# Extending Analytics for LS Central

Analytics 2026.1 and later versions

Using bc2adls extension

# Contents


**1 Introduction ..... 4**

**2 Initial setup ..... 5**

**3 Add Company ..... 6**

**4 The bc2adls extension ..... 7**

**5 Staging ..... 8**

Staging with bc2adls ..... 8

Why schema export is required ..... 8

Prestaging vs staging tables ..... 8

Adding new tables or columns ..... 8

Running Initial Load vs Factory Reset ..... 9

Important note about metadata regeneration ..... 9

Adding new staging table to Source Tables Map ..... **Error! Bookmark not defined.**

**6 Dimensions ..... 12**

Add new column to existing dimension ..... 12

Add column to dimension 12

Add new dimension ..... 15

**7 Facts ..... 17**

Add new column from staging table to existing fact table ..... 17

Add column to fact table 17

Modify stored procedure 17

Add new fact table ..... 19

Add new pipelines to ADF ..... 20

Modules in scheduled run 20

Structure of modules 21

**8 Third-party data ..... 23**

Open the Azure Data Factory studio ..... 23

Staging data ..... 23

Create a copy pipeline 23

Add Pipeline variables 24

Copy Activity 25
Create a new source connection 25
Connect to your data 27
Set the pre-copy script 27
Add Write to audit table action 28
Update AnalyticsAudit on success 29
Update AnalyticsAudit on failure 30
Large data source 30
Add staging pipeline to scheduled run 31
Add fact table to star schema 33
Create a connected fact table 33
Create a Stored Procedure 33
Add a fact table Stored Procedure to ADF pipeline 35
Add data directly in Power BI 39
Get a new source 39
Merge to get surrogate keys 39
Expand merged tables 40
Remove unnecessary columns 41
M query41
Define relationships in PowerBI 42
Use the new data in a report page 43

# 1 Introduction

This document describes how to extend the Analytics data warehouse and reports.

Analytics has always been thought of as a product that provides a BI base for our customers that currently have no BI solution in place.

With Analytics, we provide a basic extendable data warehouse with SQL server database and Power BI reports and measures that work with the warehouse's data structure.

We have therefore always encouraged extending Analytics in any way that suits the customer and have now decided to provide more detailed guidelines and examples of how to extend Analytics with data from LS Central base or extensions and third-party data.

In Analytics version 2026.1 we added support for using the data from the bc2adls extension which gives us huge performance benefits over the old replication methods. It should also provide better handling of staging data, scale much better with number of companies added to Analytics and provide a simpler setup of exporting data from SaaS deployments.

# 2 Initial setup

The initial setup of Analytics is described in detail in the onboarding documentation in the Analytics section the LS Central online help and will not be explained here.

There are also several LS Retail Academy courses on how to setup Analytics for different LS Central and Analytics platforms.

# 3 Add Company

In the initial setup you select which companies you want to load to Analytics.

If you want to add a new company to your Analytics setup at a later stage, that is easy to do.

You can add to the **Analytics$Companies** table using the **Add or Delete Companies** pipeline in the Analytics ADF.

When you add the trigger for the pipeline you are prompted with several fields:

- Companies (Company name as it is on the Companies page in LS Central).
    - You can add/delete several companies simultaneously by separating them with a comma.
- Delete Companies (default setting is false, but if set to true the entered companies will be deleted).

Once you have added/deleted the companies you want to include/remove from , you run the **Scheduled Run**. Data from the companies you added will be added to the staging, dimension, and fact tables when the **Scheduled Run** pipeline runs. You could also run the **Factory reset** pipeline if you prefer.

In the bc2adls version it is very important that all companies that are being exported by the bc2adls extension are registered in Analytics, otherwise you will receive errors when trying to insert NULL values into the Company column.

# 4 The bc2adls extension

We use the bc2adls extension to export data from LS Central into a storage account in Azure. The reason we use this extension is because it's simple to setup, performs very well and makes it very easy to export only the changes between each run which should make our staging process a lot more efficient.

The extension is maintained and supported by Bert Verbeek and you can find further information on his github page: https://github.com/Bertverbeek4PS/bc2adls

This guide will not go into details regarding how the extension works, but there are a few things that you should be aware of.

- All files exported by the bc2adls extension are stored in the Azure Storage Account defined during deployment.
  - The cdm.json schema files are created when you run the schema export in the extension. The cdm.json are stored in the root folder of the storage account.
  - The delta export files are stored in the deltas folder each table has its own subfolder. If you are exporting data for multiple companies, there will be one file per company but they will share the table folder.
- After the data from the delta table has been copied into the prestaging tables in Analytics, the delta file is deleted.

# 5 Staging

Staging tables in Analytics are each based on cdm.json file produced by the bc2adls extension in LS Central. The cdm.json file includes selected columns from the base table plus columns from the $ext companion table. The base table can have its origin in Business Central or be created by an app like LS Central.

## Staging with bc2adls explained

### Staging table metadata and table creation process

The prestaging and staging tables used by Analytics are created automatically during the Initial Load and Factory Reset pipelines. The SQL statements required to create and maintain these tables are stored in the metadata table: dbo.AnalyticsBC2ADLSMetadata

This table contains, per source table:

- Prestaging table name
- Staging table name
- SQL to create both tables
- SQL to create indexes
- SQL to merge prestaging into staging
- SQL to drop tables (used during resets)

The metadata table is populated by the PopulateQueryBase pipeline in Azure Data Factory. This pipeline reads the cdm.json schema files produced by the bc2adls extension during schema export, and generates the correct SQL scripts for each table and column.

### Prestaging vs staging tables

Prestaging tables are used only as a temporary landing zone for incoming delta files exported from bc2adls. Delta data is first loaded into the prestaging table. The prestaging data is then merged into the staging table using the merge SQL stored in the metadata. After a successful merge, the prestaging table is truncated. This allows the system to load deltas efficiently while keeping the staging table as the stable “latest version” dataset.

Staging tables contain the full combined dataset (base + extensions) after the merge process has completed. These staging tables are then used as the input for dimension and fact table processing.

### Why schema export is required when making changes to configuration

The schema export is required because bc2adls uses it to generate the cdm.json files, which define:

- available tables
- available fields/columns
- column names and datatypes
- structure required for the staging SQL generation

Without an updated schema export, the Analytics pipelines cannot correctly generate the staging/prestaging table structures, meaning new tables or columns will not be created in SQL.

### Running Initial Load vs Factory Reset

**Initial Load** is run when the system is being loaded for the first time. The pipeline will:

- regenerate the metadata in AnalyticsBC2ADLSMetadata
- create prestaging and staging tables
- load exported data into prestaging
- merge into staging
- populate dimension and fact tables

Use **Factory Reset** if the solution has already been running, but you need to fully rebuild the environment. The Factory Reset pipeline will:

- regenerate the metadata in AnalyticsBC2ADLSMetadata
- drop and recreate staging and prestaging tables
- delete/truncate data from dimension and fact tables
- reload and rebuild all data from scratch

During Factory Reset, reports should still work because Power BI uses Import mode, but partners should avoid refreshing Power BI datasets until the pipeline has finished.

### Important note about metadata regeneration

Both Initial Load and Factory Reset will truncate and rebuild the AnalyticsBC2ADLSMetadata table.

This ensures that the metadata always matches the most recent schema export and prevents inconsistencies when new columns or tables have been added.

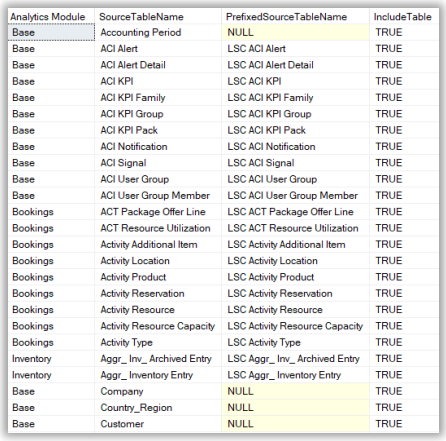
## Adding new tables or columns to staging

### 1. Change configuration and export from LS Central

If you want to add new tables or new columns into staging, the process is always:

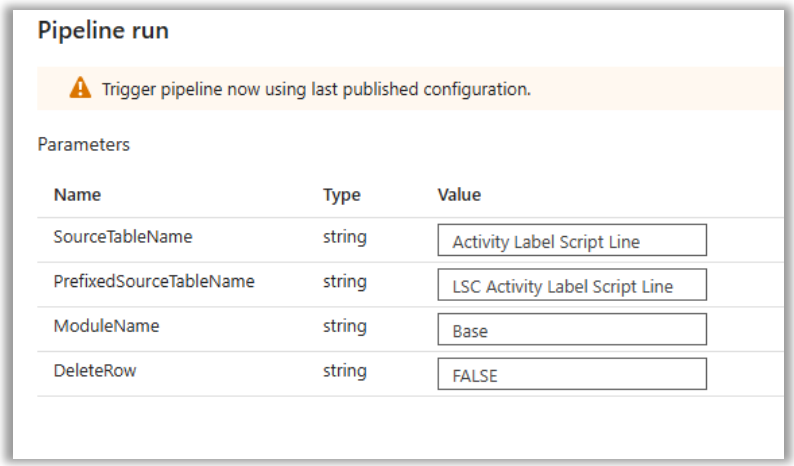
1. Add the required table(s) and field(s) in the bc2adls extension configuration in Business Central.
2. Run a schema export to regenerate the cdm.json files.
    a. This is required because Analytics generates SQL staging metadata based on the schema export.
3. Reset bc2adls tables and rerun export.
    a. We recommend running a reset of all tables in bc2adls and then running the export again. This reset is required because without it, bc2adls will only export deltas, and newly added columns/tables may not appear in the storage export until a full refresh is triggered. Reset forces the extension to read all data from Business Central and re-export the full dataset.

## 2. Adding new staging table to Source Tables Map

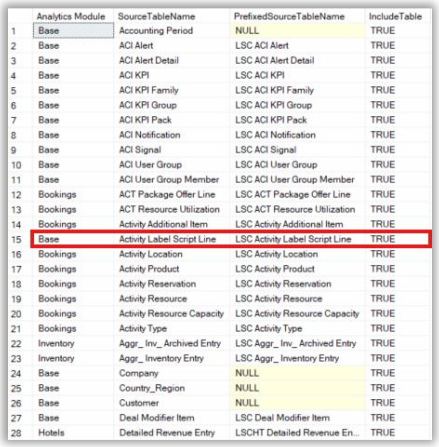


| Analytics Module | SourceTableName | PrefixedSourceTableName | IncludeTable |
|---|---|---|---|
| Base | Accounting Period | NULL | TRUE |
| Base | ACI Alert | LSC ACI Alert | TRUE |
| Base | ACI Alert Detail | LSC ACI Alert Detail | TRUE |
| Base | ACI KPI | LSC ACI KPI | TRUE |
| Base | ACI KPI Family | LSC ACI KPI Family | TRUE |
| Base | ACI KPI Group | LSC ACI KPI Group | TRUE |
| Base | ACI KPI Pack | LSC ACI KPI Pack | TRUE |
| Base | ACI Notification | LSC ACI Notification | TRUE |
| Base | ACI Signal | LSC ACI Signal | TRUE |
| Base | ACI User Group | LSC ACI User Group | TRUE |
| Base | ACI User Group Member | LSC ACI User Group Member | TRUE |
| Bookings | ACT Package Offer Line | LSC ACT Package Offer Line | TRUE |
| Bookings | ACT Resource Utilization | LSC ACT Resource Utilization | TRUE |
| Bookings | Activity Additional Item | LSC Activity Additional Item | TRUE |
| Bookings | Activity Location | LSC Activity Location | TRUE |
| Bookings | Activity Product | LSC Activity Product | TRUE |
| Bookings | Activity Reservation | LSC Activity Reservation | TRUE |
| Bookings | Activity Resource | LSC Activity Resource | TRUE |
| Bookings | Activity Resource Capacity | LSC Activity Resource Capacity | TRUE |
| Bookings | Activity Type | LSC Activity Type | TRUE |
| Inventory | Aggr_ Inv_ Archived Entry | LSC Aggr_ Inv_ Archived Entry | TRUE |
| Inventory | Aggr_ Inventory Entry | LSC Aggr_ Inventory Entry | TRUE |
| Base | Company | NULL | TRUE |
| Base | Country_Region | NULL | TRUE |
| Base | Customer | NULL | TRUE |

1. Once you have added a new table to the bc2adls export, you need to add it to the **Analytics$SourceTablesMap** table. If you don't add the table new tables to this table (or set the IncludeTable field to FALSE), then the staging process will not pickup the exported data from bc2adls.
2. You need to add the **table name** you want to use for the staging table in **SoureTableName** column and the actual name of the table in LS Central, without any GUID or Company name, in **PrefixedSourceTableName** column and set the Include field to **TRUE**. If the actual name of the table in LS Central is the same as the name you want use for the staging table then the value in **PrefixedSourceTableName** column can be NULL. You also need to assign the source table to a module. You can assign it to one of the Analytics modules if that's where the table fits logically, or create a custom module which you can reference at later stages in the ADF pipeline.
3. If the Include Table field is set to false, the table will not be included in Analytics staging.
4. You can add to the **Analytics$SourceTablesMap** table using the **Add or Delete Source Tables** pipeline in the Analytics ADF.
5. In the example below we will show how you would add a staging table to Analytics for the table
6. CRONUS - LS Central$LSC Activity Label Script Line$5ecfc871-5d82-43f1-9c54-59685e82318d
7. When you trigger the pipeline you are prompted to add a value to the **SourceTableName** and **PrefixedSourceTableName** fields, you can only add or delete one table at a time. If you add the value TRUE to the DeleteRow field, the source table name you specify will be deleted from the table if it exists. If you are adding a source table that is a BC standard table or you want the name of the new staging table to include the prefix, you can leave the **PrefixedSourceTableName** value empty.




When the pipeline has run and entry for the table has been added to **Analytics$SourceTablesMap** table (see image below).



| | Analytics Module | SourceTableName | PrefixedSourceTableName | IncludeTable |
|---|---|---|---|---|
| 1 | Base | Accounting Period | NULL | TRUE |
| 2 | Base | ACI Alert | LSC ACI Alert | TRUE |
| 3 | Base | ACI Alert Detail | LSC ACI Alert Detail | TRUE |
| 4 | Base | ACI KPI | LSC ACI KPI | TRUE |
| 5 | Base | ACI KPI Family | LSC ACI KPI Family | TRUE |
| 6 | Base | ACI KPI Group | LSC ACI KPI Group | TRUE |
| 7 | Base | ACI KPI Pack | LSC ACI KPI Pack | TRUE |
| 8 | Base | ACI Notification | LSC ACI Notification | TRUE |
| 9 | Base | ACI Signal | LSC ACI Signal | TRUE |
| 10 | Base | ACI User Group | LSC ACI User Group | TRUE |
| 11 | Base | ACI User Group Member | LSC ACI User Group Member | TRUE |
| 12 | Bookings | ACT Package Offer Line | LSC ACT Package Offer Line | TRUE |
| 13 | Bookings | ACT Resource Utilization | LSC ACT Resource Utilization | TRUE |
| 14 | Bookings | Activity Additional Item | LSC Activity Additional Item | TRUE |
| 15 | Base | Activity Label Script Line | LSC Activity Label Script Line | TRUE |
| 16 | Bookings | Activity Location | LSC Activity Location | TRUE |
| 17 | Bookings | Activity Product | LSC Activity Product | TRUE |
| 18 | Bookings | Activity Reservation | LSC Activity Reservation | TRUE |
| 19 | Bookings | Activity Resource | LSC Activity Resource | TRUE |
| 20 | Bookings | Activity Resource Capacity | LSC Activity Resource Capacity | TRUE |
| 21 | Bookings | Activity Type | LSC Activity Type | TRUE |
| 22 | Inventory | Aggr_ Inv_ Archived Entry | LSC Aggr_ Inv_ Archived Entry | TRUE |
| 23 | Inventory | Aggr_ Inventory Entry | LSC Aggr_ Inventory Entry | TRUE |
| 24 | Base | Company | NULL | TRUE |
| 25 | Base | Country_Region | NULL | TRUE |
| 26 | Base | Customer | NULL | TRUE |
| 27 | Base | Deal Modifier Item | LSC Deal Modifier Item | TRUE |
| 28 | Hotels | Detailed Revenue Entry | LSCHT Detailed Revenue En... | TRUE |

You can then move on and run the **Initial load or Factory reset** pipeline.

### 3. Run Initial load or Factory reset

Like explained above you should only run the Initial load pipeline if you made modifation before running it for the first time otherwise use the Factory reset.

The pipelines (Initial load and Factory reset) will both fetch the latest metadata from the cdm.json file and generate the data for the **AnalyticsBC2ADLSMetadata** table. If you run the Factory reset pipeline, then it will drop and recreate all staging and prestaging tables. Finally both pipelines will trigger the **Scheduled Run** pipeline that populates the new or updated staging table.

Once the pipeline has run the staging table should be populated with the new columns and fields. This does, however, not have any impact on the dimension or fact tables since those are populated using stored procedures that need to be created in the database according to the star schema design rules.

In the next sections we will explain in more detail how you can add a column to a dimension, add a new dimension and connect it to a fact table. And then how you can update the reports to include the new information.

# 6 Dimensions

The dimension tables are populated and updated by a stored procedure. The dimension stored procedures often combine more than one staging table into a single dimension.

All dimensions included in the DW have been created by the Analytics team and are included in the Analytics database.

Please note that when changing a dimension table in an Analytics environment that is already up and running, you need to be careful when making changes. The instructions below assume that you have not populated the data warehouse and can therefore drop and recreate tables as needed.

If you have already populated the data warehouse, you will either need to make the changes without dropping the dimension table or run the Factory reset pipeline after you have made all the changes. This is because dropping the table will create new SK values which link the dimension to the fact table, causing the data to be incorrect.

## Add new column to existing dimension

If you want to add new columns from new or existing staging tables that is a very straight forward process.

Let's imagine that you want to add information about whether an item is a scale item or not so you can see whether that is impacting sales in any way and so you can compare sales between stores for scale items only, since you have a feeling that some stores sell more scale items than others, but you want to confirm that suspicion.

The **stg$Item** table has a field called “Scale Item” that you can use to distinguish between scale items and non-scale items.

### Add column to dimension

The first thing you do is add a new column to the **dItem** dimension table.

The best way to edit a table in an existing Analytics database is to connect to the database using SQL Server Management Studio (SSMS).

We recommend connecting to the Azure database from SSMS (the connection information for the Analytics database was provided in the deployment summary) and then following these steps:

1) In the Analytics database expand Tables.
2) Select the **dItem** table
3) Right-click and select Script Table as > DROP and CREATE To > New query editor window.
    a. Like mentioned above you can also do ALTER TABLE here if you only want to update the values in the dimension, for example when adding a new column, and not create new SK values that would require you to recreate the fact tables.
4) The script will open in a new window.
5) In the CREATE TABLE part of the script, add the new column. Here you can use the same datatype as in the staging table if you are using the value as is, or you can change it if you want to transform the value in any way.

```
CREATE TABLE [DW].[dItem](
    [SK_Item] [int] IDENTITY(-1,1) NOT NULL,
    [Company] [int] NOT NULL,
    [No] [nvarchar](100) NOT NULL,
    [Description] [nvarchar](100) NULL,
    [Category] [nvarchar](100) NULL,
    [ProductGroup] [nvarchar](100) NULL,
    [InventoryPostingGroup] [nvarchar](100) NULL,
    [BaseUnitOfMeasure] [nvarchar](100) NULL,
    [Item Capacity Value] [decimal](38, 20) NULL,
    [UnitPrice] [decimal](38, 20) NULL,
    [LastItemCost] [decimal](38, 20) NULL,
    [VendorNo] [nvarchar](100) NULL,
    [VendorItemNo] [nvarchar](100) NULL,
    [Division] [nvarchar](100) NULL,
    [Scale Item] [tinyint] NULL,
    [RowID] [bigint] NULL,
    [Batchdate] [datetime] NULL,
```

6) Then add an ALTER TABLE section to set the default value of the new column. In this example we have set the default value to zero. But the default selected depends on the datatype.
7) Now run the script
8) The message “Commands completed successfully” is displayed, and the table will be dropped and recreated including the new column. The table will be empty and to populate it again you need to edit the dimMergedItem stored procedure.

To modify the stored procedure that loads data into the dItem table, do the following in SSMS:

1) In the Analytics database expand Programmability.
2) Select the dbo.dimMergedItem (or vX.XdimMergedItem) stored procedure for the LS Central version you are using.
3) Right-click and select Modify.
4) A modification script for the procedure is opened.
5) Since you are already selecting from the **stg$Item** table in the procedure, you just need to add the column where needed, and since this is tiny int value there is no need to check for NULL values. So, what you do is select from the Scale item column into the temp table and add it to the GROUP BY aggregation as well.

```
tItem
AS
(SELECT
        sITM.[CompanyPrefix]
       ,sITM.[No_] AS [No]
       ,sITM.[Description]
       ,COALESCE(tITC.[Description], '') AS [Category]
       ,COALESCE(tRPG.[Description], '') AS [ProductGroup]
       ,sITM.[Inventory Posting Group]
       ,sITM.[Base Unit of Measure]
       ,sITM.[Item Capacity Value]
       ,COALESCE(tSAP.[Unit Price], 0) AS [Unit Price]
       ,CASE sITM.[Costing Method]
            WHEN 4 THEN COALESCE(IIF(sITM.[Standard Cost] = 0, NULL, sITM.[Standard
            ELSE COALESCE(IIF(sITM.[Last Direct Cost] = 0, NULL, sITM.[Last Direct
        END AS [LastItemCost]
       ,sITM.[Vendor No_]
       ,sITM.[Vendor Item No_]
       ,COALESCE(tDIV.[Description], '') AS [DivisionName]
       ,sITM.[Scale Item]
       ,MAX(sITM.[bigint_timestamp]) AS [RowID]
       ,GETUTCDATE() AS [BatchDate]
    FROM [Stg$Item] sITM
    LEFT JOIN [tItemCategory] tITC
        ON sITM.[CompanyPrefix] = tITC.[CompanyPrefix]
        AND sITM.[Item Category Code] = tITC.[Code]
    LEFT JOIN [tProductGroup] tRPG
        ON sITM.[CompanyPrefix] = tRPG.[CompanyPrefix]
        AND sITM.[Retail Product Code] = tRPG.[Code]
    LEFT JOIN [tDivision] tDIV
        ON sITM.[CompanyPrefix] = tDIV.[CompanyPrefix]
        AND sITM.[Division Code] = tDIV.[Code]
    LEFT JOIN [tSalesPrice] tSAP
        ON sITM.[CompanyPrefix] = tSAP.[CompanyPrefix]
        AND sITM.No_ = tSAP.[Item No_]
    GROUP BY sITM.CompanyPrefix
            ,sITM.[No_]
            ,sITM.[Description]
            ,tITC.[Description]
            ,tRPG.[Description]
            ,sITM.[Inventory Posting Group]
            ,sITM.[Base Unit of Measure]
            ,sITM.[Item Capacity Value]
            ,sITM.[Unit Price]
            ,COALESCE(IIF([Standard Cost] = 0, NULL, [Standard Cost]), IIF([Last Di
            ,COALESCE(IIF([Last Direct Cost] = 0, NULL, [Last Direct Cost]), IIF([U
            ,sITM.[Vendor No_]
            ,sITM.[Vendor Item No_]
            ,tDIV.[Description]
            ,tSAP.[Unit Price]
            ,sITM.[Costing Method]
            ,sITM.[Scale Item])
```

If the column you wanted to add to the dimension was from a new staging table, you would need to create a temp table for that staging table as you did for ItemCategory and ProductGroup and do the join here. As you can see from the other join statements, we do a LEFT JOIN on the ID and the CompanyPrefix.

6) Then select the column from tItem in the MERGE section.

```
MERGE [DW].[dItem] AS Target USING (SELECT
        COALESCE(dCOM.[SK_Company], -1) AS [Company]
       ,tITM.[No]
       ,tITM.[Description]
       ,tITM.[Category]
       ,tITM.[ProductGroup]
       ,tITM.[Inventory Posting Group]
       ,tITM.[Base Unit of Measure]
       ,tITM.[Item Capacity Value]
       ,tITM.[Unit Price]
       ,tITM.[LastItemCost]
       ,tITM.[Vendor No_]
       ,tITM.[Vendor Item No_]
       ,tITM.[DivisionName]
       ,tITM.[Scale Item]
       ,tMRI.[RowID]
       ,tITM.[BatchDate]
    FROM [tItem] tITM
    RIGHT JOIN [tMAXRowIDItem] tMRI
        ON tITM.[CompanyPrefix] = tMRI.[CompanyPrefix]
        AND tITM.[No] = tMRI.[No_]
        AND tITM.[RowID] = tMRI.[RowID]
    LEFT JOIN [DW].[dCompany] dCOM
        ON tITM.[CompanyPrefix] = dCOM.[CompanyPrefix]) AS Source
```

7) Add the column to the UPDATE statement.

```
WHEN MATCHED
    THEN UPDATE
        SET [Description] = Source.[Description]
           ,[Category] = Source.[Category]
           ,[ProductGroup] = Source.[ProductGroup]
           ,[InventoryPostingGroup] = Source.[Inventory Posting Group]
           ,[BaseUnitOfMeasure] = Source.[Base Unit of Measure]
           ,[Item Capacity Value] = Source.[Item Capacity Value]
           ,[UnitPrice] = Source.[Unit Price]
           ,[LastItemCost] = Source.[LastItemCost]
           ,[VendorNo] = Source.[Vendor No_]
           ,[VendorItemNo] = Source.[Vendor Item No_]
           ,[Division] = Source.[DivisionName]
           ,[Scale Item] = Source.[Scale Item]
           ,[RowID] = Source.[RowID]
           ,[Batchdate] = Source.[Batchdate]
```

8) And lastly, add the column to the INSERT statement.

```
WHEN NOT MATCHED BY TARGET
       THEN INSERT ([Company], [No], [Description], [Category], [ProductGroup],
       [InventoryPostingGroup], [BaseUnitOfMeasure], [Item Capacity Value],
       [UnitPrice], [LastItemCost], [VendorNo], [VendorItemNo], [Division], [Scale
       Item], [RowID], [BatchDate])
       VALUES (Source.[Company], source.[No], Source.[Description],
       Source.[Category], Source.[ProductGroup], Source.[Inventory Posting Group],
       Source.[Base Unit of Measure], Source.[Item Capacity Value], Source.[Unit
       Price], Source.[LastItemCost], Source.[Vendor No_], Source.[Vendor Item
       No_], Source.[DivisionName], Source.[Scale Item], Source.[RowID],
       Source.[BatchDate])
```

9) Now run the ALTER procedure script.
10) A “Commands completed successfully” message is displayed.

Now that the procedure has been modified, you can either execute it from SSMS or you can trigger the Scheduled Run pipeline from Azure Data Factory.

When the procedure has been executed, the data in the new column has been populated with the correct data from the staging table.

## Add new dimension

If you would like to add a new dimension to the Analytics data warehouse from LS Central, the process is straight forward.

**Here are the steps you need to follow:**

1. The first thing you do is add the table name to the **Analytics$SourceTablesMap** table. Using the **Add or Delete Source Tables** pipeline in the Azure data factory. Remember to assign the new table to a module.

2. If you are running LS Central SaaS you then need to follow the process of adding one or more prestaging tables in the chapters above.
3. Re-run the **Populate Query Base** pipeline in the Azure data factory.
4. Run the **Scheduled Run** pipeline to retrieve the data from the new tables.
5. Create a new dimension table with the columns you want to include.
6. Create a new stored procedure to populate the dimension table with data from one or more staging tables. For the stored procedure to be run automatically you must keep to the naming convention and prefix SP name with 'dim' and the correct module name.
7. Add connections from this new dimension table to the appropriate fact tables.

There are several dimension tables in the DW schema that you can use as examples for this, and you can view the stored procedure used to populate them from the staging tables under Programmability. All the dimension stored procedures are prefixed with 'dim' and then the module they belong to (except base). So if you want to add a dimension to the Hospitality module, your procedure must be named dimHospitalityMerged%YOUR_DIM_NAME% so that the data factory will execute the procedure with the correct module. If you have no module name (dimMerged%YOUR_DIM_NAME%), then the dimension procedure will be executed as part of the Base module.

# 7 Facts

## Add new column from staging table to existing fact table

To add a new column from staging table to fact table you would go through a similar process as was used for new column to a dimension table. But since the stored procedures that update fact and dimension tables are different, we will go through an example.

In the following example we will show how to add the points column from the stg$xxx table to the fDiscount table and how to modify the stored procedure that loads the fDiscount table, so it includes inserts and updates to the newly added column.

### Add column to fact table

The first thing you do is add a new column to the **fDiscount** fact table.

The best way to edit a table in an existing database is to connect to the database using SQL Server Management Studio (SSMS).

We recommend connecting to the Azure database from SSMS (the connection information for the Analytics database was provided in the deployment summary) and then following these steps:

1) In the Analytics database, open a new query.
2) Enter the following ALTER script for the fDiscount table:

```
2
3  ALTER TABLE [DW].[fDiscounts]
4    ADD [Points] [decimal](38, 20) NULL
5    GO
6
```

3) If you want, you can add a default value constraint of the new column. In this example we have not set a default value.
4) Now run the script.
5) A Commands completed successfully message is displayed, the table will be altered, and the new column added but containing only NULL values. To populate the new column with values, you need to modify the stored procedure that loads the fDiscount table.

### Modify stored procedure

To modify the stored procedure that loads data into the dItem table, do the following in SSMS:

1) In the Analytics database, expand Programmability.
2) Select the dbo.factDiscount.
3) Right-click and select Modify.
4) A modification script for the procedure opens.
   Since you are already selecting from the stg$ table in the procedure, you just need to add the column reference where needed, and since this is decimal value there is no need to check for NULL values. So, what you do in the tDiscount temp table creation selection, is select from the [Points] column in the **stg$Trans_ Discount Entry** staging table.

```
/*Temp table for Discount entries*/
tDiscounts
AS
(SELECT
        sTDE.[CompanyPrefix]
       ,CAST(sTRH.[Date] AS DATE) AS [Date]
       ,CAST(DATEADD(mi, DATEDIFF(mi, 0, sTRH.[Time]), 0) AS TIME) AS [Time]
       ,tLOC.[SK_Location]
       ,tOFF.[SK_Offer]
       ,tPOT.[SK_POSTerminal]
       ,CASE sTDE.[Offer Type]
            WHEN 0 THEN 'Promotion'
            WHEN 1 THEN 'Deal'
            WHEN 2 THEN 'Multibuy'
            WHEN 3 THEN 'Mix&Match'
            WHEN 4 THEN 'Disc. Offer'
            WHEN 5 THEN 'Total Discount'
            WHEN 6 THEN 'Tender Type'
            WHEN 7 THEN 'Item Point'
            WHEN 8 THEN 'Line Discount'
            WHEN 9 THEN 'Customer'
            WHEN 10 THEN 'Infocode'
            WHEN 11 THEN 'Member Point'
            WHEN 12 THEN 'Coupon'
            WHEN 13 THEN 'Total'
            WHEN 14 THEN 'Line'
            ELSE 'No Offer type'
        END AS [Offer Type]
       ,sTDE.[Receipt No_]
       ,sTDE.[Line No_]
       ,CAST(sTDE.[Transaction No_] AS NVARCHAR(100)) AS [TransactionNo]
       ,sTDE.[Discount Amount]
       ,sTDE.[Points]
       ,sTDE.bigint_timestamp AS [RowID]
    FROM [stg$Trans_ Discount Entry] sTDE
    LEFT OUTER JOIN [stg$Transaction Header] sTRH
        ON sTDE.[Store No_] = sTRH.[Store No_]
        AND sTDE.[POS Terminal No_] = sTRH.[POS Terminal No_]
        AND sTDE.[Transaction No_] = sTRH.[Transaction No_]
        AND sTDE.[CompanyPrefix] = sTRH.[CompanyPrefix]
```

If the column you wanted to add to the fact table was from a new staging table, you would need to add this new staging table to the left outer join below.

5) Select the column from tDiscoutns in the MERGE section:

```
MERGE [DW].[fDiscounts] AS TARGET USING (SELECT
        COALESCE(dCOM.[SK_Company], -1) AS [Company]
       ,tDIS.[Date]
       ,tDIS.[Time]
       ,COALESCE(tDIS.[SK_Location], -1) AS [SK_Location]
       ,COALESCE(tDIS.[SK_Offer], -1) AS [SK_Offer]
       ,COALESCE(tDIS.[SK_POSTerminal], -1) AS [SK_POSTerminal]
       ,COALESCE(tSAL.[SK_Item], -1) AS [SK_Item]
       ,COALESCE(tSAL.[SK_Member], -1) AS [SK_Member]
       ,COALESCE(tSAL.[SK_Staff], -1) AS [SK_Staff]
       ,tDIS.[Offer Type]
       ,tDIS.[Receipt No_]
       ,tDIS.[Line No_]
       ,tDIS.[TransactionNo]
       ,tDIS.[Discount Amount]
       ,tDIS.[Points]
       ,tSAL.NetSalesAmountLCY AS SalesAmount
       ,tSAL.CostAmountLCY AS CostAmount
       ,tDIS.[RowID]
```

6) Add the column to the UPDATE statement:

```
WHEN MATCHED
    THEN UPDATE
        SET [Time] = Source.[Time]
           ,[Receipt No_] = Source.[Receipt No_]
           ,[Discount Amount] = Source.[Discount Amount]
           ,[Points] = Source.[Points]
           ,[RowID] = Source.[RowID]
           ,[Batchdate] = GETUTCDATE()
           ,[SK_Item] = Source.[SK_Item]
           ,[SK_Member] = Source.[SK_Member]
           ,[SK_Staff] = Source.[SK_Staff]
           ,[SalesAmount] = Source.[SalesAmount]
           ,[CostAmount] = Source.[CostAmount]
```

7) And lastly, add the column to the INSERT statement.

```
WHEN NOT MATCHED BY TARGET
       THEN INSERT ([Company]
              , [Date]
              , [Time]
              , [SK_Location]
              , [SK_Offer]
              , [SK_POSTerminal]
              , [Offer Type]
              , [Receipt No_]
              , [Line No_]
              , [TransactionNo]
              , [Discount Amount]
              , [Points]
              , [RowID]
              , [Batchdate]
              , [SK_Item]
              , [SK_Member]
              , [SK_Staff]
              , [SalesAmount]
              , [CostAmount])
                    VALUES (Source.[Company], Source.[Date], Source.[Time],
                    Source.[SK_Location], Source.[SK_Offer],
                    Source.[SK_POSTerminal], Source.[Offer Type], Source.[Receipt
                    No_], Source.[Line No_], Source.[TransactionNo],
                    Source.[Discount Amount],Source.[Points], Source.[RowID],
                    GETUTCDATE(), Source.[SK_Item], Source.[SK_Member],
                    Source.[SK_Staff], Source.[SalesAmount], Source.[CostAmount]);
```

1) Run the ALTER procedure script.
2) A Comands completed successfully message is displayed.

Now that the procedure has been modified, you need to execute it from SSMS with these parameter values:

@CurrentRowID = 0

@NewRowID = 999999999999999

This will ensure that all the rows of the factDiscount table will be updated with the points value from the staging table.

## Add new fact table

The steps needed to add a new fact table to the Analytics data warehouse are described in chapter **7.4 Adding fact table to star schema** in the section about third-party data. The process of creating a new fact table from LS Central data is exactly the same, but in that case the staging tables hold data from LS Central instead of third-party data. To utilize the Analytics dynamic staging process for these new fact tables you need to add the source tables to Analytics$SourceTableMap.

## Add new pipelines to ADF

With the modular setup, we recommend that you follow the same structure as Analytics. Each module is comprised of a few objects.

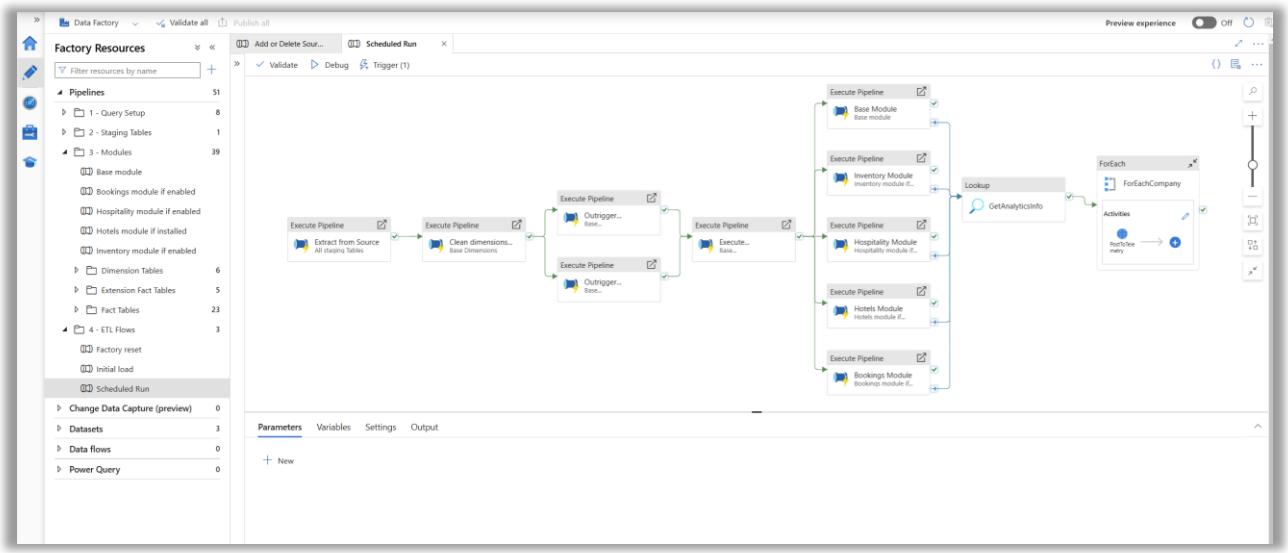
- Source tables (defined in the Analytics$SourceTablesMap table)
- Dimension tables and procedures.
    - Not all modules have their own dimensions (for example, Inventory)
    - All dimension procedures should follow the naming convention of dim'ModuleName'Merged'ProcName'. dimHospitalityMergedDiningTable is for example the name of the procedure that loads the **DiningTable** dimension in the **Hospitality** module.
- Fact tables and procedures
- Seperate pipeline in ADF
    - Each module pipeline starts by checking if module is enabled and then proceeds to run pipelines if true.
    - Each module pipeline executes dimension procedures first and then the fact table procedures once dimensions are all completed.

We recommend that you create all pipelines in bicep or arm templates so that you can easily apply your customizations to new environments when updating or setting up new installations.

### Modules in scheduled run

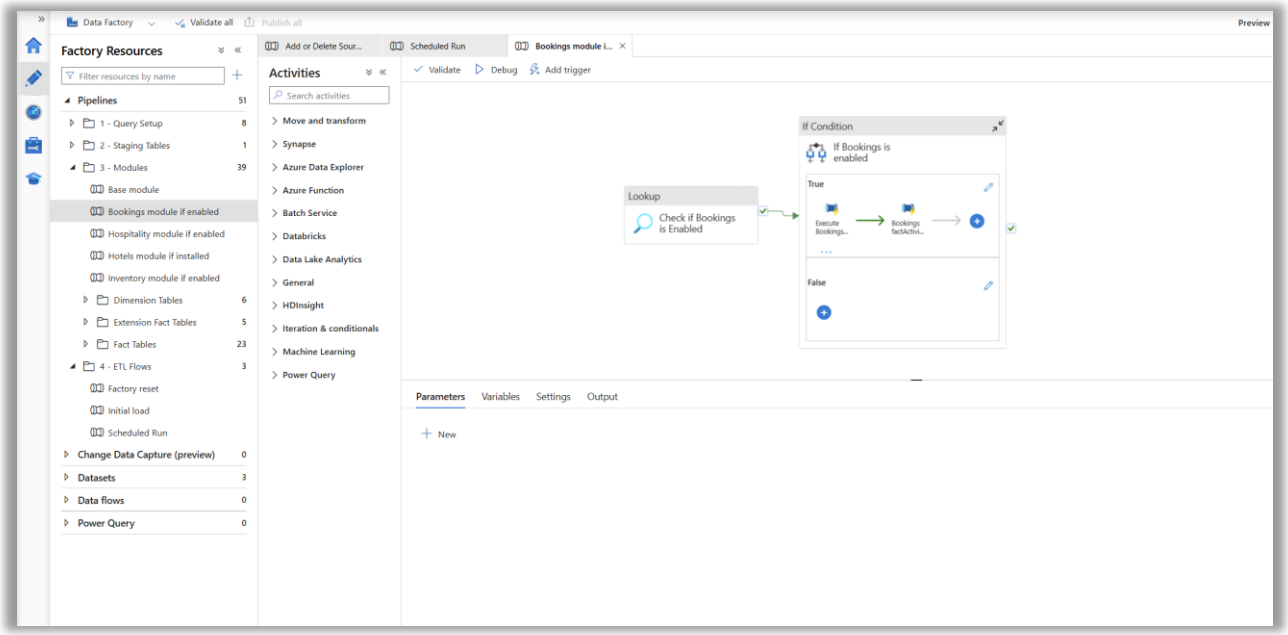
The module pipelines are then all part of the **Scheduled run** pipeline.

On the image below you can see how the Scheduled run pipeline contains execute pipeline activities for all modules.

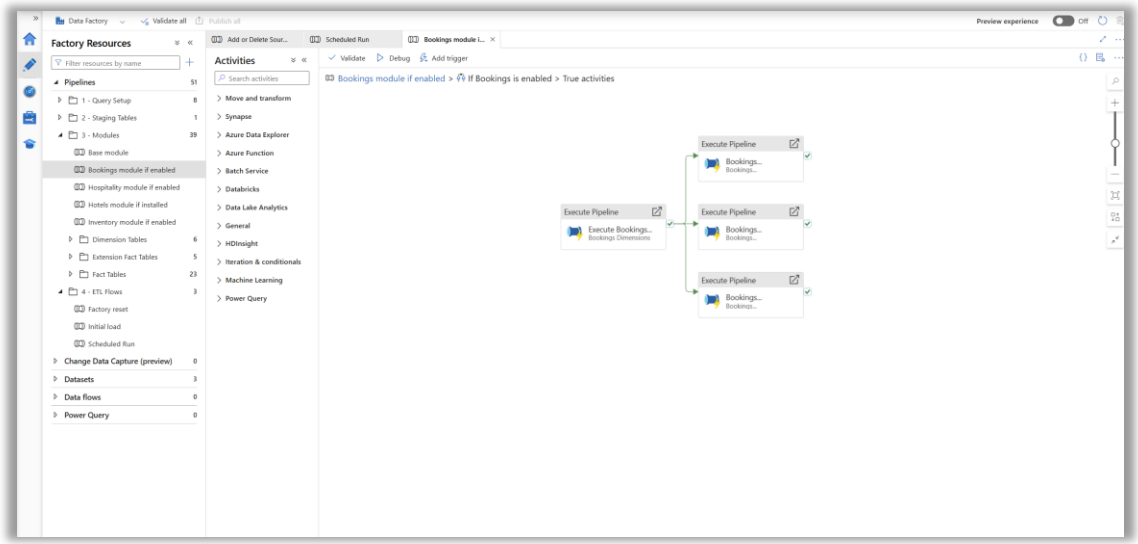


## Structure of modules

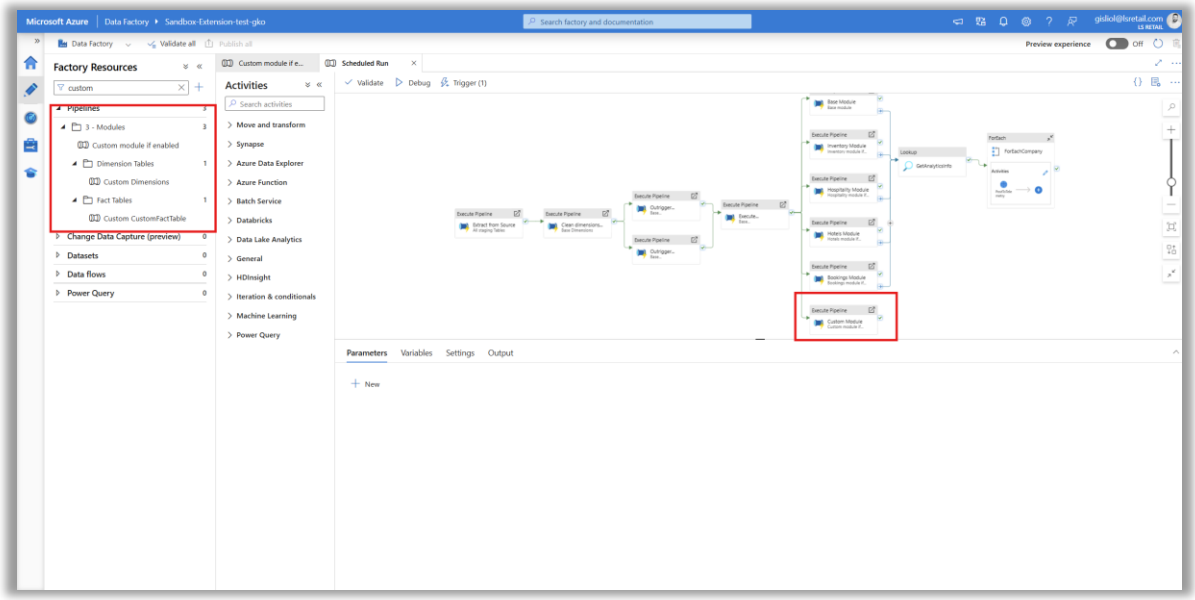
For each module we have defined a wrapper pipeline which contains all pipelines for that particular module. Each module wrapper pipeline contains a check to see if the module is active, and if it is, the pipelines are triggered.




The module wrapper pipeline can contain multiple dimension- and fact-pipelines as seen here.




Here you can see what a custom module might look like in the data factory. You can download the bicep files with the custom module (which is just for demonstration purposes) to get a better understanding of how to create and maintain your custom modules.




Here you can download the bicep files with the added custom module shown in the example above.

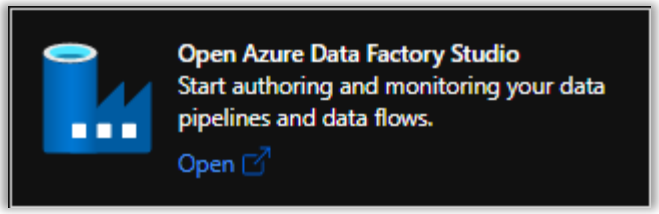
https://analyticsdelivery.blob.core.windows.net/analyticsdownloadsite/arm_template_module_demo.zip

# 8 Third-party data

Here are the recommended steps you need to take to add third-party data to Analytics. In this example you will be adding customer counter data from file. We will first describe how to add the data to the Analytics data warehouse, and then how the data could be added directly to the Power BI report, if you want to bypass the data warehouse.

## Open the Azure Data Factory studio

Log into your Azure environment (https://portal.azure.com/), and open the Analytics Azure Data Factory. You will find all resources in "All resources".

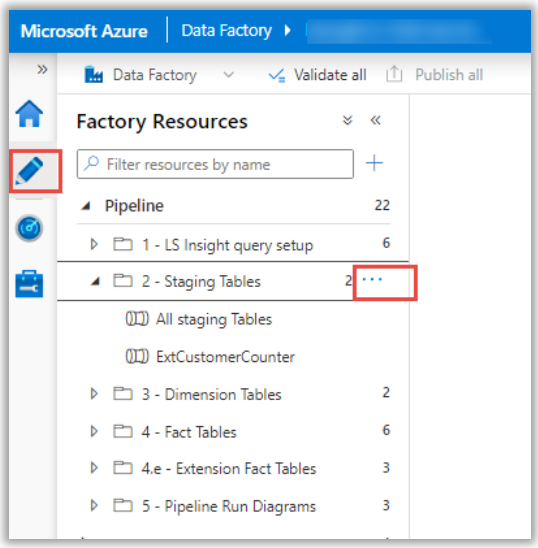



## Staging data

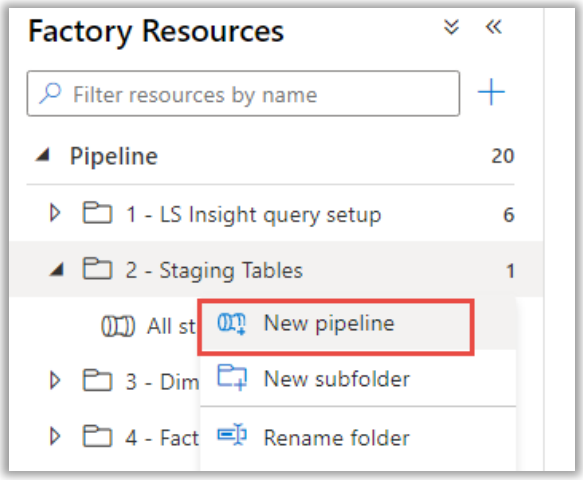
The first step is to get the source data in a staging table in the data warehouse (DW). In some scenarios this is not needed, but it is good practice to stage the source data before cleaning and writing to the star schema tables. There are many methods available to ingest the data and create the staging table – in this example we are using Azure data factory (ADF).

### Create a copy pipeline

Open the editor option, and click the three dots next to "2 – Staging Tables":

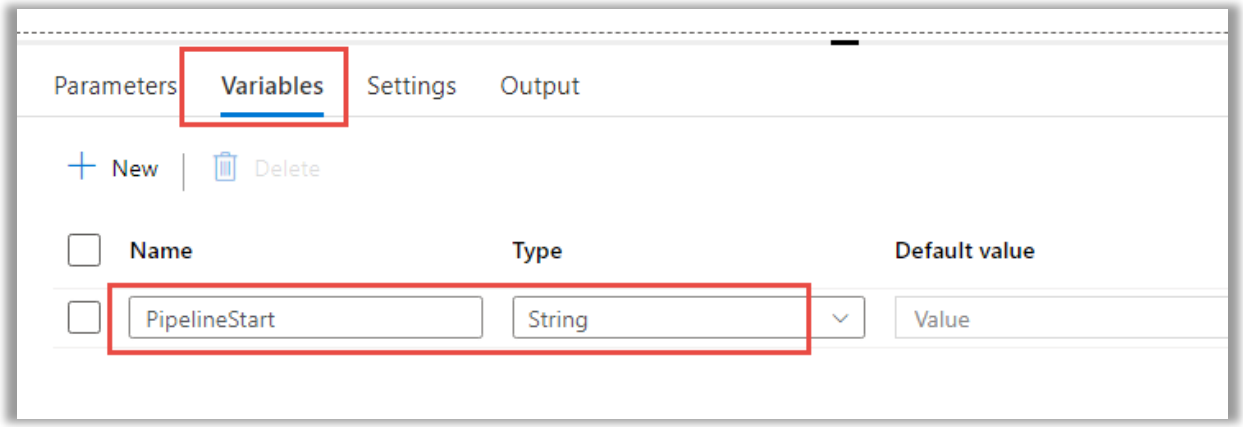



Select to create a new pipeline in the selected folder by clicking **New pipeline**. And give it a descriptive name. It is a good idea to add an affix to the name to distinguish it from Analytics pipelines.

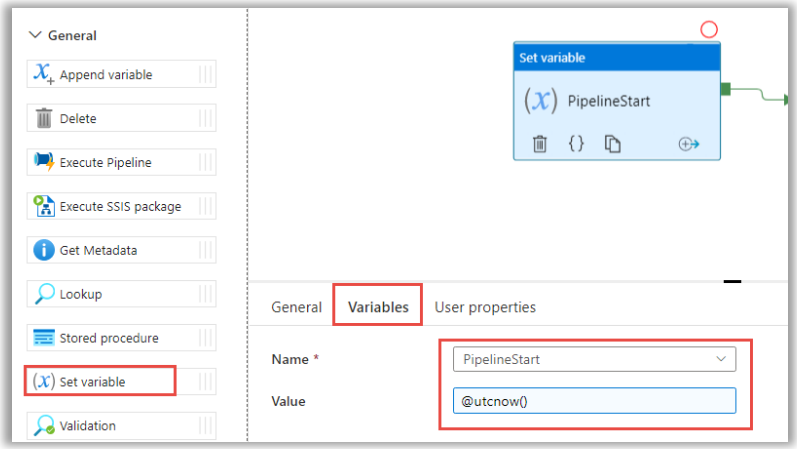



### Add Pipeline variables

Add a pipeline variable name “PipelineStart” to hold value of utcnow() to be able to log the pipeline start in the AnalyticsAudit table in the Analytics database.

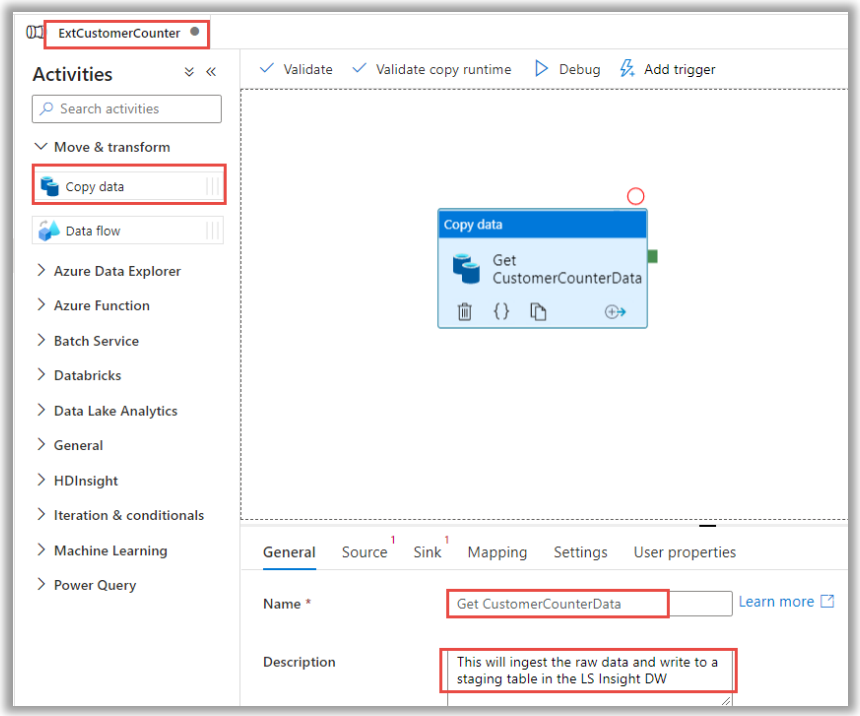



Set the current timestamp using the “Set variable” activity using “Add dynamic content”



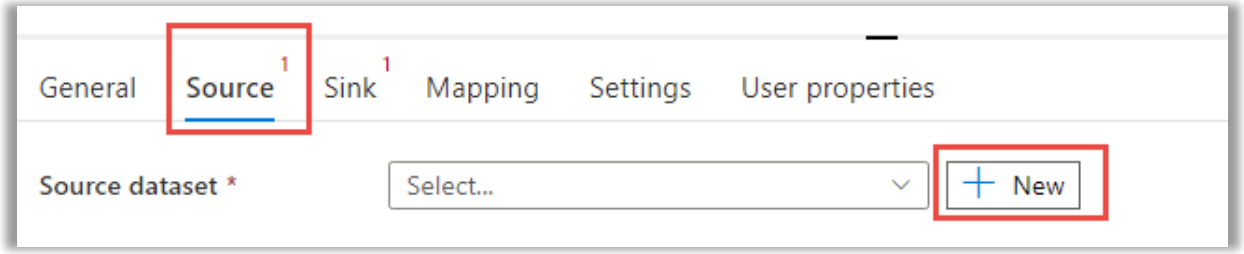
## Copy Activity

Add a copy activity to copy the data from the source and write to a staging table in your DW.

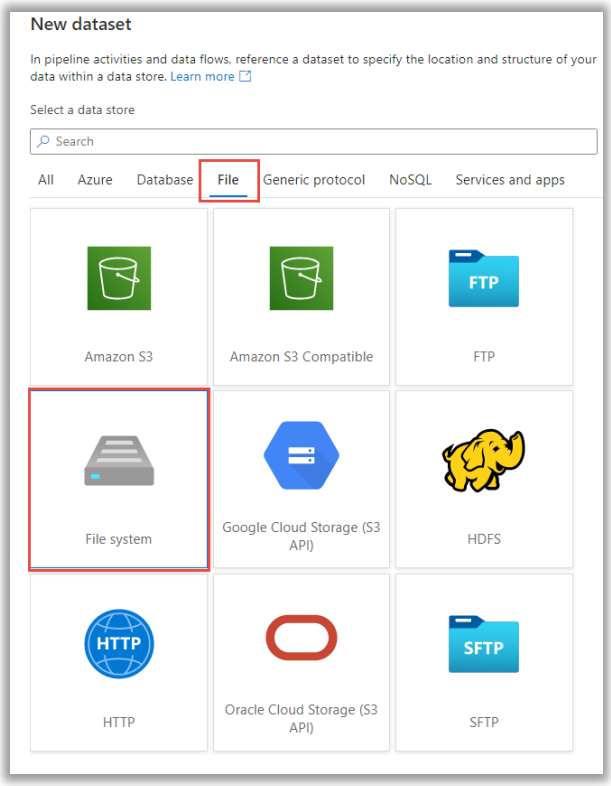



## Create a new source connection

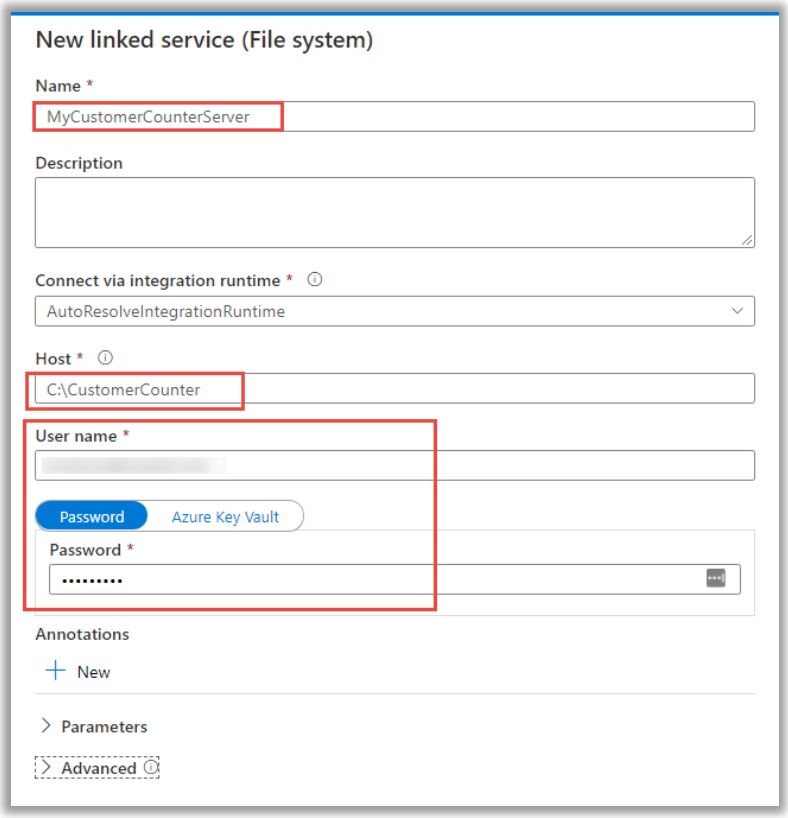
The source connection needs to be set. If this connection does not exist in your ADF, a new connection is created in the copy activity.




For this example, you will be using a flat file connection – you can choose from over 200 connectors, depending on your data.

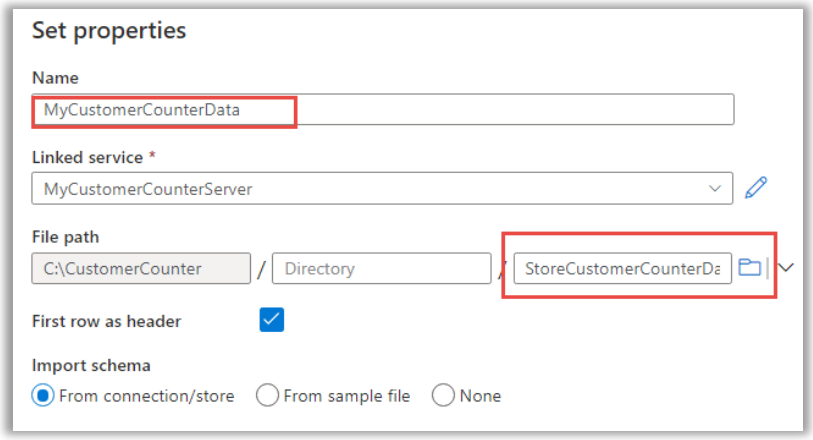



Create a connection to the host.



### Connect to your data

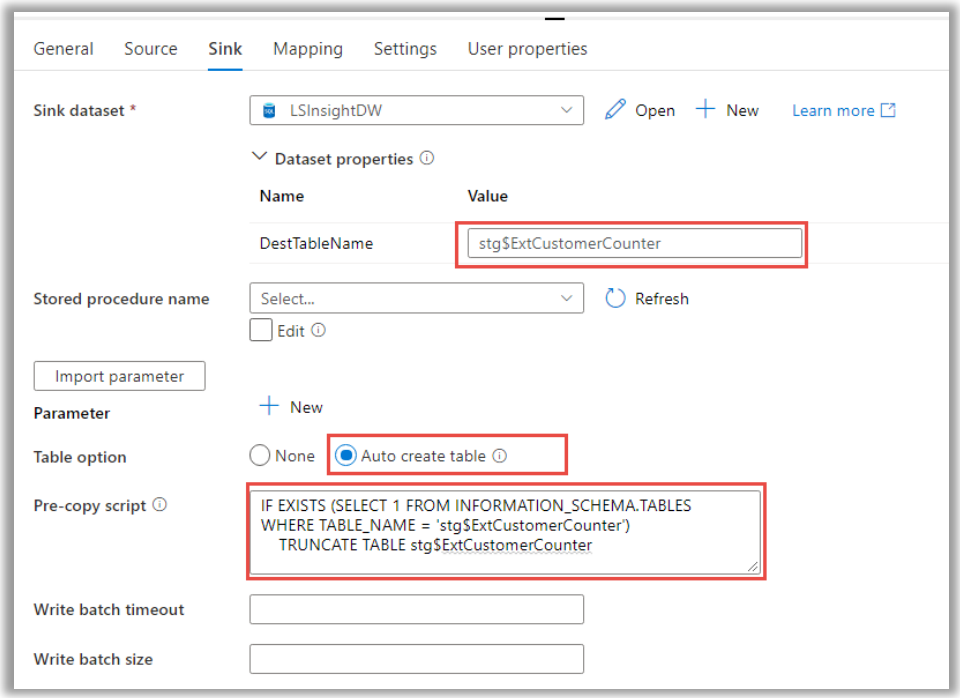
Using the connection, select the data source. In this example the source is a text file in your local file system.



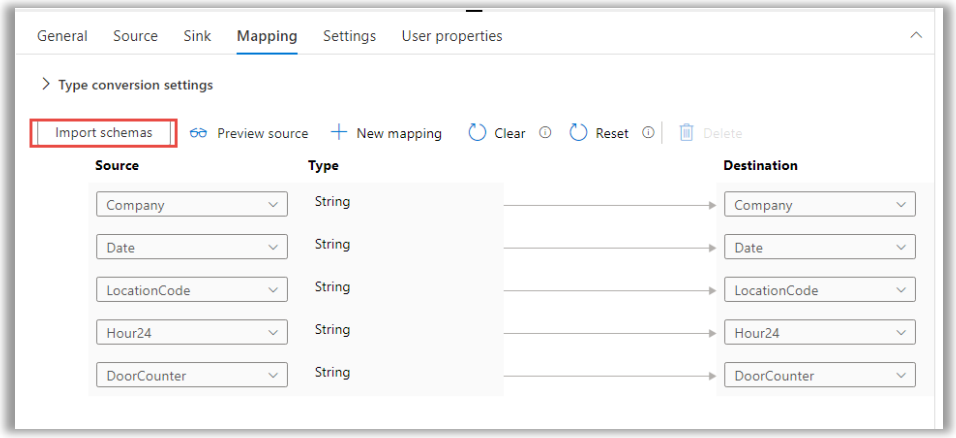

### Set the pre-copy script

In this example the full dataset is written to the staging table. The pre-copy script truncates the staging table before writing the data from the source again on a schedule.

If there is not a separate method to ensure the staging table exists, there is an option to auto create the table based on the source data.

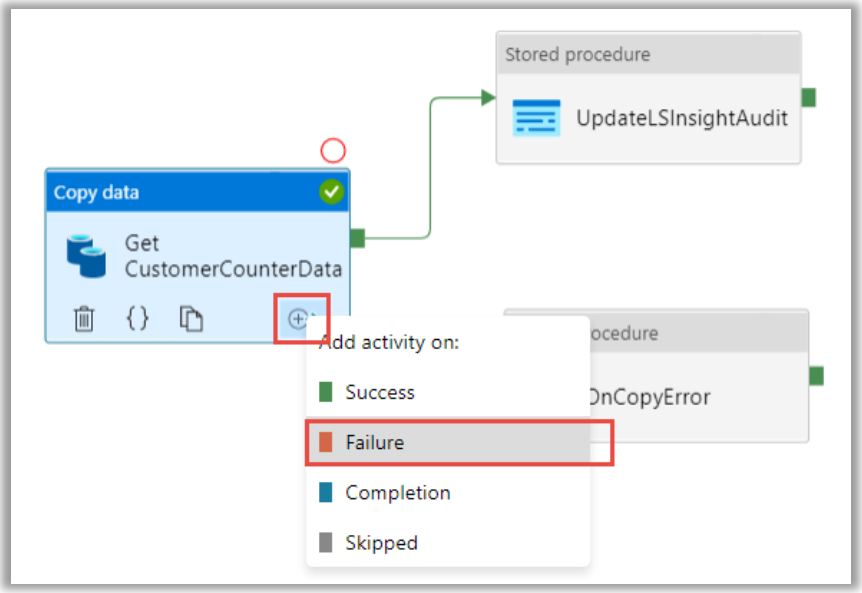



Import the schemas and verify the mapping.

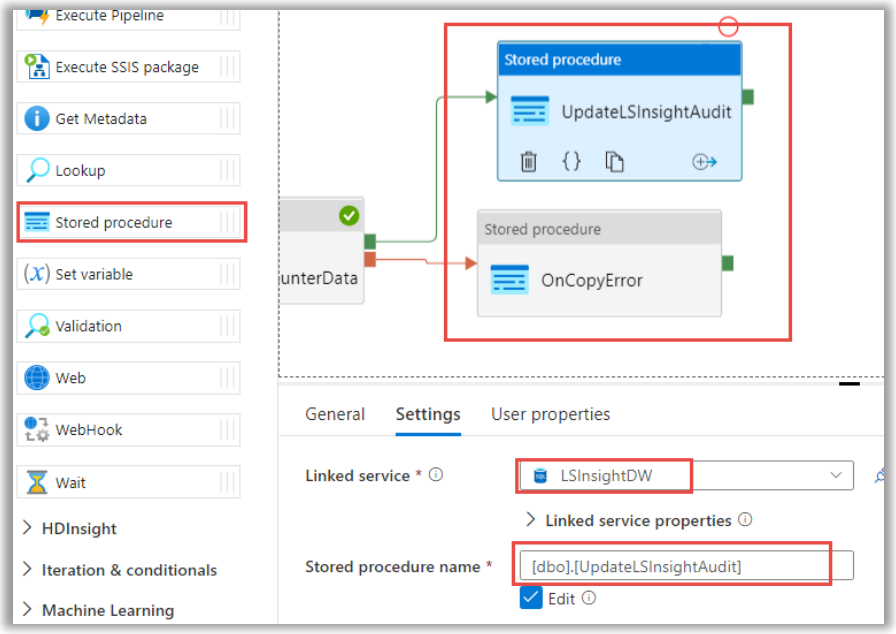



### Add Write to audit table action

Add a failure activity to the copy action to be able to trigger an activity when the copy activity has a failure.

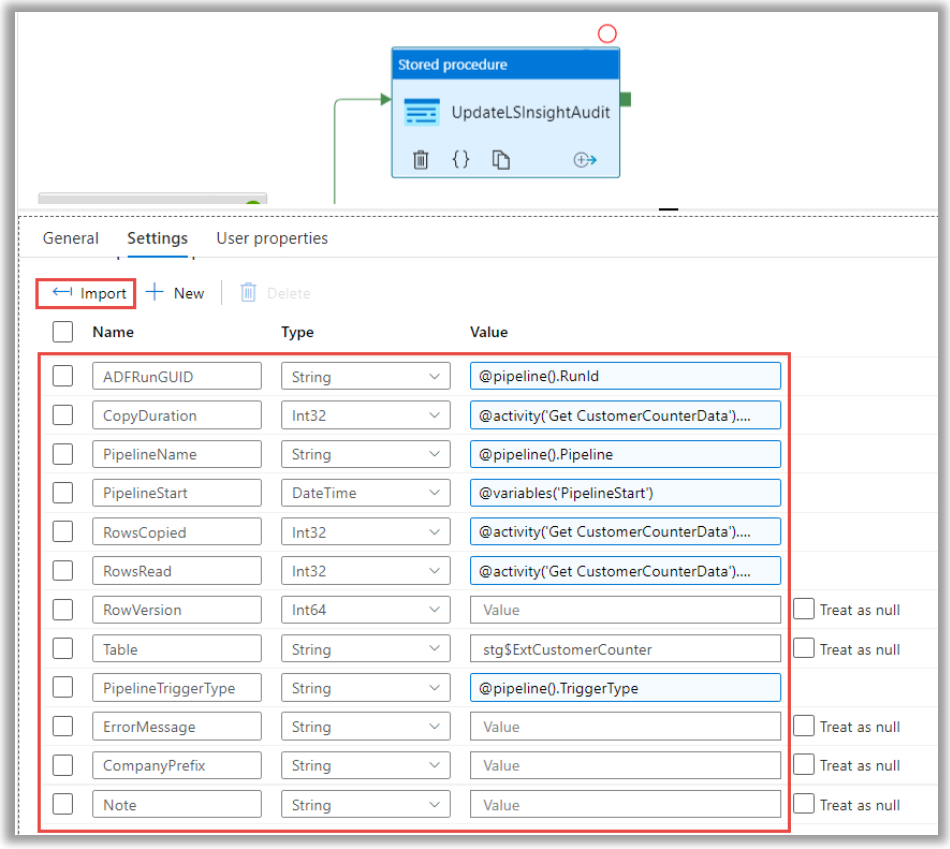



Add two "Stored procedure" activities that will both execute [dbo].[UpdateAnalyticsAudit] and connect Success to *UpdateAnalyticsAudit* and failure to *OnCopyError*




## Update AnalyticsAudit on success

Set the SP variables as follows for the success activity – *UpdateAnalyticsAudit:*



ADFRunGUID = @pipeline().RunId

CopyDuration = @activity('Get CustomerCounterData').output.copyDuration

PipelineName = @pipeline().Pipeline

PipelineStart = @variables('PipelineStart')

RowsCopied = @activity('Get CustomerCounterData').output.rowsCopied

RowsRead = @activity('Get CustomerCounterData').output.rowsRead

RowVersion is only used when you need to set up Incremental loading

Table = stg$ExtCustomerCounter (the name of the staging table you are writing to)

PipelineTriggerType = @pipeline().TriggerType

ErrorMessage is not used here

CompanyPrefix used if you have multiple company setup in your data

Note – free space to add additional information

### Update AnalyticsAudit on failure

On the failure activity, *OnCopyError,* the following settings apply for the same stored procedure:

ADFRunGUID = @pipeline().RunId

CopyDuration = 0

PipelineName = @concat('Failed Run - ', pipeline().Pipeline)

PipelineStart = @variables('PipelineStart')

RowsCopied = 0

RowsRead = 0

RowVersion = 0

Table = stg$ExtCustomerCounter (the name of the staging table you are writing to)

PipelineTriggerType = @pipeline().TriggerType

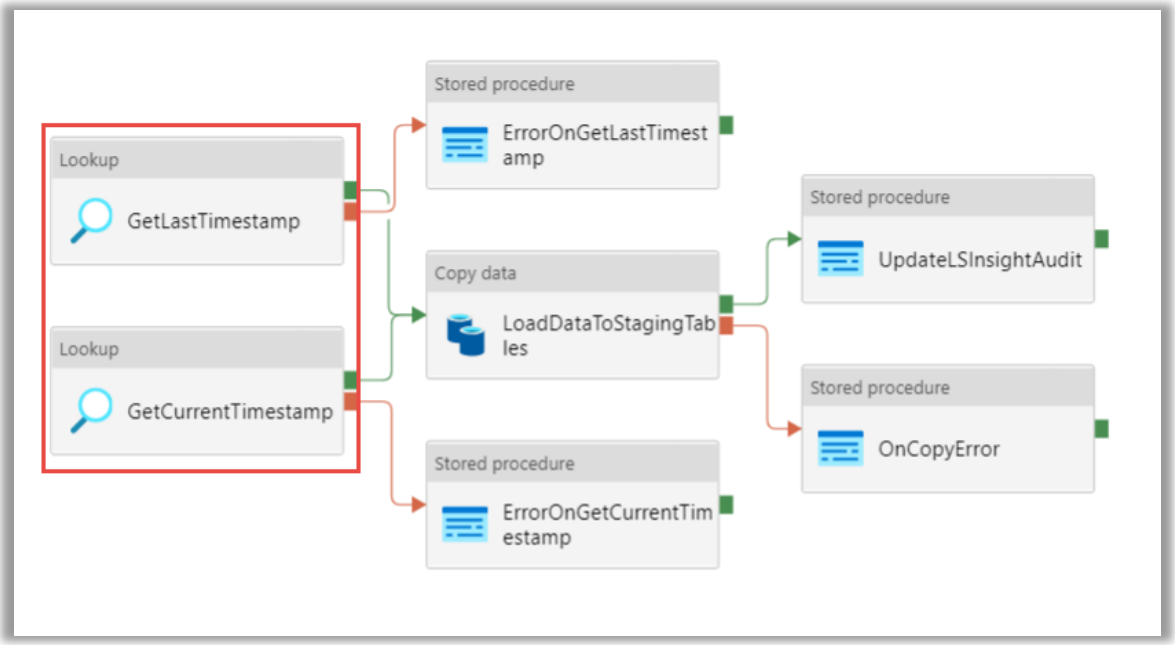
ErrorMessage = @activity('Get CustomerCounterData').Error.Message

CompanyPrefix used if you have multiple company setup in your data

### Large data source

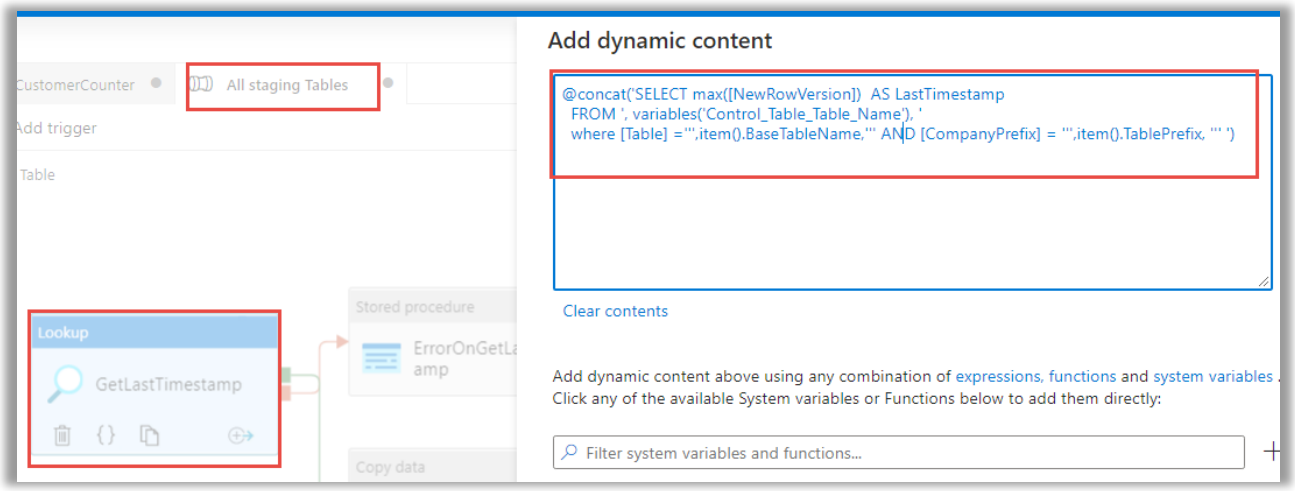
For large datasets where daily full load is not an option you will need incremental load. For this you will need to get the Current timestamp (or the column used in your case to determine the incremental load) and the last timestamp from the audit table.

Here is an example from the All staging Tables pipeline:




The last timestamp is extracted from the table AnalyticsAudit

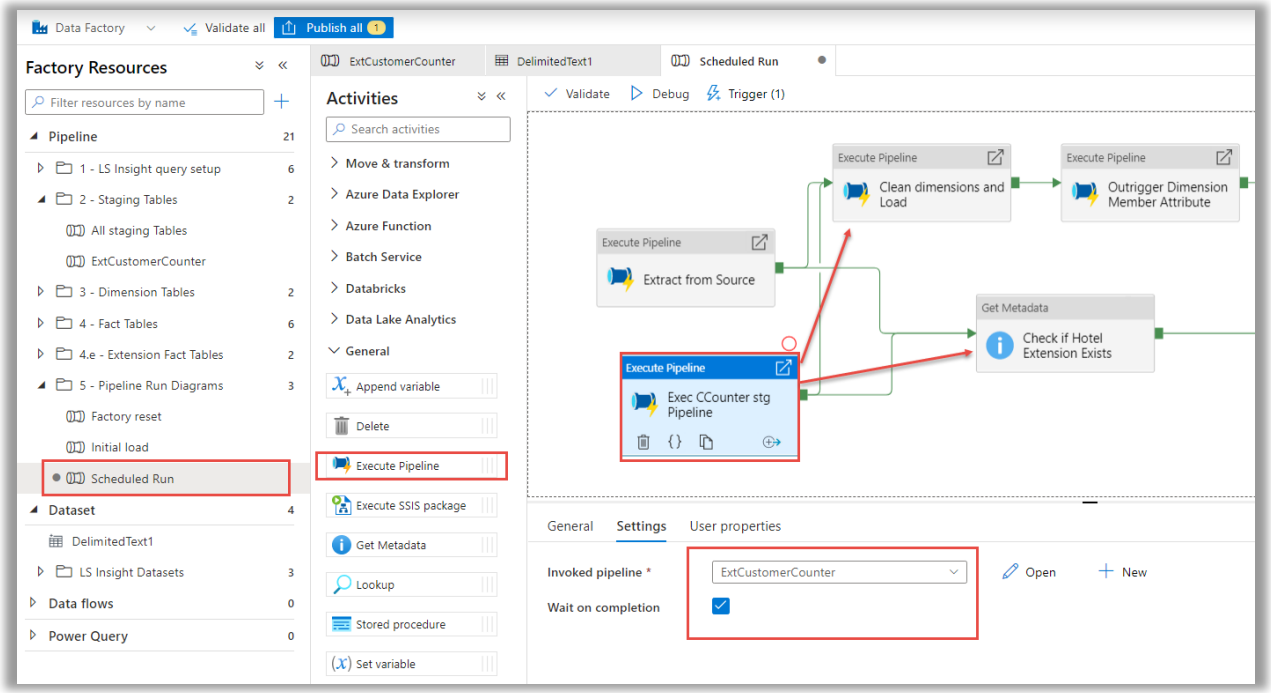
(variables('Control_Table_Table_Name'))



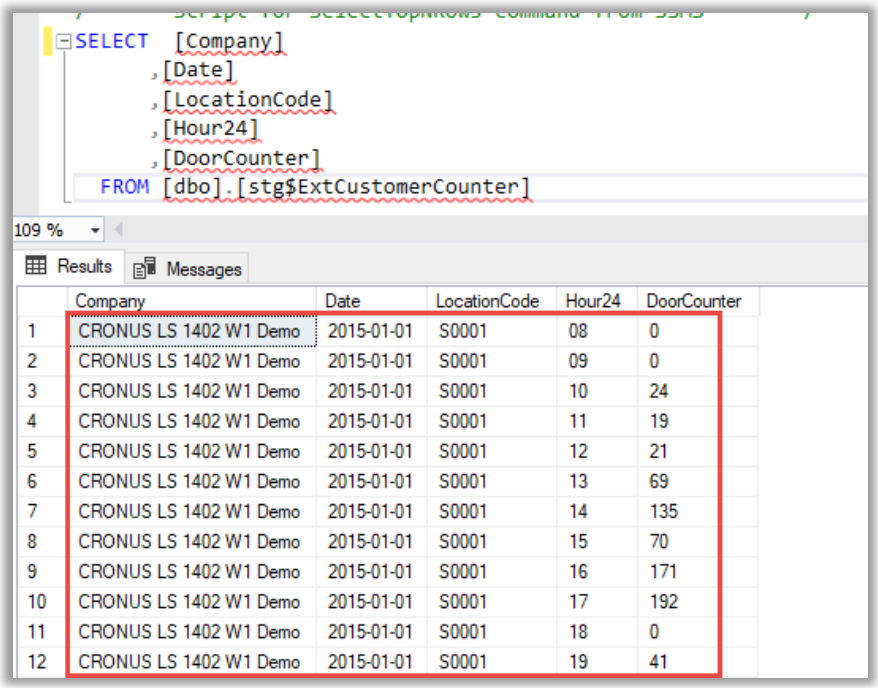

A similar method is used to get the maximum timestamp from the source data and then this information is used in the source query to only load new data to the DW.

### Add staging pipeline to scheduled run

Add the staging pipeline to the Scheduled run pipeline to have the data loaded on the selected schedule with the Analytics schedule:




Now when the scheduled run is triggered, the source data will be written to the Analytics database as a staging table:



```
SELECT  [Company]
      ,[Date]
      ,[LocationCode]
      ,[Hour24]
      ,[DoorCounter]
  FROM [dbo].[stg$ExtCustomerCounter]
```

109 %

Results | Messages

| | Company | Date | LocationCode | Hour24 | DoorCounter |
|---|---|---|---|---|---|
| 1 | CRONUS LS 1402 W1 Demo | 2015-01-01 | S0001 | 08 | 0 |
| 2 | CRONUS LS 1402 W1 Demo | 2015-01-01 | S0001 | 09 | 0 |
| 3 | CRONUS LS 1402 W1 Demo | 2015-01-01 | S0001 | 10 | 24 |
| 4 | CRONUS LS 1402 W1 Demo | 2015-01-01 | S0001 | 11 | 19 |
| 5 | CRONUS LS 1402 W1 Demo | 2015-01-01 | S0001 | 12 | 21 |
| 6 | CRONUS LS 1402 W1 Demo | 2015-01-01 | S0001 | 13 | 69 |
| 7 | CRONUS LS 1402 W1 Demo | 2015-01-01 | S0001 | 14 | 135 |
| 8 | CRONUS LS 1402 W1 Demo | 2015-01-01 | S0001 | 15 | 70 |
| 9 | CRONUS LS 1402 W1 Demo | 2015-01-01 | S0001 | 16 | 171 |
| 10 | CRONUS LS 1402 W1 Demo | 2015-01-01 | S0001 | 17 | 192 |
| 11 | CRONUS LS 1402 W1 Demo | 2015-01-01 | S0001 | 18 | 0 |
| 12 | CRONUS LS 1402 W1 Demo | 2015-01-01 | S0001 | 19 | 41 |

## Add fact table to star schema

### Create a connected fact table

First create the destination table with the correct surrogate keys (SK_*) for the connected dimensions. Name the schema “DW” and the first letter in the table name “f” to distinguish it from staging and dimension tables.

Here is the create script for this example:

```
CREATE TABLE [DW].[fCustomerCounter](
       [Company] [int] NULL,
       [SK_Location] [int] NULL,
       [LocationCode] [nvarchar](100) NULL,
       [Date] [date] NULL,
       [Hour24] [nvarchar](2) NULL,
       [DoorCounter] [int] NULL
) ON [PRIMARY]
```

### Create a Stored Procedure

Create a Stored Procedure that meets your requirements for populating data in the fact table. Make sure to add the surrogate keys (SK_*) for the connected dimensions. The surrogate keys are used in the Power BI reports to determine the table relationships.

Here is an example of the stored procedure for the customer counter data used in this documentation.

It is a good idea to have a naming convention for any extra items in the Analytics database – in this example the affix “Ext” is used.

```
CREATE PROCEDURE [dbo].[ExtfactCustomerCounter]

AS

/* Ensure the stored procedure does not execute unless the source staging table exist */
IF EXISTS (SELECT
                *
        FROM INFORMATION_SCHEMA.TABLES
        WHERE TABLE_NAME = 'stg$ExtCustomerCounter')
BEGIN

SET NOCOUNT ON;

WITH
/* Get the list of companies used */
tCompanies
AS
(SELECT
                dCOM.[SK_Company] AS [Company]
           ,dCOM.[CompanyPrefix]
        FROM [DW].[dCompany] dCOM
        WHERE dCOM.[SK_Company] <> -1),

/* Get the dimension(s) needed to connect to the fact table */
```

```
tLocation
AS
(SELECT
                dLOC.[SK_location]
           ,tCOM.[CompanyPrefix]
           ,dLOC.[LocationCode]
        FROM DW.[dLocation] dLOC
        LEFT OUTER JOIN [tCompanies] tCOM
                ON dLOC.[Company] = tCOM.[Company]),

/* Get the data from the staging table and connect to the dimensions used */
tCustomerCounter
AS
(SELECT
                COALESCE(dCOM.[SK_Company], -1) AS [Company]
           ,COALESCE(tLOC.SK_Location, -1) AS [SK_Location]
           ,sECC.[LocationCode]
           ,[Date]
           ,[Hour24]
           ,[DoorCounter]
        FROM [dbo].[stg$ExtCustomerCounter] sECC
        LEFT JOIN [tLocation] tLOC
                ON sECC.[Company] = tLOC.[CompanyPrefix]
                AND sECC.[LocationCode] = tLOC.[LocationCode]
        LEFT JOIN [DW].[dCompany] dCOM
                ON dCOM.[CompanyPrefix] = sECC.[Company])

/* Use merge to update the fact table */
MERGE [DW].[fCustomerCounter] AS Target USING (SELECT
                [Company]
           ,[SK_Location]
           ,[LocationCode]
           ,[Date]
           ,[Hour24]
           ,[DoorCounter]
        FROM tCustomerCounter) AS Source
ON Target.[Company] = Source.[Company]
        AND Target.[LocationCode] = Source.[LocationCode]
        AND Target.[Date] = Source.[Date]
        AND Target.[Hour24] = Source.[Hour24]

WHEN MATCHED
        THEN UPDATE
                SET [SK_Location] = Source.[SK_Location]
                   ,[DoorCounter] = Source.[DoorCounter]

WHEN NOT MATCHED BY TARGET
        THEN INSERT ([Company]
                , [SK_Location]
                , [LocationCode]
                , [Date]
                , [Hour24]
                , [DoorCounter])
                        VALUES        (Source.[Company],        Source.[SK_Location],Source.[LocationCode],
Source.[Date], Source.[Hour24], Source.[DoorCounter])

/* OPTIONAL if records should be deleted in the DW is they are removed from the source data
Usually records are not deleted from DW

WHEN NOT MATCHED BY SOURCE
```

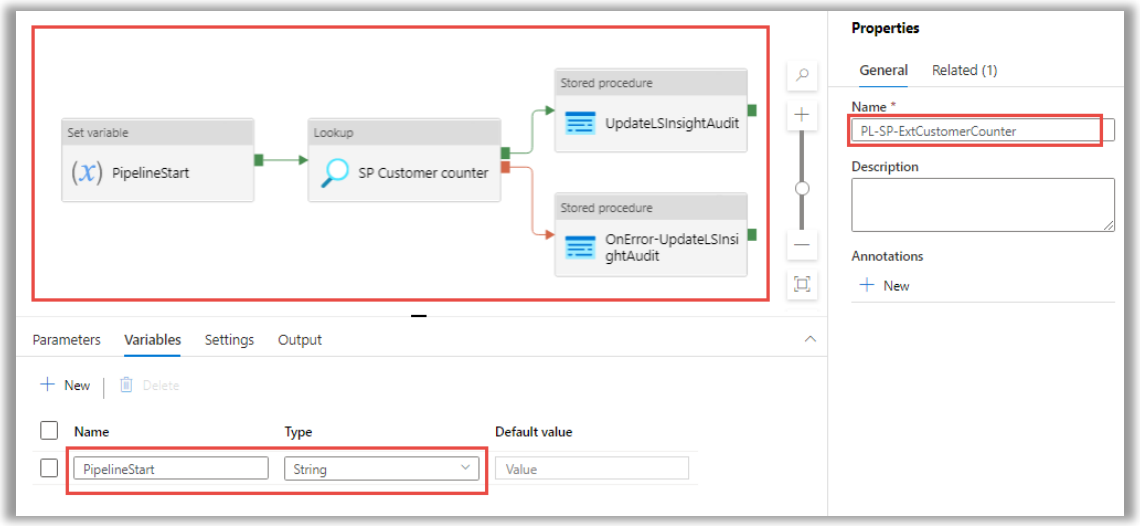
```
THEN DELETE
*/
;
/* SELECT Rowcount is needed for the Azure Data Factory pipeline so the activity has results to determine
successful execution*/
SELECT
        'RowCount' = @@rowcount
;
END
```

Add a fact table Stored Procedure to ADF pipeline

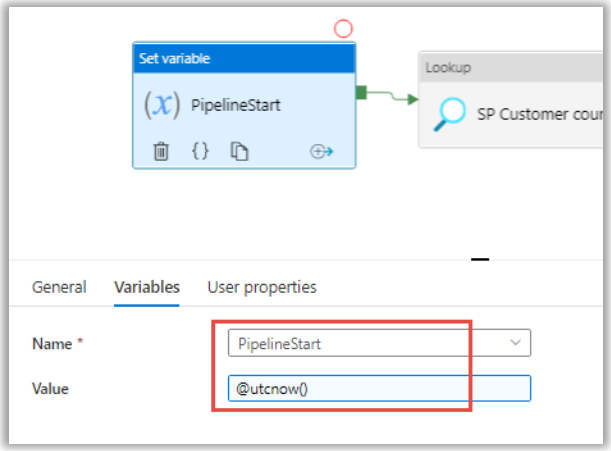
**Create Pipeline to execute SP**

Create a new pipeline and place it in the folder "4.e - Extension Fact Tables". Give the pipeline a descriptive name, for example "PL-SP-ExtCustomerCounter", and create a variable named "PipelineStart".

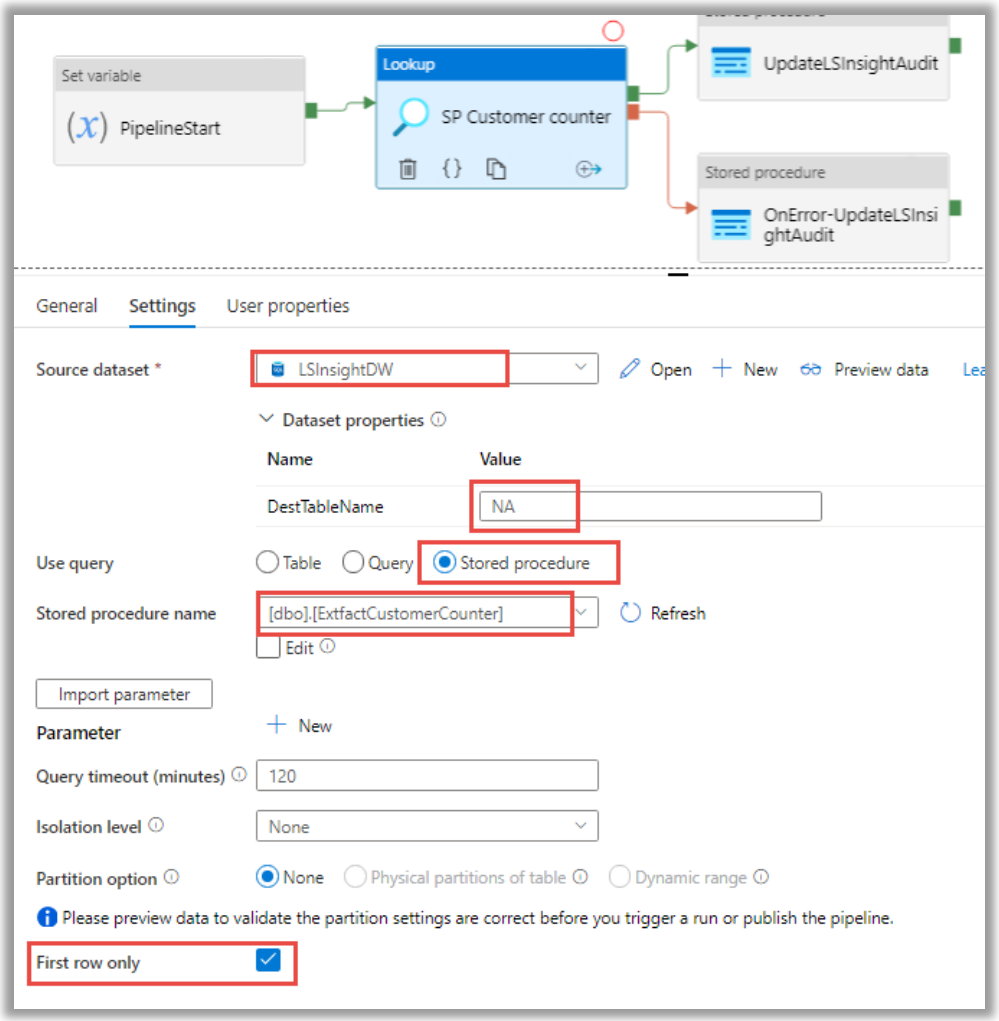
Add the following activities: *Set variable*, Lookup, and 2 instances of Stored procedure. Connect as shown in the image:




Set the Variable to utcnow() using "Add dynamic content":

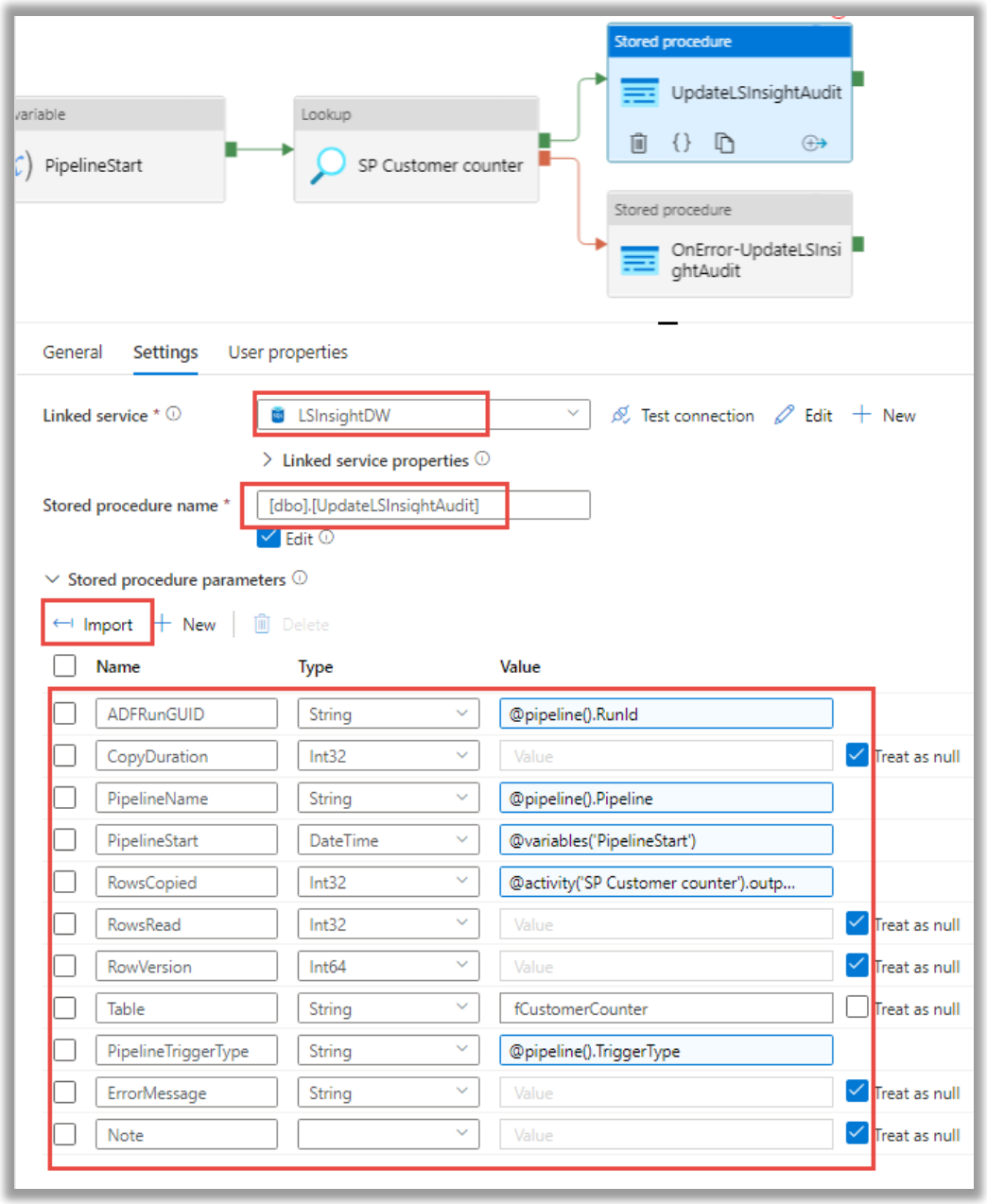


Set the Lookup activity to execute the stored procedure "[dbo].[ExtfactCustomerCounter]":



### Update Analytics Audit

Set the properties for *UpdateAnalyticsAudit* and *OnError-UpdateAnalyticsAudit* to write correct information in the AnalyticsAudit table.




ADFRunGUID = @pipeline().RunId

CopyDuration = Treat AS null

PipelineName = @pipeline().Pipeline

PipelineStart = @variables('PipelineStart')

RowsCopied = @activity('SP Customer counter').output.firstRow.RowCount

RowsRead = Treat AS null

RowVersion is only used when you need to set up Incramental loading - Treat AS null in this scenario

Table = fCustomerCounter (the name of the fact table you are writing to)

PipelineTriggerType = @pipeline().TriggerType

ErrorMessage is not used here

Note – free space to add additional information

**Update AnalyticsAudit on failure**

On the failure activity the following settings apply for the same stored procedure

ADFRunGUID = @pipeline().RunId

CopyDuration = 0

PipelineName = @concat('Failed Run - ', pipeline().Pipeline)

PipelineStart = @variables('PipelineStart')

RowsCopied = 0

RowsRead = 0

RowVersion = 0

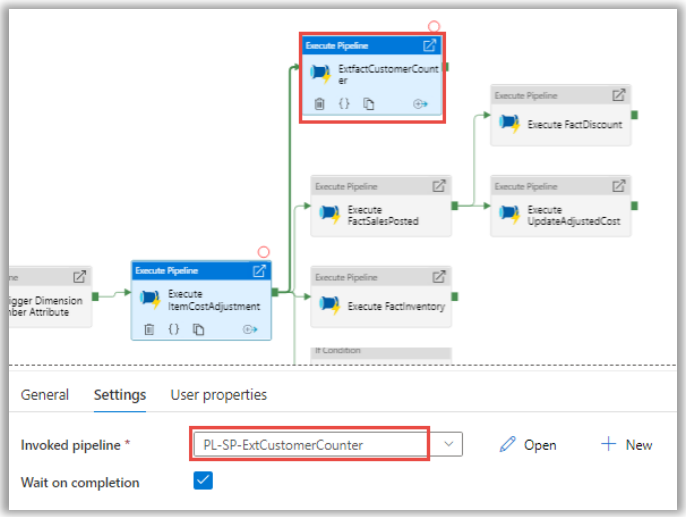
Table = fCustomerCounter (the name of the fact table you are trying to write to)

PipelineTriggerType = @pipeline().TriggerType

ErrorMessage = @activity('SP Customer counter').error.message

**Add Facttable pipeline to scheduled run**

Add an "Execute pipeline" activity to the Scheduled Run pipeline. Depending on your requirements set the depend linage – In this demo the customer counter fact table will start populating after all dimension activities have completed.



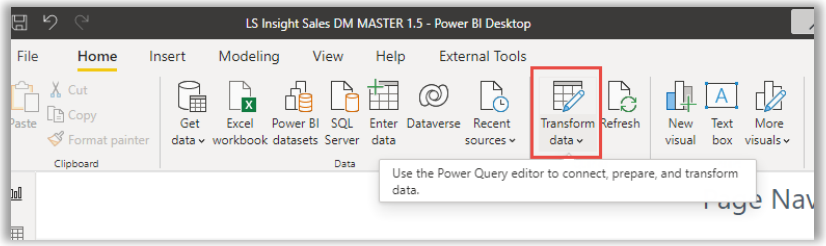
On your next scheduled run the new table will be available in the Analytics DW and you can add to new data to your reports.

## Add data directly in Power BI

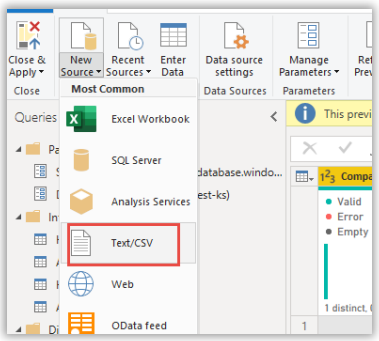
There is also an option to add data tables directly in the data model in Power BI. In this demo the same data source is added to the model and used in a report page. Same or similar steps apply when using the data from the fact table created above. The difference is the source type and with the fact table there is no need to look up the correct surrogate key from the relevant dimensions.

### Get a new source

In Power BI desktop, open *Home – Transform data – Transform data:*



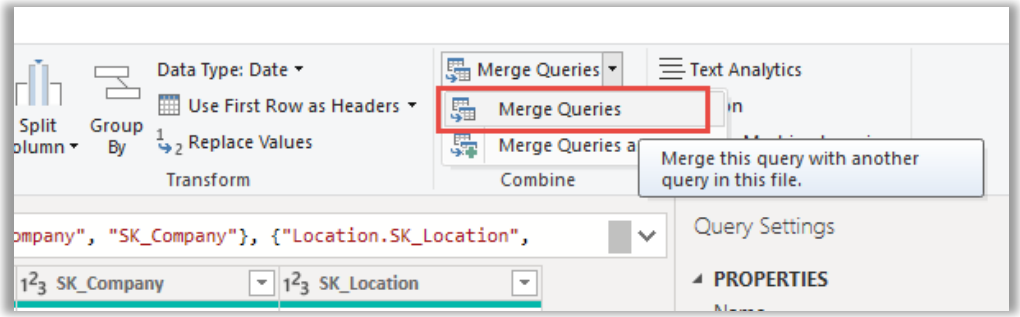

Select *New Source* and in this case *Text/csv:*



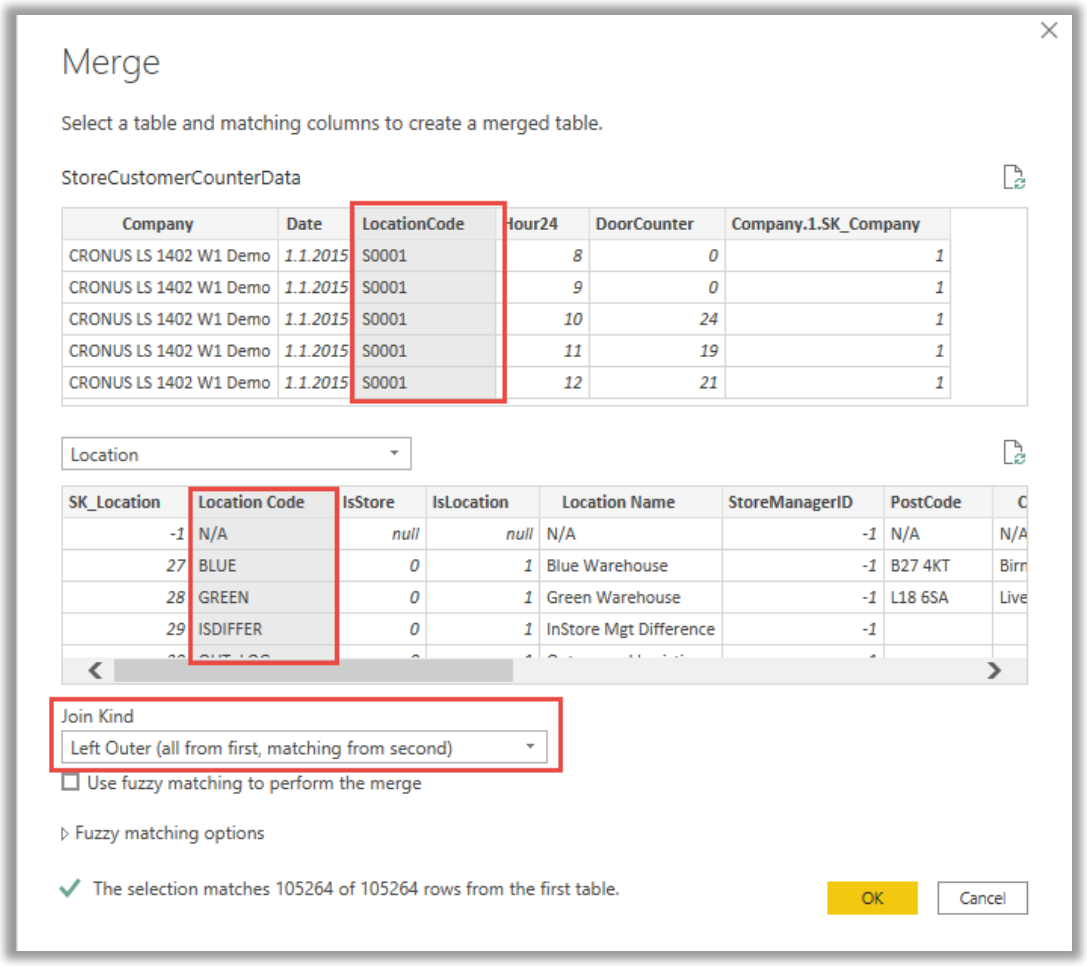

Select the source data and verify columns before loading to the data model.

### Merge to get surrogate keys

Fact tables are linked to dimensions through surrogate keys. Merge queries is a simple method to add the correct keys from the dimensions.



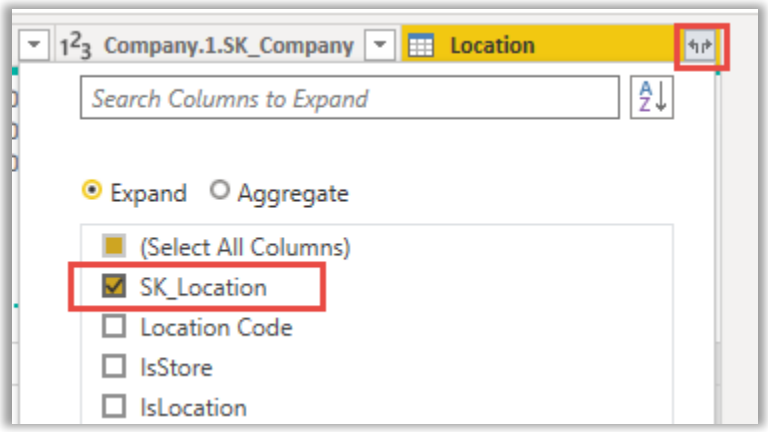
Select the keys that match and click “OK”:




This needs to be done for the Location and Company dimensions (in this example).

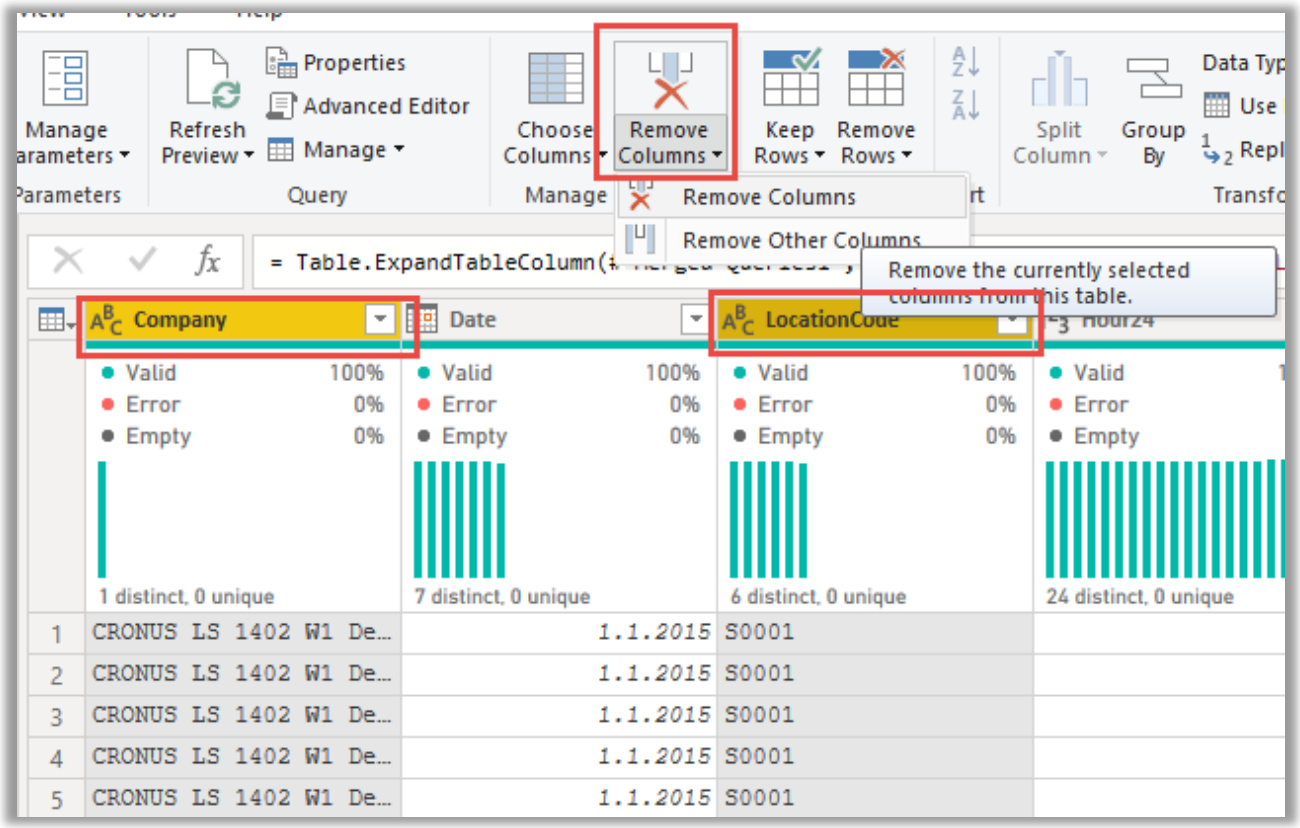
### Expand merged tables

You only need to add the surrogate key from the linked tables. This is done by expanding the linked table and selecting the desired columns:



### Remove unnecessary columns

After the surrogate keys have been added to the new fact table, the business keys in the fact table can be removed as they will never be used.




### M query

The final M query looks like this:

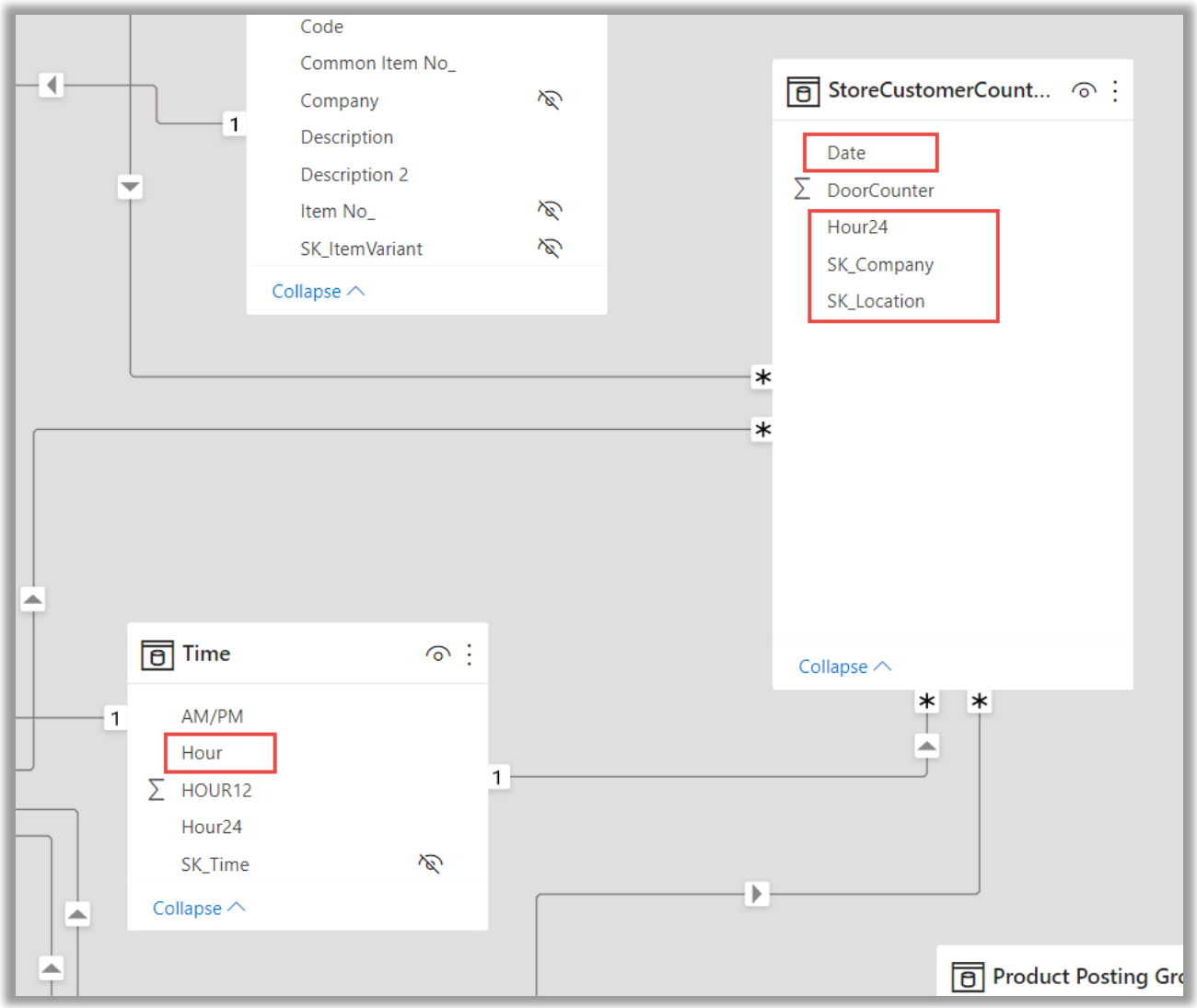
```
let

    Source = Csv.Document(File.Contents("C:\CustomerCounter\StoreCustomerCounterData.txt"),[Delimiter="
        ", Columns=5, Encoding=1252, QuoteStyle=QuoteStyle.None]),

    #"Promoted Headers" = Table.PromoteHeaders(Source, [PromoteAllScalars=true]),

    #"Changed Type" = Table.TransformColumnTypes(#"Promoted Headers",{{"Company", type text}, {"Date",
type date}, {"LocationCode", type text}, {"Hour24", Int64.Type}, {"DoorCounter", Int64.Type}}),

    #"Merged Queries" = Table.NestedJoin(#"Changed Type", {"Company"}, Company, {"Company Name"},
"Company.1", JoinKind.LeftOuter),

    #"Expanded Company.1" = Table.ExpandTableColumn(#"Merged Queries", "Company.1", {"SK_Company"},
{"Company.1.SK_Company"}),

    #"Merged Queries1" = Table.NestedJoin(#"Expanded Company.1", {"LocationCode"}, Location, {"Location
Code"}, "Location", JoinKind.LeftOuter),
```

```
#"Expanded Location" = Table.ExpandTableColumn(#"Merged Queries1", "Location", {"SK_Location"}, {"Location.SK_Location"}),

#"Removed Columns" = Table.RemoveColumns(#"Expanded Location",{"Company", "LocationCode"}),

#"Renamed Columns" = Table.RenameColumns(#"Removed Columns",{{"Company.1.SK_Company", "SK_Company"}, {"Location.SK_Location", "SK_Location"}})

in

#"Renamed Columns"
```

### Define relationships in PowerBI

The steps from this point apply both for using a custom fact table in the Analytics DW or for using the steps above to add data directly to the data model in Power BI.

Define the relationships from the new fact table to the related dimensions. In this example, the connected dimensions are: Date, Time, Company, and Location.



## Use the new data in a report page

With the new data it is possible to create custom DAX calculations or, depending on the data, use the data directly in a current or new visual.

Here is an example where the average number of customers entering the locations is compared with the average net sale per transaction: