
      i. The cached sink can optionally write your output data to the input of the next pipeline activity.
      ii. This allows to quickly and easily pass data out of your data flow activity without needing to persist the data in a data store.
      iii. It’s a convenient way to streamline data flow execution and pass intermediate results to subsequent activities.

Activities

Data Flows permits different activities between Source and Sink.



Multiple Input/Outputs

1. New Branch: It allows to parallelly process single source data in two different processing tasks.
2. Join: It allows joins between two sources. Please refer to section [APPENDIX B: Creating a Simple Data Flow in ADF] for more details.
3. Conditional Split: It allows branching but based on a condition.
4. Exists: It allows filtering of rows based on a value.
5. Union: It allows union of two sources.
6. Lookup: It allows to lookup values based on another source. Similar activity to left outer join.

Schema Modifiers

1. **Aggregate**: Define various aggregations (e.g., SUM, MIN, MAX, COUNT) grouped by existing or computed columns.

2. **Cast**: Change column data types with type checking.

3. **Derived Column**: Generate new columns or modify existing fields using the data flow expression language.

4. **Pivot**: Transform distinct row values into individual columns.

5. **Rank**: Generate ordered rankings based on sort conditions.

6. **Select**: Alias columns, reorder, and drop unwanted columns.

7. **Surrogate Key**: Add an incrementing non-business arbitrary key value.
8. **Unpivot**: Pivot columns into row values.
9. **Window**: Define window-based aggregations of columns in your data streams.

Row Modifiers

Alter Row: Set insert, delete, update, and upsert policies on rows.

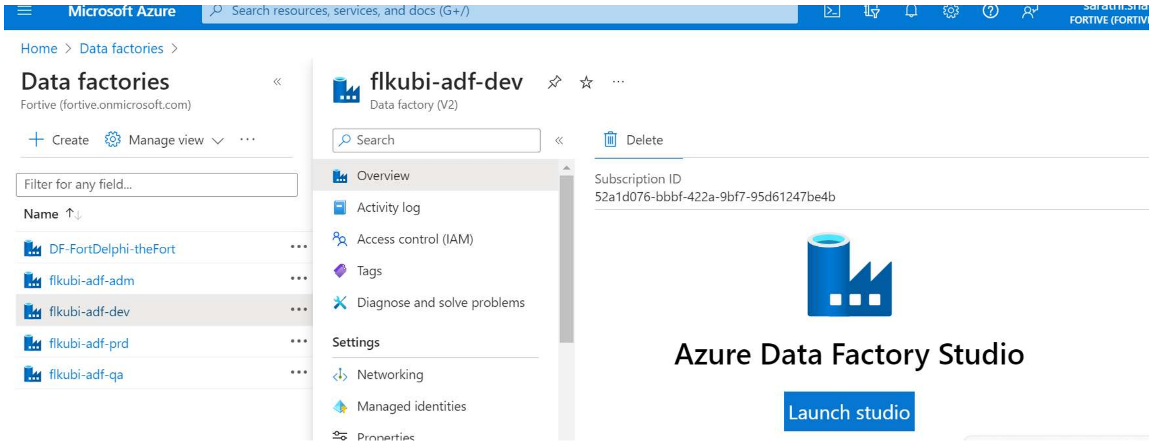
1. **Filter**: Filter rows based on conditions.
2. **Sort**: Sort incoming rows.
3. **Assert:** To assert some conditions for values type enforcements.

Formatters

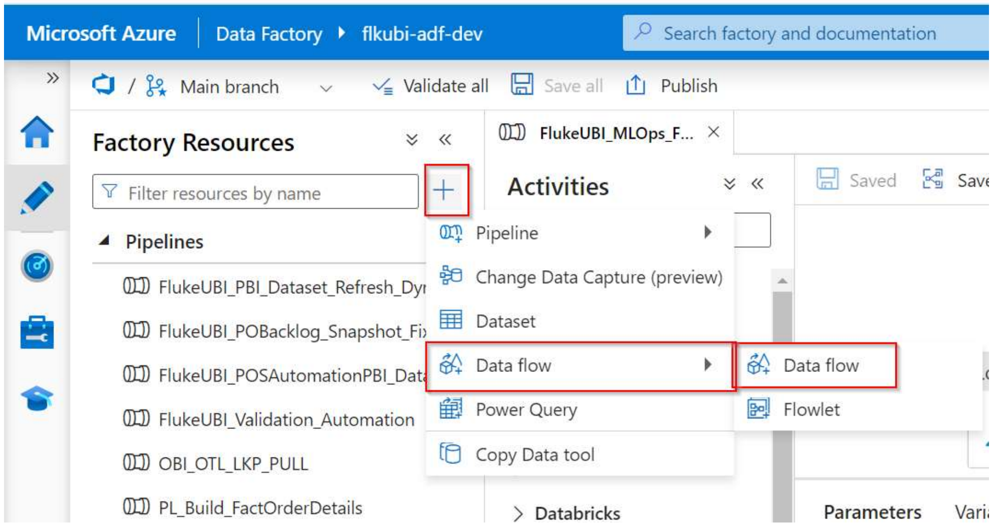
1. **Flatten**: Unroll array values inside hierarchical structures (e.g., JSON) into individual rows.
2. **Stringify**: Turn complex types into plain strings.

APPENDIX B: Creating a Simple Data Flow in ADF

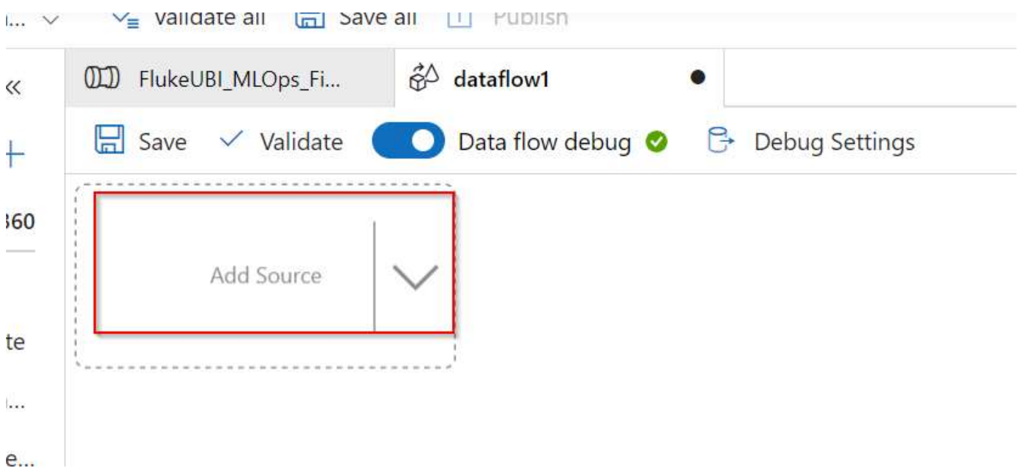
1. Open the Azure Data Factory, click on the Data Flow.



2. Click on '+' -> Data Flow -> Data Flow.

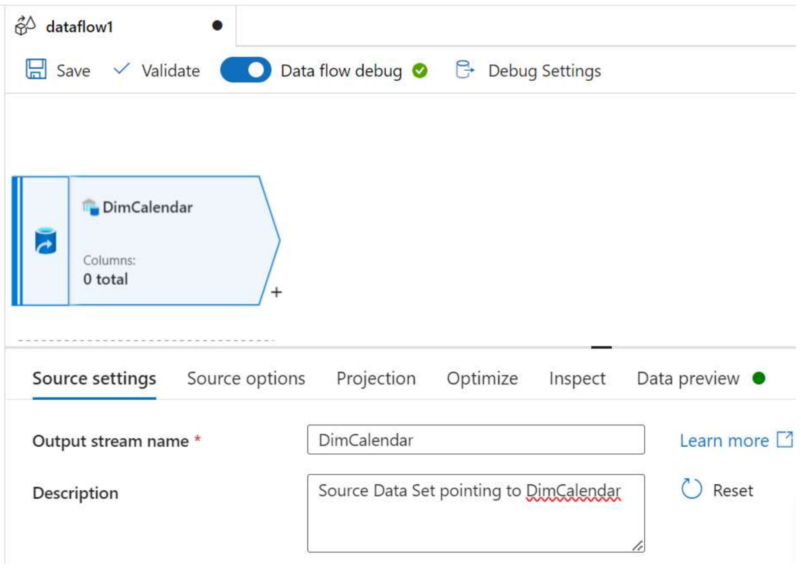


3. Enable the Debug Mode as shown in the picture below.
4. Click on Add Data Source to configure a data source for the flow.



Get Dim Calendar

5. Update in Source Settings tab.
    a. Add Flow activity name and description.
    b. Select the Synapse Dataset to query on the Synapse table.
    c. Keep all other values of that tab as it is.



Output stream name * DimCalendar

Description Source Data Set pointing to DimCalendar

6. Update in the Source Options tab.
    a. SELECT query in **Input.**
    b. Add following query in the query section.
        i. SELECT * FROM [ubidw].[vw_DimCalendar]
        ii. Click on Import Projections option. It will detect the metadata (ColumName and Datatypes) and import it. It can be reviewed in the **Projection** tab.
    c. Keep other values as it is.
7. Under the Optimize tab,

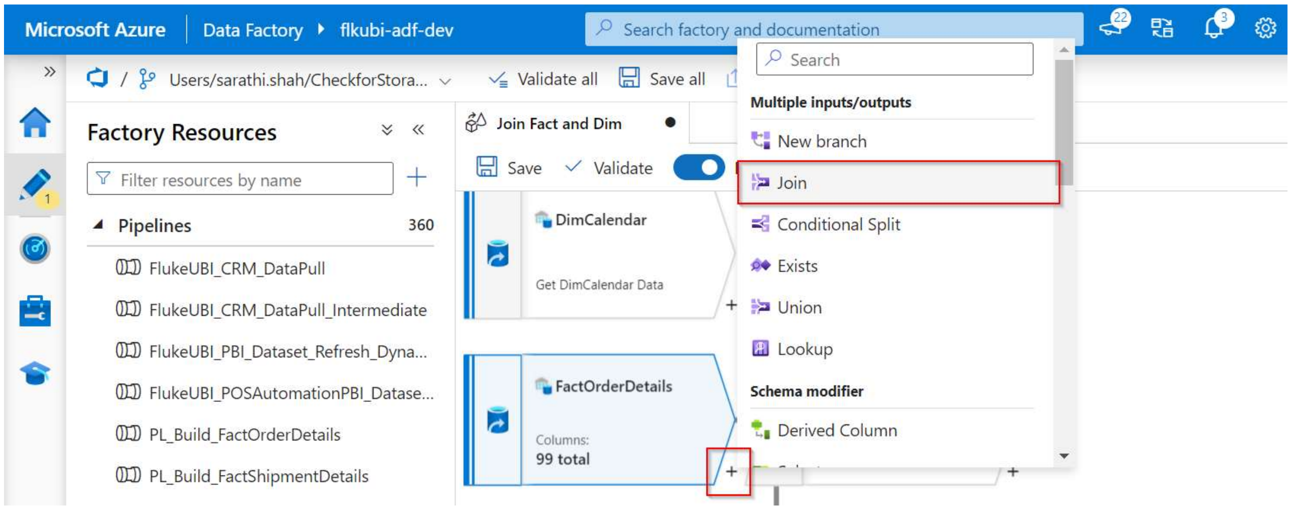
        a. Keep the Single Partition selected.
8. Under the Inspect tab,
    a. Review the column name and mapping.
9. Under the Data Preview tab,
    a. Verify mapping of columns and their values with the datatypes detected.

Get LatestIDTable

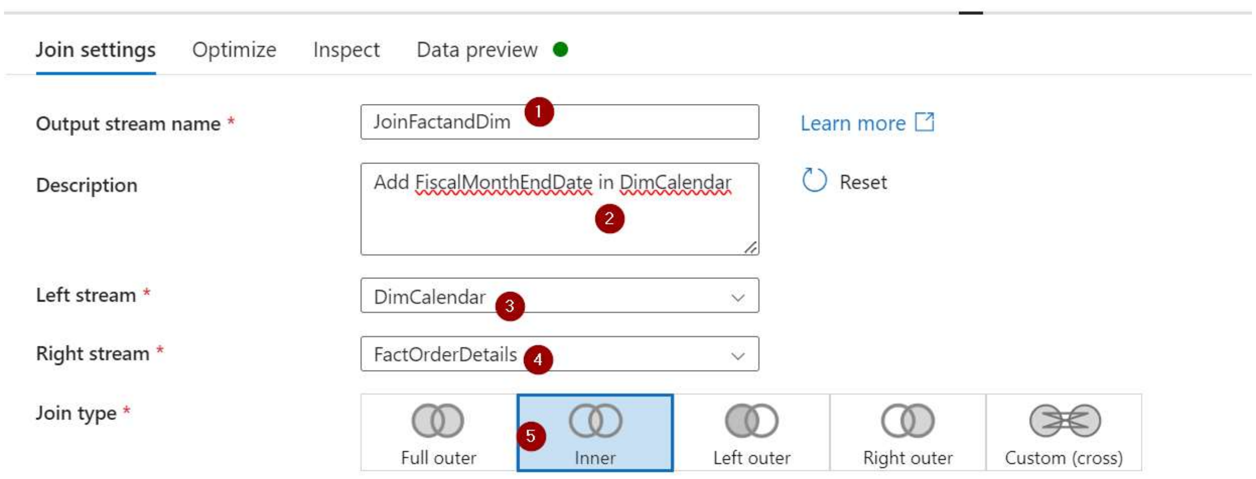
10. Repeat steps 5-10 to add one more data source for FactOrderDetails table. Use the following query for the table.
    a. SELECT TOP 10 * FROM [ubidw].[FactOrderDetails]

Join and Add Fiscal Month End Date Column

11. Click on '+' mark after the LatestID Table activity and click on joins from the opened menu.



12. Update the details in 'Join Settings' tab.
    a. Add Output Stream name. (i.e., AddFiscalMonthEndDate)
    b. Add Description.
    c. Select table on the left side of the join. (i.e., ActivityDate)
    d. Select table on the right side of the join. (i.e., FactOrderDetails)
    e. Select join type as 'Inner'.



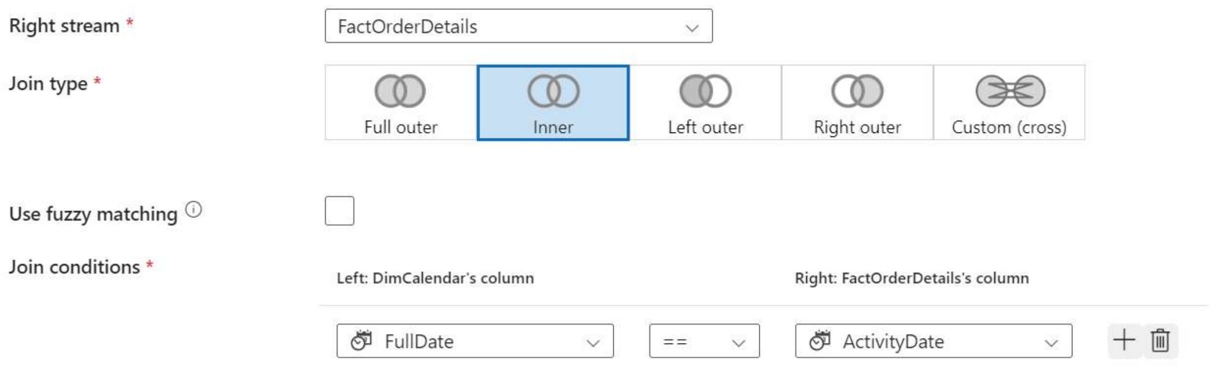
Output stream name * JoinFactandDim

Description Add FiscalMonthEndDate in DimCalendar

Left stream * DimCalendar

Right stream * FactOrderDetails

Join type * Full outer [✓ Inner] Left outer Right outer Custom (cross)



Right stream * FactOrderDetails

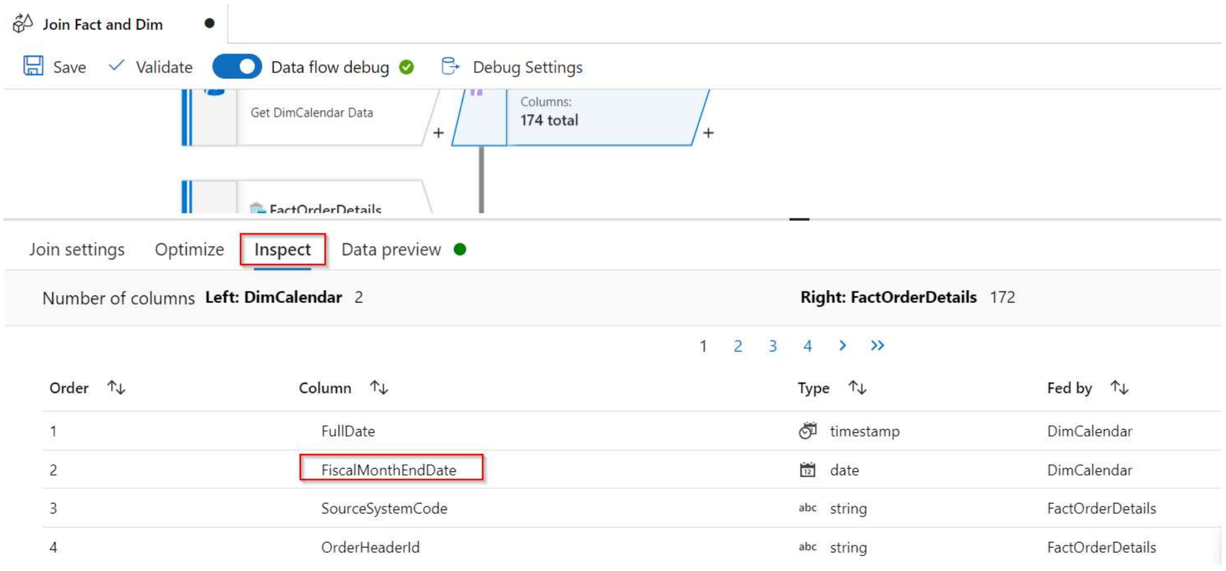
Join type * Full outer [✓ Inner] Left outer Right outer Custom (cross)

Use fuzzy matching ☐

Join conditions *

Left: DimCalendar's column		Right: FactOrderDetails's column
FullDate	==	ActivityDate

13. Select Join Conditions.
    a. Select **FullDate** for **DimCalendar**.
    b. Select **ActivityDate** from **FactOrderDetails**
    c. Note: Selected columns must be having same datayes
14. Under the Inspect tab, view the Join output columns and datatypes.


Join Fact and Dim

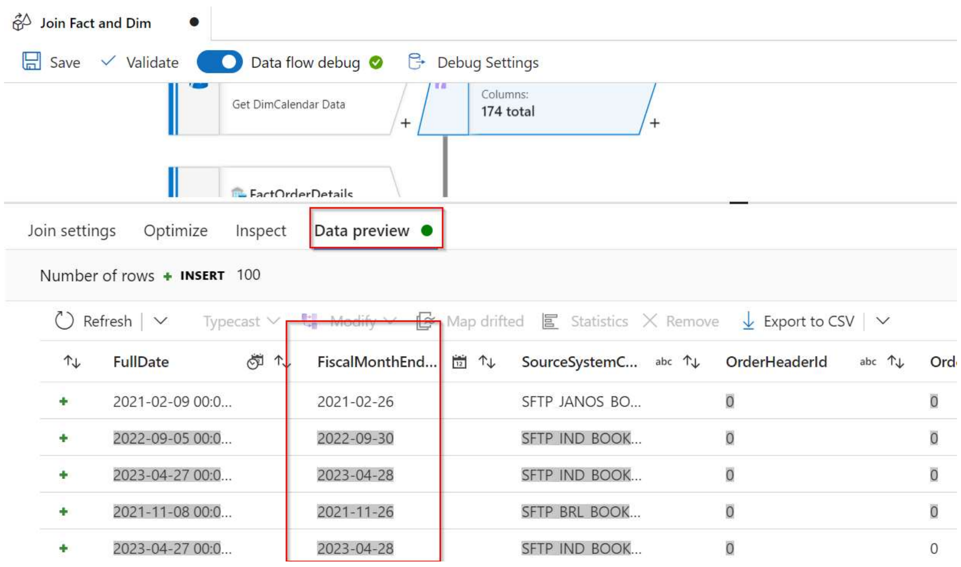
Save | Validate | Data flow debug | Debug Settings

Join settings | Optimize | Inspect | Data preview

Number of columns **Left: DimCalendar** 2 **Right: FactOrderDetails** 172

Order	Column	Type	Fed by
1	FullDate	timestamp	DimCalendar
2	FiscalMonthEndDate	date	DimCalendar
3	SourceSystemCode	string	FactOrderDetails
4	OrderHeaderId	string	FactOrderDetails

15. Under the Data Preview tab, review the joined outputs. Verify the values of FiscalMonthEndDate column.


Join Fact and Dim

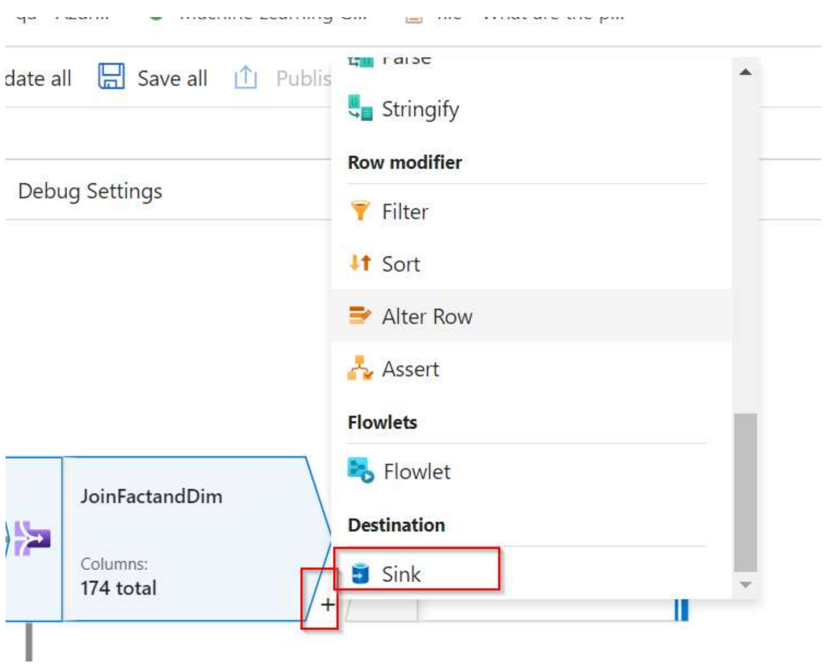
Save | Validate | Data flow debug | Debug Settings

Join settings | Optimize | Inspect | Data preview

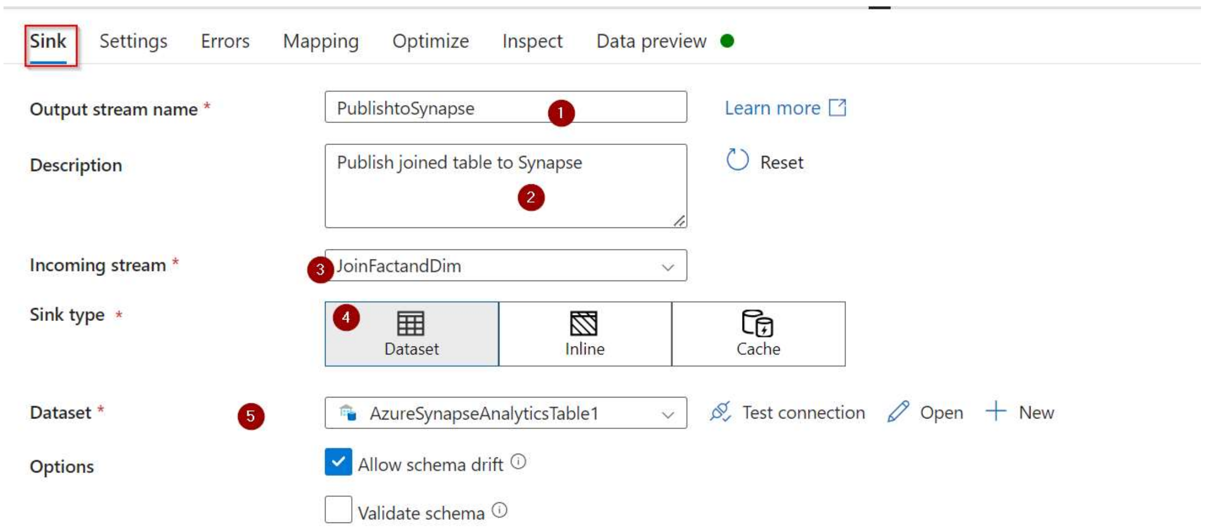
Number of rows + INSERT 100

	FullDate	FiscalMonthEnd...	SourceSystemC...	OrderHeaderId	Ord
+	2021-02-09 00:0...	2021-02-26	SFTP JANOS BO...	0	0
+	2022-09-05 00:0...	2022-09-30	SFTP IND BOOK...	0	0
+	2023-04-27 00:0...	2023-04-28	SFTP IND BOOK...	0	0
+	2021-11-08 00:0...	2021-11-26	SFTP BRL BOOK...	0	0
+	2023-04-27 00:0...	2023-04-28	SFTP IND BOOK...	0	0

16. Now click on the '+' icon just after join activity and select on **Sink.**



17. Configure the sink.
    a. Add **PublishtoSynapse** as Output Stream Name.
    b. Add **Publish joined table to Synapse** in description.
    c. Select **JoinFactandDim** and Incoming Stream.
    d. Select the dataset for Synapse Connection.



Output stream name * PublishtoSynapse

Description Publish joined table to Synapse

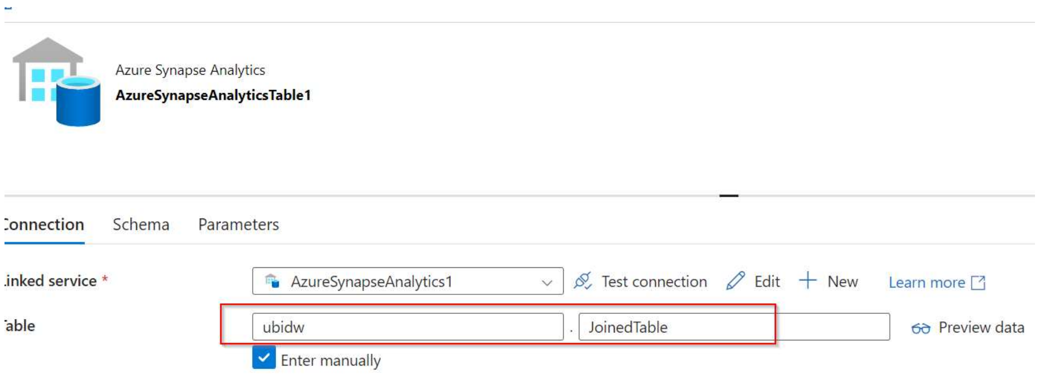
Incoming stream * JoinFactandDim

Sink type * Dataset

Dataset * AzureSynapseAnalyticsTable1

Options
- [x] Allow schema drift
- [ ] Validate schema

18. Verify that selected dataset in previous step is having proper table name for publish.


Azure Synapse Analytics
AzureSynapseAnalyticsTable1

Connection | Schema | Parameters

Linked service * : AzureSynapseAnalytics1

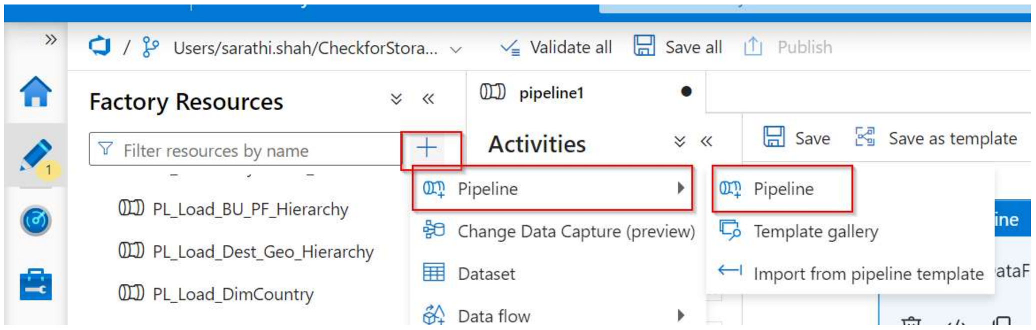
Table: ubidw . JoinedTable

☑ Enter manually

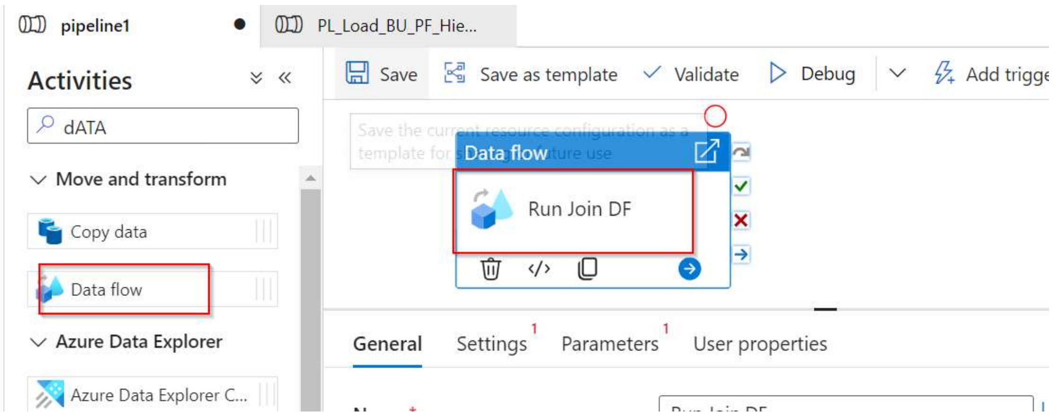
19. Under the **Setting** tab, keep everything as it is.
    a. One can add Pre-Copy and Post-Copy script if required in this tab.
20. Under the Mapping tab,
    a. Verify that Skip duplicate input column is selected.
    b. Verify that Skip duplicate output column is selected.
    c. This selection will skip the duplicate columns, keeping only a single copy to satisfy Synapse constraints.
21. Under the Inspect tab,
    a. Verify the columns and datatypes.
22. Under the data preview tab,
    a. Verify the data for publish.
23. Click on **Save** to create it.

Run the Data Flow

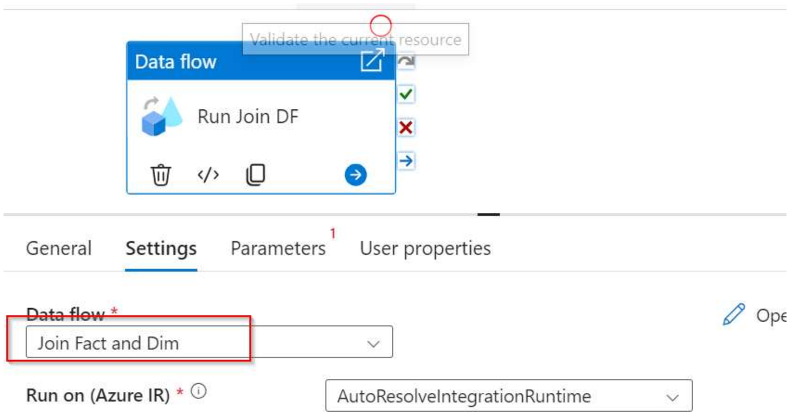
1. To run the Data Flow we need to trigger it through a data pipeline.
    a. Create a pipeline and call a flow from there.



    b. Add an activity of **DataFlow**.
        i. Now under the general tab, add the name of the flow as **Run DataFlow.**



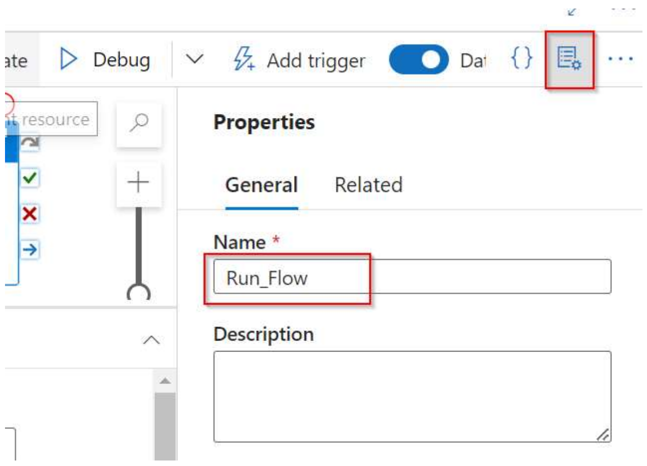
c. Now, under the settings tab, select the Data Flow.
   i. Below that, select Staging linked service.
   ii. Select storage path for sink.



Data flow * Join Fact and Dim

Run on (Azure IR) * AutoResolveIntegrationRuntime

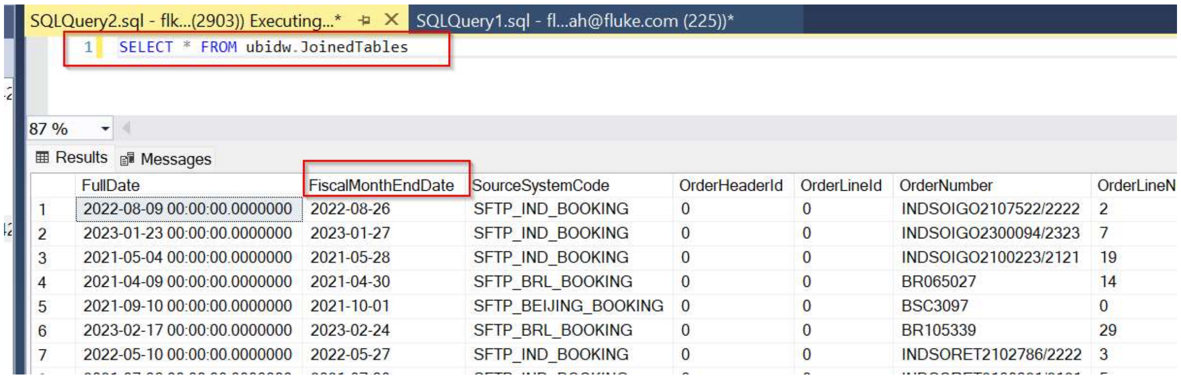
d. Click on Save to save the pipeline.
   i. Rename the pipeline with proper name.



Name * Run_Flow

Description

e. Click on debug to run it.
f. From the pipeline activity runs, we can monitor the run status.
g. Once run is completed, we can validate the table publish on synapse using following query.
    i. SELECT * FROM [ubidw].[JoinedTables]


SQLQuery2.sql - flk...(2903)) Executing...*
SQLQuery1.sql - fl...ah@fluke.com (225))*
87 %
Results
Messages

```
SELECT * FROM ubidw.JoinedTables
```

	FullDate	FiscalMonthEndDate	SourceSystemCode	OrderHeaderId	OrderLineId	OrderNumber	OrderLineN
1	2022-08-09 00:00:00.0000000	2022-08-26	SFTP_IND_BOOKING	0	0	INDSOIGO2107522/2222	2
2	2023-01-23 00:00:00.0000000	2023-01-27	SFTP_IND_BOOKING	0	0	INDSOIGO2300094/2323	7
3	2021-05-04 00:00:00.0000000	2021-05-28	SFTP_IND_BOOKING	0	0	INDSOIGO2100223/2121	19
4	2021-04-09 00:00:00.0000000	2021-04-30	SFTP_BRL_BOOKING	0	0	BR065027	14
5	2021-09-10 00:00:00.0000000	2021-10-01	SFTP_BEIJING_BOOKING	0	0	BSC3097	0
6	2023-02-17 00:00:00.0000000	2023-02-24	SFTP_BRL_BOOKING	0	0	BR105339	29
7	2022-05-10 00:00:00.0000000	2022-05-27	SFTP_IND_BOOKING	0	0	INDSORET2102786/2222	3

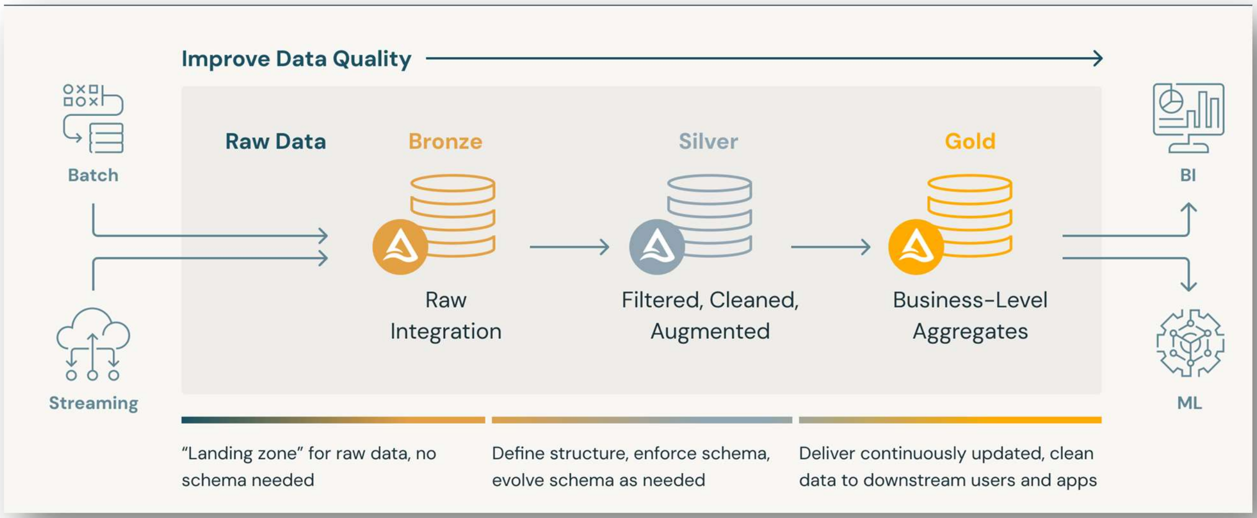
APPENDIX C: Medallion Architecture

[Reference: Link]

Medallion architecture is a data design pattern used to logically store the data in a Lakehouse.

The architecture separates the data based on their quality and the structure of data.

Please refer to the following image.



Improve Data Quality
Batch
Streaming
Raw Data
Bronze
Silver
Gold
Raw Integration
Filtered, Cleaned, Augmented
Business-Level Aggregates
BI
ML
"Landing zone" for raw data, no schema needed
Define structure, enforce schema, evolve schema as needed
Deliver continuously updated, clean data to downstream users and apps

The architecture defines three main layers.

1. **Bronze:** Contains raw data, or slightly processed data.
2. **Silver:** Contains processed data.
    a. Business / Processing logics are defined on the Bronze layer tables, to generate valuable informative data.
    b. This layer defines the base data for reporting of any analytics tasks.
    c. Operations like data filtering, augmentation, cleaning are performed in Silver layer.
3. **Gold**
    a. It defines more fine-grained data that can directly be consumed in reporting.
    b. Only required columns are selected from Silver layer. Different aggregations are performed to generate required grain-ed data.
    c. Different views creation is majorly considered under Gold layer.

APPENDIX D: Full Table of Content

- Ensuring data privacy and compliance
- Security best practices

8. **Optimizing Data Flows**
    - Performance tuning
    - Monitoring and troubleshooting
    - Optimization best practices
9. **Use Cases and Scenarios**
    - Real-world examples of using Azure Data Flows
    - Industry-specific use cases
10. **Integration with Power BI and Other Services**
    - Using Data Flows with Power BI
    - Integration with other Azure services
11. **Best Practices and Tips**
    - Design principles
    - Tips for efficient data processing
    - Lessons learned from practical implementations
12. **References and Additional Resources**
    - Links to official documentation
    - External resources for further learning
    - Community forums and support channels