Extending LS Insight

LS Insight v. 2020.5 - 2022.2

Contents

1 Introduction

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

LS Insight 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 LS Insight we provide a basic extendable data warehouse in SQL server database and Power BI reports and measures that work with the data structure of the warehouse.

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

2 Initial setup

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

3 Add Company

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

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

You can add to the **LSInsight$PublisherAffixReg** table using the **Add or Delete App Affix** pipeline in the LS Insight 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 LS Insight, you need to manually run the **Generate LS Insight Query Base** pipeline. To update the query base, you run the **Scheduled Run** or wait for the next scheduled run. Data from the companies you added/removed will be added or removed from the staging, dimension, and fact tables when the **Scheduled Run** pipeline runs.

4 Staging

All staging tables in LS Insight are based on one base table in Business Central (BC) plus its extensions. The base table and the extension tables are joined together to form one large table with all columns from both base and extension tables.

The **Item** table is a perfect example.

In BC there is a base Item table and then you have a few extension Item tables that come with LS Central.

Each Item extension table has an extension GUID. The base table also has GUID that is the same for all BC base tables.

Each column of an extension table has the appropriate prefix, so if you have more extensions in your LS Central instance, they will also be added to the Item staging table in LS Insight.

Stand-alone extension tables that are tables that do not extend an existing base table have a name that is affixed with a partner prefix or suffix. For LS Central this prefix is LSC. All prefixes that you want to remove from table names need to be added to the **LSInsight$PublisherAffixReg** table.

The names for those tables changed from LS Central version 17.4 to 17.5.

When the table names were prefixed along with the AL objects, some of the table names became too long and were changed. So, to not have to change the name of the staging tables in LS Insight we created a mapping from the original table names to the prefixed ones. This mapping is stored in the **LSInsight$SourceTablesMap** table.

In NAV versions older than 14.2 tables were not extended in this manner.

Adding new affix

If you want to extend LS Insight with staging tables from an extension that you have added to your LS Central instance, you need to start by adding the affix to the **LSInsight$PublisherAffixReg** table.

You can add to the **LSInsight$ PublisherAffixReg** table using the **Add or Delete App Affix** pipeline in the LS Insight ADF.

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

- AppID (extension GUID from LS Central table)
- AppName (name of installed extension)
- Publisher (name of publisher of Extension)
- Prefix (the three-letter prefix used, if used. Not required)
- Suffix (the three-letter suffix used, if used. Not required)

These are the fields you need to populate to add a record to the table.

In addition, there is the DeleteApp field that decides whether a record is added or deleted from the table. It is set to FALSE by default, but if it is changed to TRUE it deletes a matching record from the

table. When deleting, all you need to do is enter the Extension GUID in the AppID field and change the DeleteApp value to TRUE. You can only add or delete one App record at a time.

Adding new prestaging table – LS Central SaaS only

If you are running LS Insight against LS Central SaaS, you need to add the prestaging tables manually to the LS Insight database.

This can be done by accessing the Scheduler server database, that you are using to schedule the replication jobs, and saving a creation script for the tables you want to add. If the table is extended, you need to save scripts for the base and extension tables.

You can then either modify the creation script like we have done for the prestaging scripts included in the product package (so it queries the **LSInsight$Companies** table to create a table for each company), or you can have a script per company.

The prestaging scripts are included in the product package under the

.. \Modules\LSInsight\Resources\Prestaging tables folder

Once you have created the prestaging script, you need to add the table as subjob to the appropriate scheduler jobs to replicate the data from the LS Central SaaS source table to the LS Insight prestaging table.

Adding new table mapping

If you need to change the name of a prefixed table to a regular staging table name, you can enter a mapping record to the **LSInsight$ShortTableNameMap**.

You can add to the **LSInsight$ShortTableNameMap** table using the **AddorDeleteShortTable NameMapping** pipeline in the LS Insight ADF.

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

- OrgTableName (the name you want to use for the staging table)
- NewTableName (the name of the table with affix in LS Central)
- DeleteRow (FALSE by default, set to TRUE to delete record)

Adding new staging table

|	SourceTableName	IncludeTable
1	Company	TRUE
2	Country_Region	TRUE
3	Customer	TRUE
4	Dimension Value	TRUE
5	Division	TRUE
6	Gen_ Business Posting Group	TRUE
7	Gen_ Product Posting Group	TRUE
8	Item	TRUE
9	Item Category	TRUE
10	Item Ledger Entry	TRUE
11	Item Status	FALSE
12	Item Status Link	FALSE
13	Item Variant	TRUE
14	Location	TRUE

To add a new staging table to LS Insight you simply add it to the **LSInsight$SourceTables** table. You need to add the **table name** without any GUID or Company name and set the Include field to **TRUE**.

If the Include Table field is set to false, the table will not be included in LS Insight staging.

Currently, all the core tables used in LS Insight have been added to this table in the template database, so you only need to add to it if you want to add tables that are currently not used in LS Insight.

You can add to the **LSInsight$SourceTables** table using the **Add or Delete Source Tables** pipeline in the LS Insight ADF.

When you add the trigger for the pipeline you are prompted with the Source table field, and you can add or delete one table at a time. If you add the value true to the DeleteSourceTab field, the source table name you specify will be deleted from the table, if it exists.
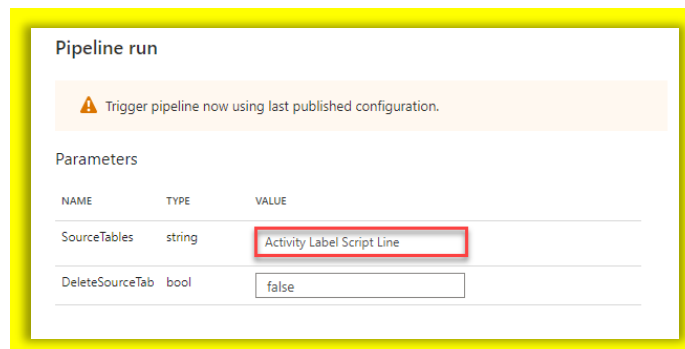

When the pipeline has run and entry for the table has been added to **LSInsight$SourceTables** table.
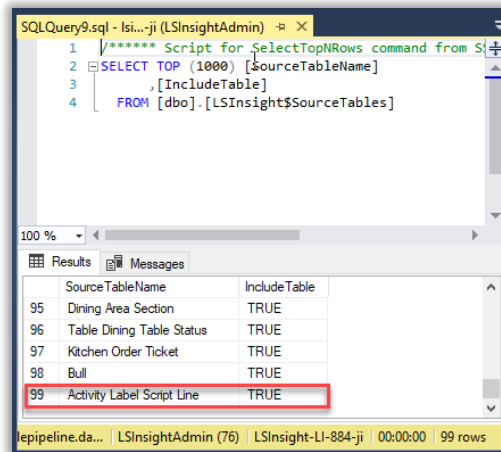
 Then you can move on and run the **Generate LS Insight Query Base** pipeline.
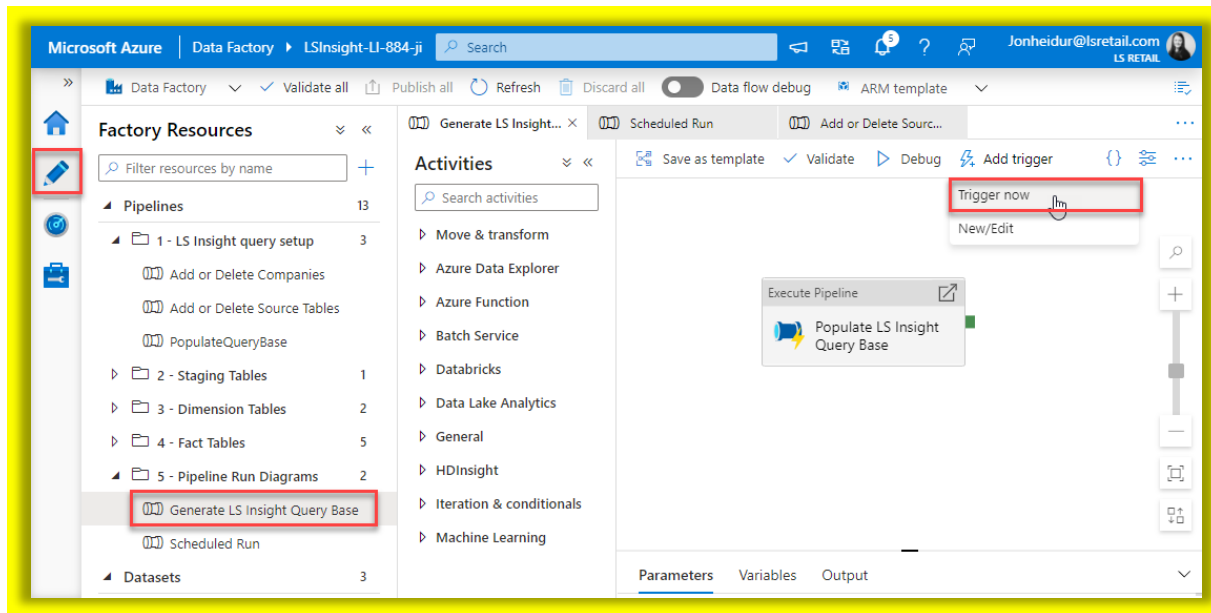
This pipeline will fetch all the information about the tables from LS Central, create the query for the new table and write this information to the **LSInsight$QUERY BASE** table, in addition to updating the queries for the other staging tables, if anything has changed in the LS Central database since this was run initially.
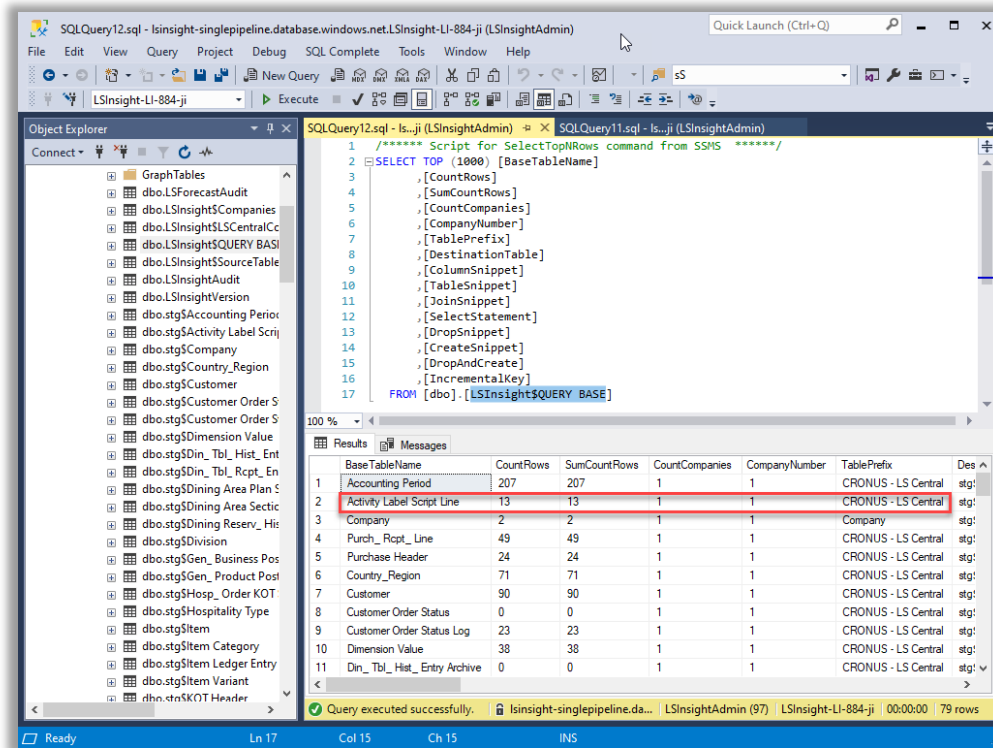
The new staging table will now be created and populated during the next run of the **Scheduled Run** pipeline or, if you want to update the DW right away, you can trigger the pipeline manually.
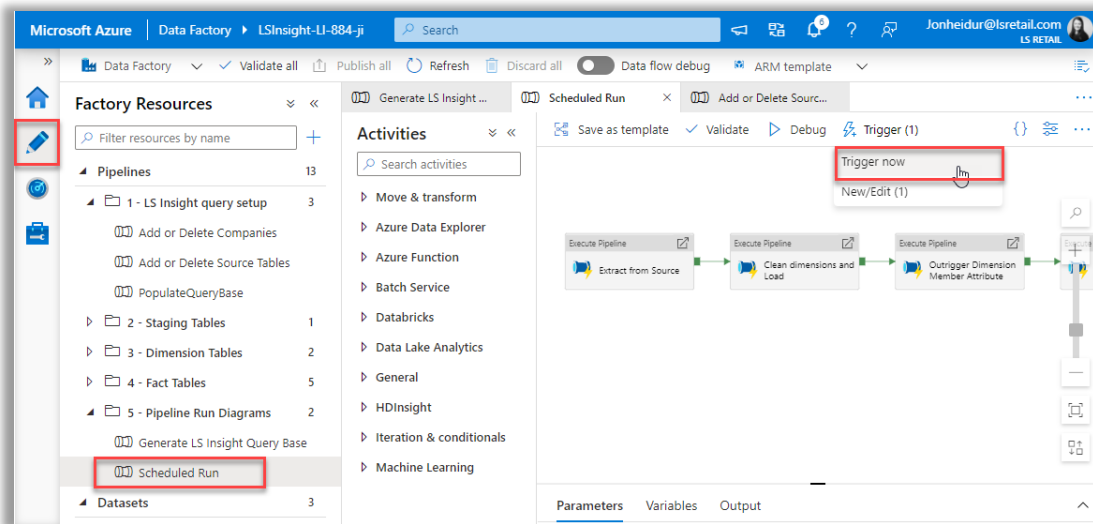
Once the pipeline has run the staging table should be populated. This does, however, not have any impact on the dimension or fact tables since those are populated using stored procedures that need to be created in the database according to the star schema design rules.

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.

5 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 LS Insight team and are included in the LS Insight database.

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.
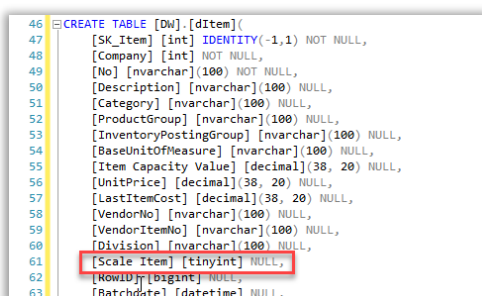
Add column to dimension

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 LS Insight 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 LS Insight database was provided in the deployment summary) and then following these steps:

1) In the LS Insight database expand Tables.
2) Select the **dItem** table
3) Right-click and select Script Table as > DROP and CREATE To > New query editor window.
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.

```
46  CREATE TABLE [DW].[dItem](
47      [SK_Item] [int] IDENTITY(-1,1) NOT NULL,
48      [Company] [int] NOT NULL,
49      [No] [nvarchar](100) NOT NULL,
50      [Description] [nvarchar](100) NULL,
51      [Category] [nvarchar](100) NULL,
52      [ProductGroup] [nvarchar](100) NULL,
53      [InventoryPostingGroup] [nvarchar](100) NULL,
54      [BaseUnitOfMeasure] [nvarchar](100) NULL,
55      [Item Capacity Value] [decimal](38, 20) NULL,
56      [UnitPrice] [decimal](38, 20) NULL,
57      [LastItemCost] [decimal](38, 20) NULL,
58      [VendorNo] [nvarchar](100) NULL,
59      [VendorItemNo] [nvarchar](100) NULL,
60      [Division] [nvarchar](100) NULL,
61      [Scale Item] [tinyint] NULL,
62      [RowID] [bigint] NULL,
63      [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 LS Insight 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.

```
114  tItem
115  AS
116  (SELECT
117          sITM.[CompanyPrefix]
118          ,sITM.[No_] AS [No]
119          ,sITM.[Description]
120          ,COALESCE(tITC.[Description], '') AS [Category]
121          ,COALESCE(tRPG.[Description], '') AS [ProductGroup]
122          ,sITM.[Inventory Posting Group]
123          ,sITM.[Base Unit of Measure]
124          ,sITM.[Item Capacity Value]
125          ,COALESCE(tSAP.[Unit Price], 0) AS [Unit Price]
126          ,CASE sITM.[Costing Method]
127              WHEN 4 THEN COALESCE(IIF(sITM.[Standard Cost] = 0, NULL, sITM.[Standard
128              ELSE COALESCE(IIF(sITM.[Last Direct Cost] = 0, NULL, sITM.[Last Direct
129          END AS [LastItemCost]
130          ,sITM.[Vendor No_]
131          ,sITM.[Vendor Item No_]
132          ,COALESCE(tDIV.[Description], '') AS [DivisionName]
133          ,sITM.[Scale Item]
134          ,MAX(sITM.[bigint_timestamp]) AS [RowID]
135          ,GETUTCDATE() AS [BatchDate]
136      FROM [Stg$Item] sITM
137      LEFT JOIN [tItemCategory] tITC
138          ON sITM.[CompanyPrefix] = tITC.[CompanyPrefix]
139          AND sITM.[Item Category Code] = tITC.[Code]
140      LEFT JOIN [tProductGroup] tRPG
141          ON sITM.[CompanyPrefix] = tRPG.[CompanyPrefix]
142          AND sITM.[Retail Product Code] = tRPG.[Code]
143      LEFT JOIN [tDivision] tDIV
144          ON sITM.[CompanyPrefix] = tDIV.[CompanyPrefix]
145          AND sITM.[Division Code] = tDIV.[Code]
146      LEFT JOIN [tSalesPrice] tSAP
147          ON sITM.[CompanyPrefix] = tSAP.[CompanyPrefix]
148          AND sITM.No_ = tSAP.[Item No_]
149      GROUP BY sITM.CompanyPrefix
150          ,sITM.[No_]
151          ,sITM.[Description]
152          ,tITC.[Description]
153          ,tRPG.[Description]
154          ,sITM.[Inventory Posting Group]
155          ,sITM.[Base Unit of Measure]
156          ,sITM.[Item Capacity Value]
157          ,sITM.[Unit Price]
158          ,COALESCE(IIF([Standard Cost] = 0, NULL, [Standard Cost]), IIF([Last Di
159          ,COALESCE(IIF([Last Direct Cost] = 0, NULL, [Last Direct Cost]), IIF([U
160          ,sITM.[Vendor No_]
161          ,sITM.[Vendor Item No_]
162          ,tDIV.[Description]
163          ,tSAP.[Unit Price]
164          ,sITM.[Costing Method]
165          ,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.

```
166  MERGE [DW].[dItem] AS Target USING (SELECT
167        COALESCE(dCOM.[SK_Company], -1) AS [Company]
168        ,tITM.[No]
169        ,tITM.[Description]
170        ,tITM.[Category]
171        ,tITM.[ProductGroup]
172        ,tITM.[Inventory Posting Group]
173        ,tITM.[Base Unit of Measure]
174        ,tITM.[Item Capacity Value]
175        ,tITM.[Unit Price]
176        ,tITM.[LastItemCost]
177        ,tITM.[Vendor No_]
178        ,tITM.[Vendor Item No_]
179        ,tITM.[DivisionName]
180        ,tITM.[Scale Item]
181        ,tMRI.[RowID]
182        ,tITM.[BatchDate]
183      FROM [tItem] tITM
184    RIGHT JOIN [tMAXRowIDItem] tMRI
185        ON tITM.[CompanyPrefix] = tMRI.[CompanyPrefix]
186        AND tITM.[No] = tMRI.[No_]
187        AND tITM.[RowID] = tMRI.[RowID]
188    LEFT JOIN [DW].[dCompany] dCOM
189        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 LS Insight 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 **LSInsight$SourceTables** table. Using the **Add or Delete Source Tables** pipeline in the Azure data factory.
2. Re-run the **Populate Query Base** pipeline in the Azure data factory.
3. Run the **Scheduled Run** pipeline to retrieve the data from the new tables.
4. Create a new dimension table with the columns you want to include.
5. 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'.
6. 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'.

6 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 LS Insight database was provided in the deployment summary) and then following these steps:

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

```
ALTER TABLE [DW].[fDiscounts]
  ADD [Points] [decimal](38, 20) NULL
 GO
```

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 LS Insight 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.

```
70   /*Temp table for Discount entries*/
71   tDiscounts
72   AS
73   (SELECT
74           sTDE.[CompanyPrefix]
75           ,CAST(sTRH.[Date] AS DATE) AS [Date]
76           ,CAST(DATEADD(mi, DATEDIFF(mi, 0, sTRH.[Time]), 0) AS TIME) AS [Time]
77           ,tLOC.[SK_Location]
78           ,tOFF.[SK_Offer]
79           ,tPOT.[SK_POSTerminal]
80           ,CASE sTDE.[Offer Type]
81                   WHEN 0 THEN 'Promotion'
82                   WHEN 1 THEN 'Deal'
83                   WHEN 2 THEN 'Multibuy'
84                   WHEN 3 THEN 'Mix&Match'
85                   WHEN 4 THEN 'Disc. Offer'
86                   WHEN 5 THEN 'Total Discount'
87                   WHEN 6 THEN 'Tender Type'
88                   WHEN 7 THEN 'Item Point'
89                   WHEN 8 THEN 'Line Discount'
90                   WHEN 9 THEN 'Customer'
91                   WHEN 10 THEN 'Infocode'
92                   WHEN 11 THEN 'Member Point'
93                   WHEN 12 THEN 'Coupon'
94                   WHEN 13 THEN 'Total'
95                   WHEN 14 THEN 'Line'
96                   ELSE 'No Offer type'
97               END AS [Offer Type]
98           ,sTDE.[Receipt No_]
99           ,sTDE.[Line No_]
100          ,CAST(sTDE.[Transaction No_] AS NVARCHAR(100)) AS [TransactionNo]
101          ,sTDE.[Discount Amount]
102          ,sTDE.[Points]
103          ,sTDE.bigint_timestamp AS [RowID]
104      FROM [stg$Trans_ Discount Entry] sTDE
105      LEFT OUTER JOIN [stg$Transaction Header] sTRH
106          ON sTDE.[Store No_] = sTRH.[Store No_]
107          AND sTDE.[POS Terminal No_] = sTRH.[POS Terminal No_]
108          AND sTDE.[Transaction No_] = sTRH.[Transaction No_]
109          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 LS Insight 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.

7 Third-party data

Here are the recommended steps you need to take to add third-party data to LS Insight. In this example you will be adding customer counter data from file. We will first describe how to add the data to the LS Insight 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 LS Insight Azure Data Factory. You will find all resources in "All resources".
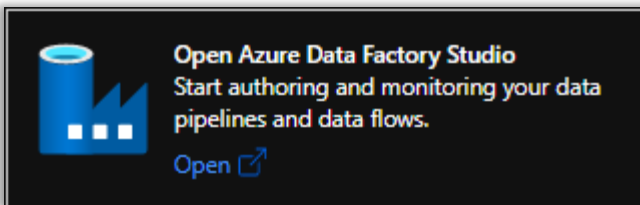
Staging data

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":
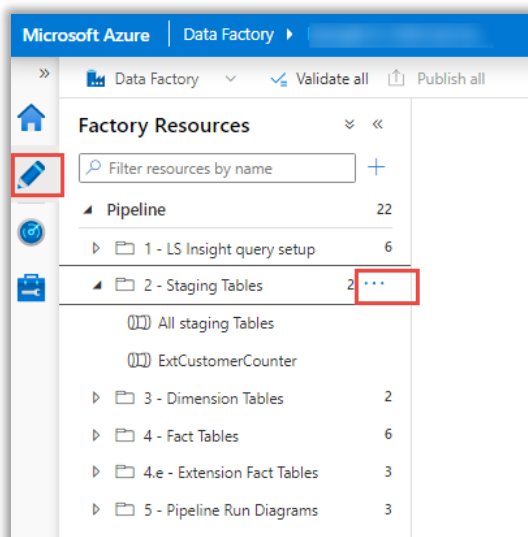
Select to create a new pipeline in the selected folder by clicking **New pipeline**. And give it a descriptive name. It is a good idea to add an affix to the name to distinguish it from LS Insight pipelines.
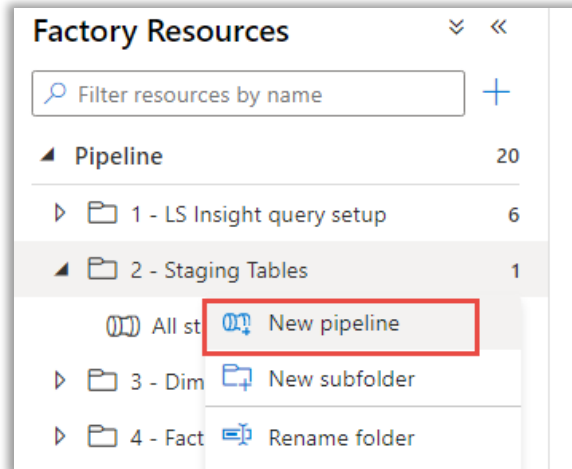
Add Pipeline variables

Add a pipeline variable name "PipelineStart" to hold value of utcnow() to be able to log the pipeline start in the LSInsightAudit table in the LS Insight database.
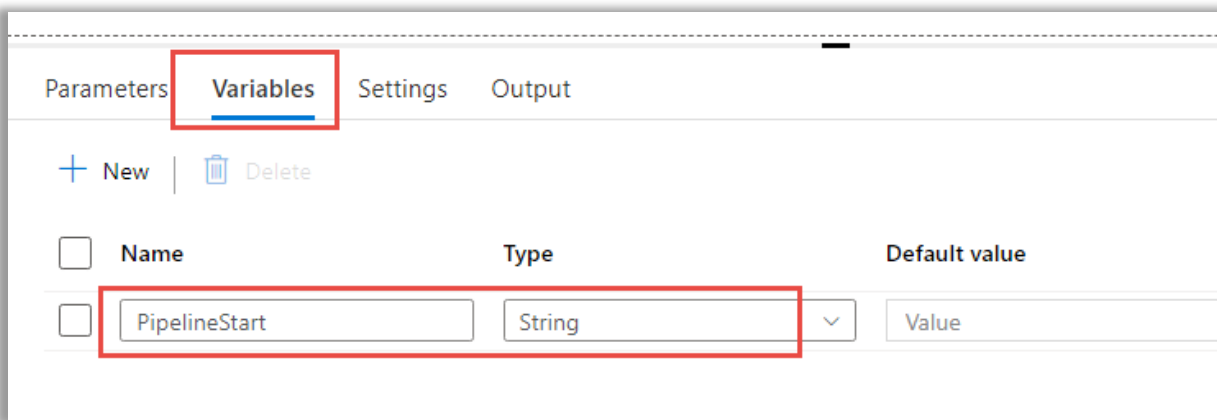
Set the current timestamp using the "Set variable" activity using "Add dynamic content"
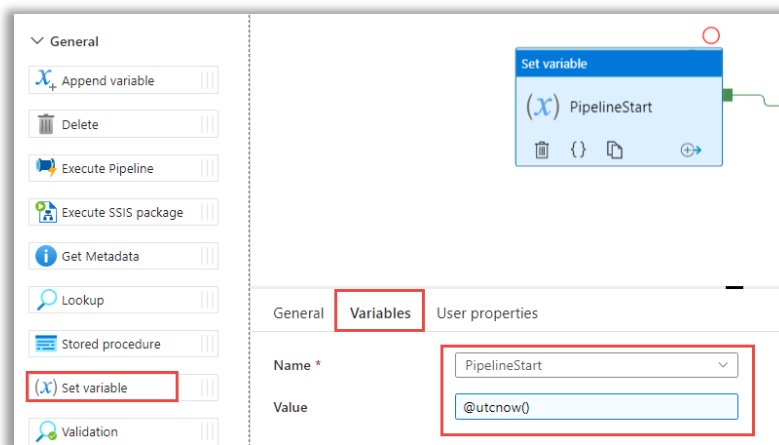
Copy Activity

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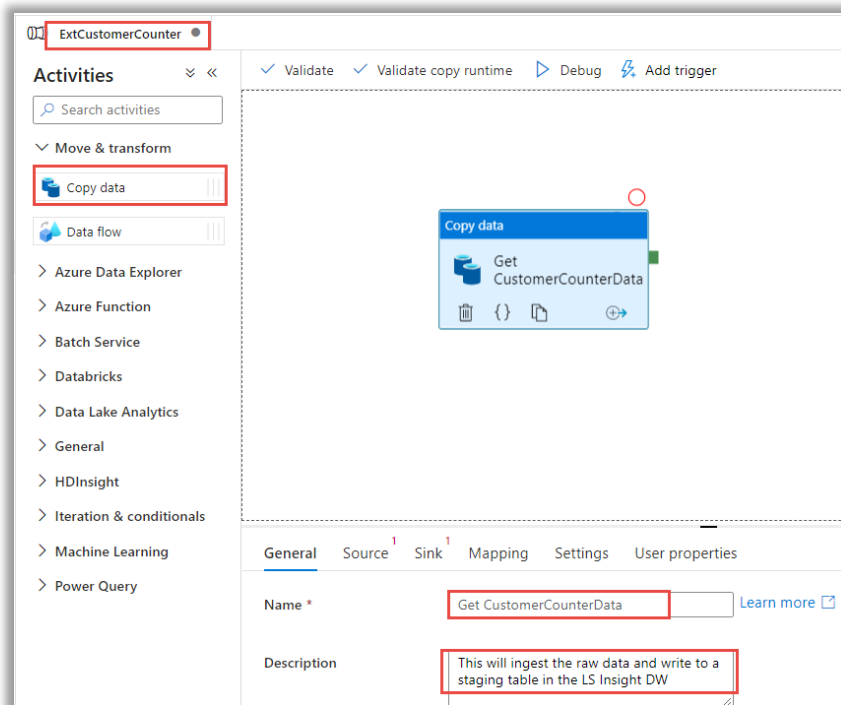
Create a new source connection

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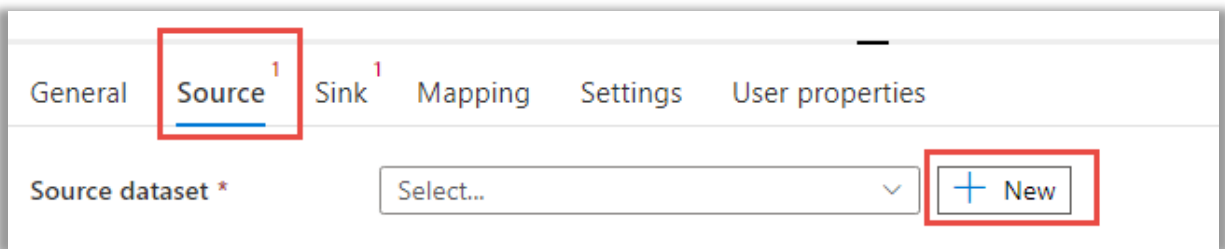
For this example, you will be using a flat file connection – you can choose from over 200 connectors, depending on your data.
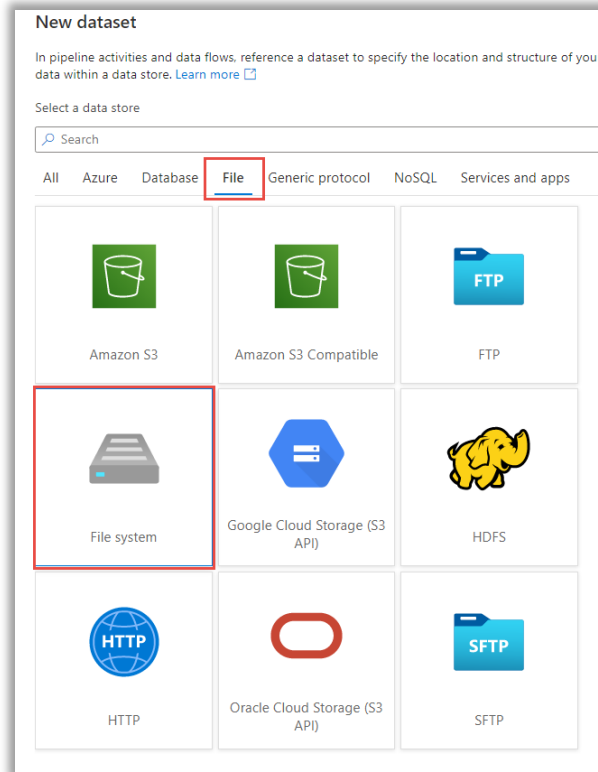
Create a connection to the host.

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.
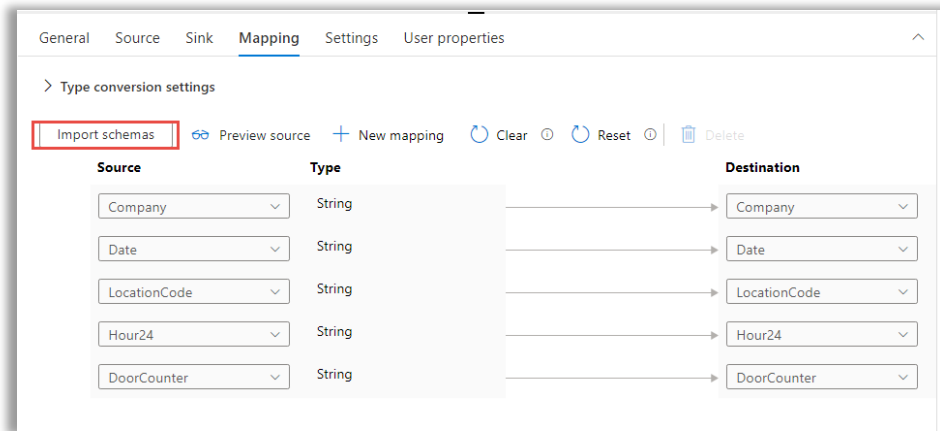
Add Write to audit table action

Add a failure activity to the copy action to be able to trigger an activity when the copy activity has a failure.
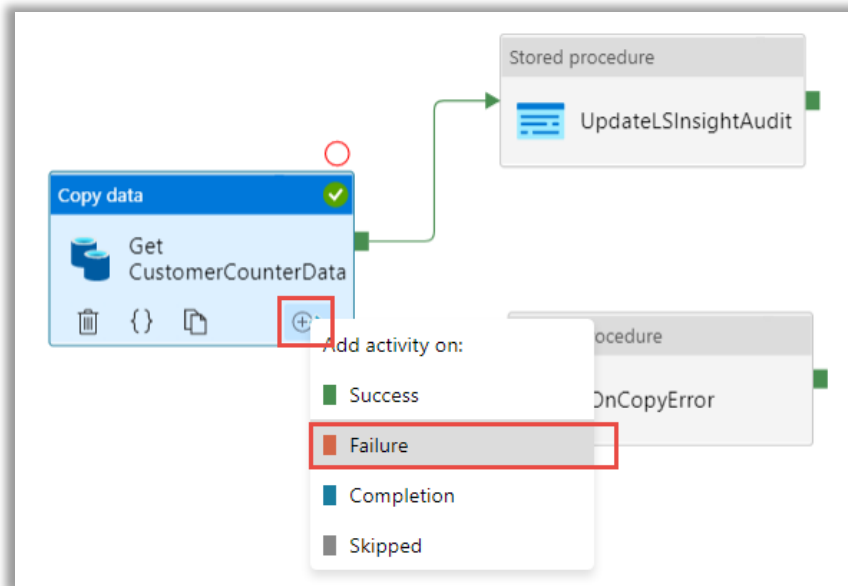
Add two "Stored procedure" activities that will both execute [dbo].[UpdateLSInsightAudit] and connect Success to *UpdateLSInsightAudit* and failure to *OnCopyError*
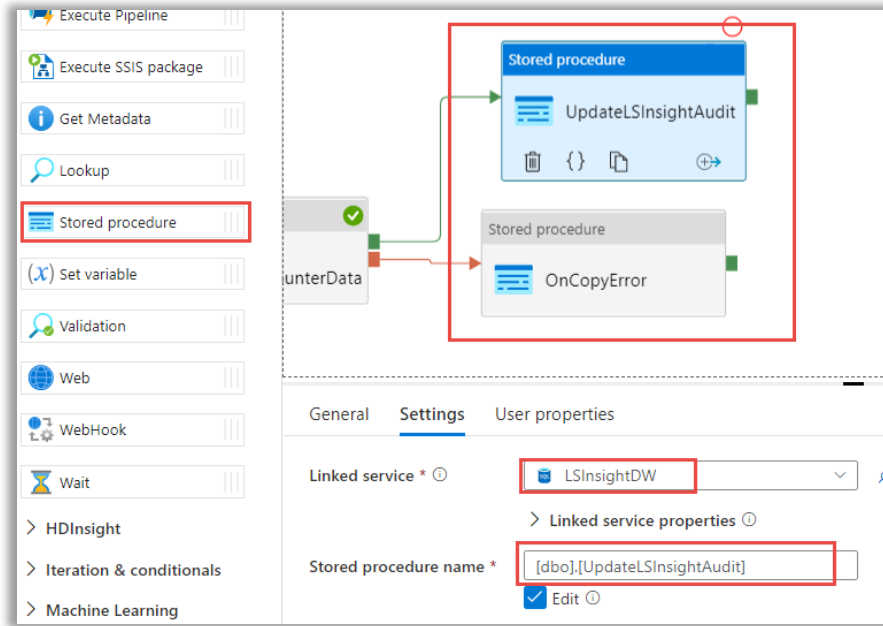
Update LSInsightAudit on success

Set the SP variables as follows for the success activity – *UpdateLSInsightAudit:*
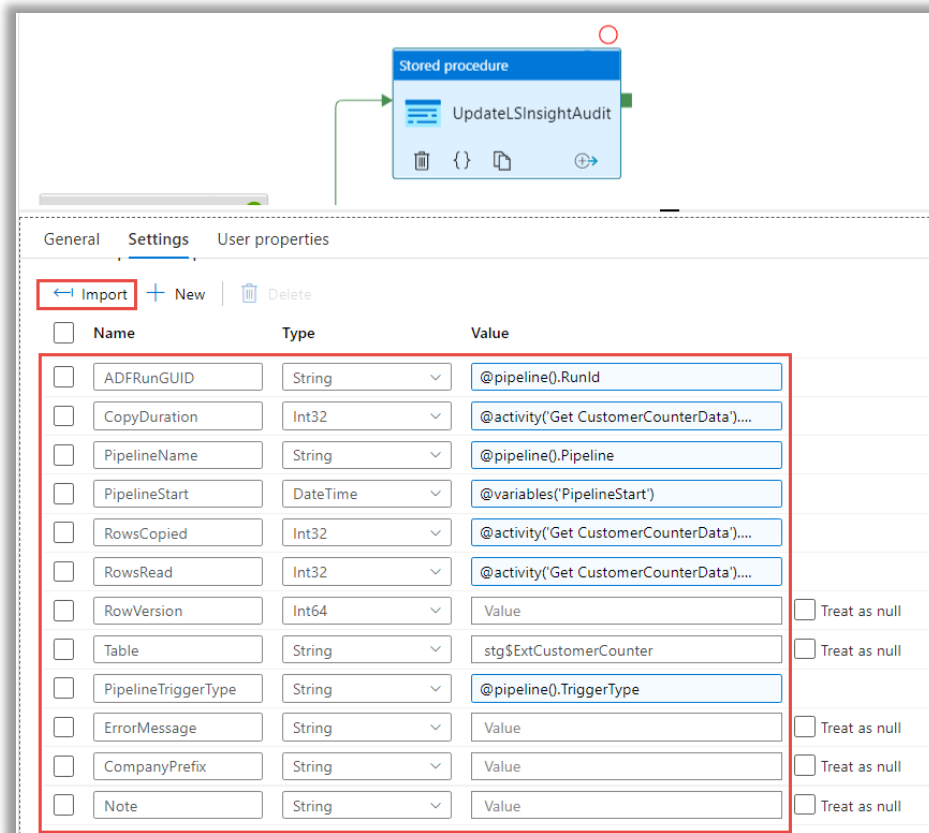
ADFRunGUID = @pipeline().RunId

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 LSInsightAudit 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:
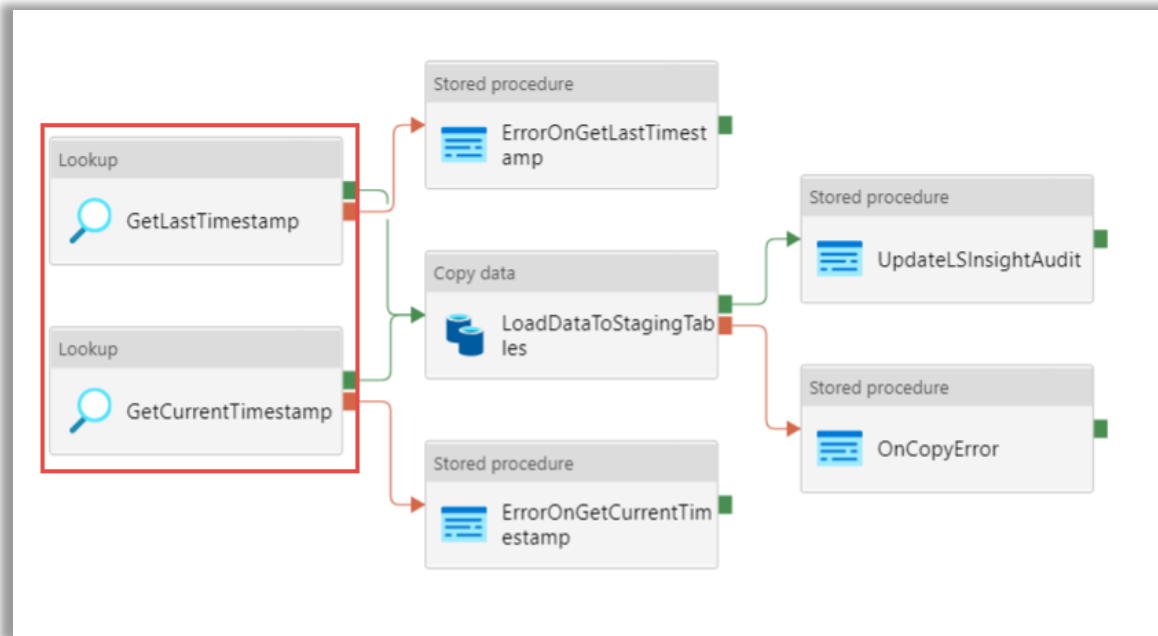
The last timestamp is extracted from the table LSInsightAudit

 (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 LS Insight schedule:
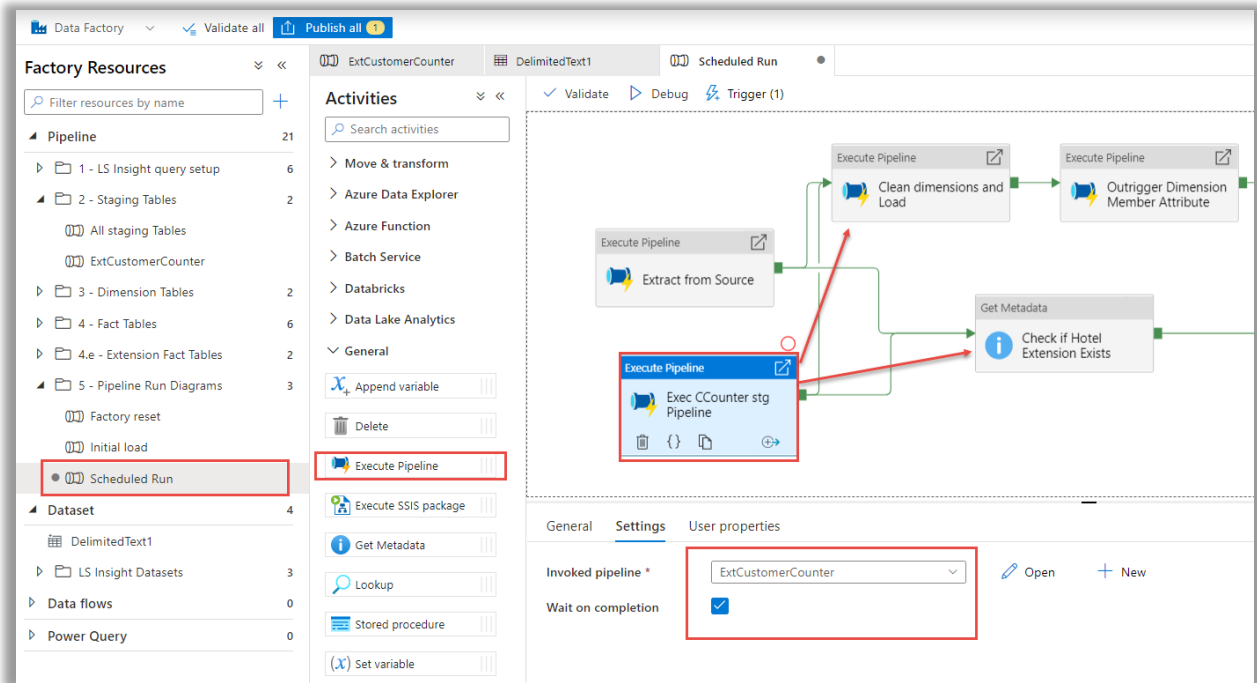
Now when the scheduled run is triggered, the source data will be written to the LS Insight database as a staging table:
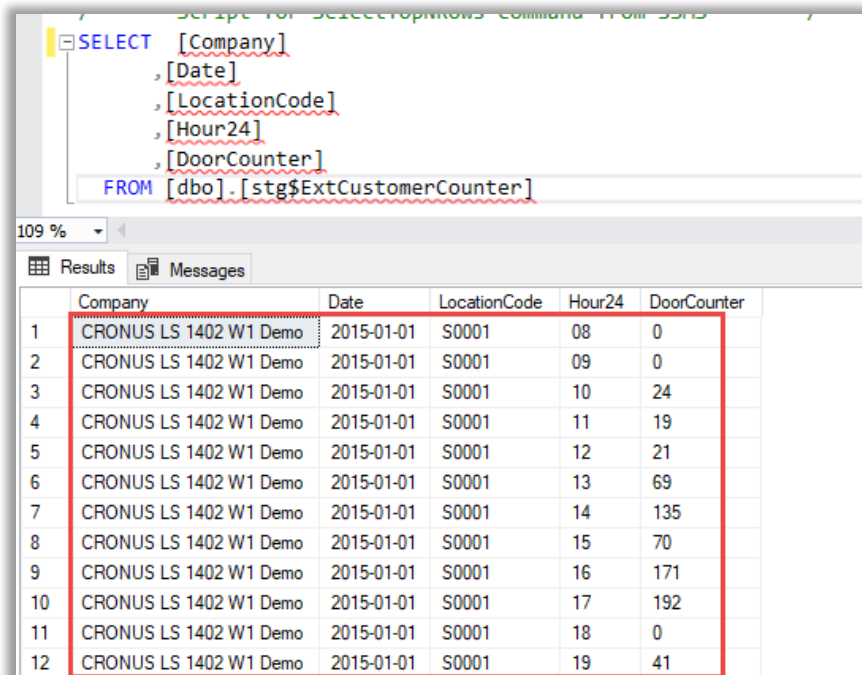
Add fact table to star schema

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 LS Insight 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 */
```

```sql
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:
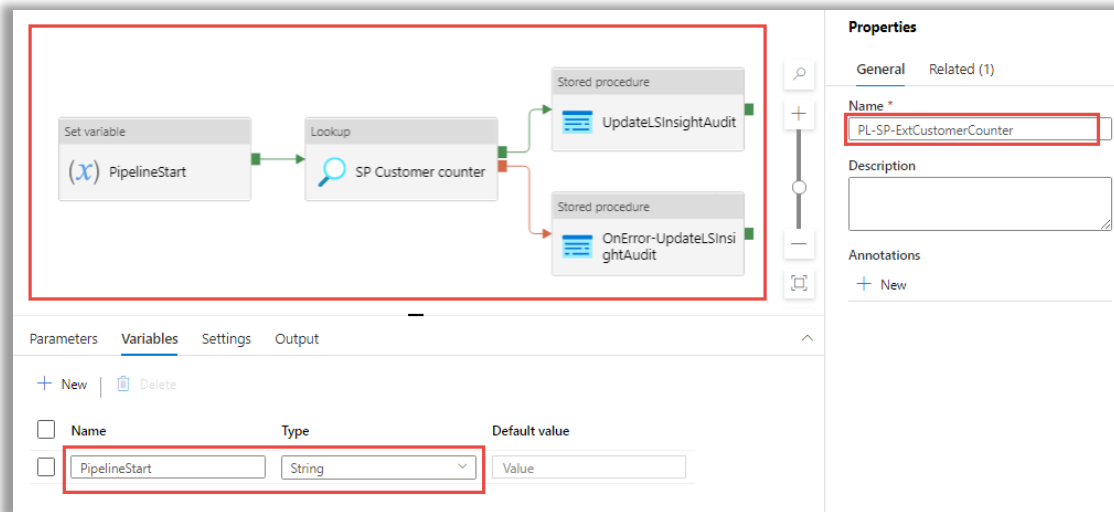
Set the Variable to utcnow() using "Add dynamic content":
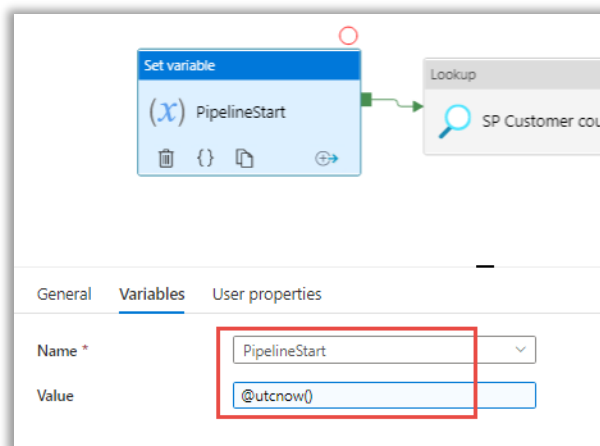
Set the Lookup activity to execute the stored procedure "[dbo].[ExtfactCustomerCounter]":

Update LS Insight Audit

Set the properties for *UpdateLSInsightAudit* and *OnError-UpdateLSInsightAudit* to write correct information in the LSInsightAudit table.
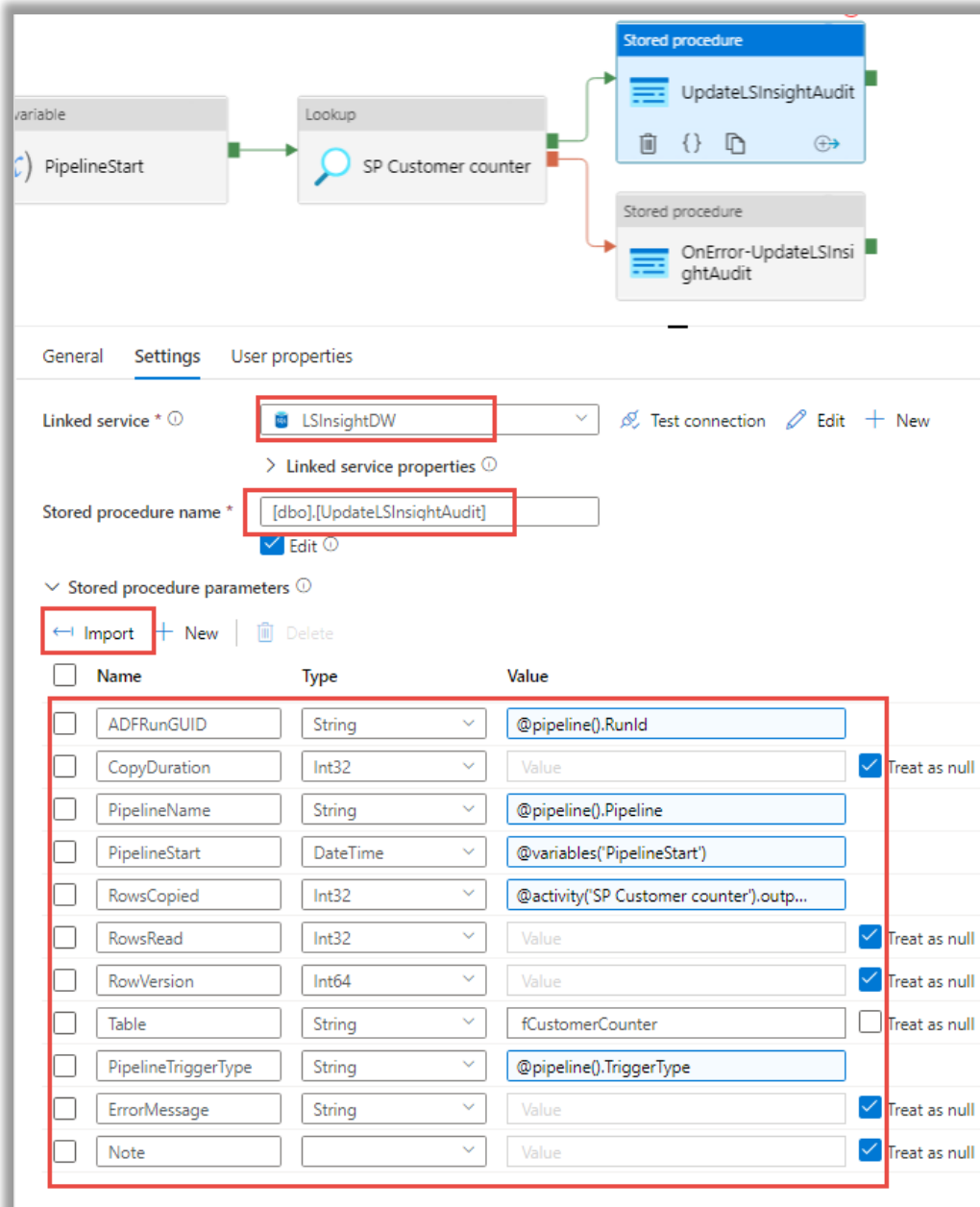
ADFRunGUID = @pipeline().RunId

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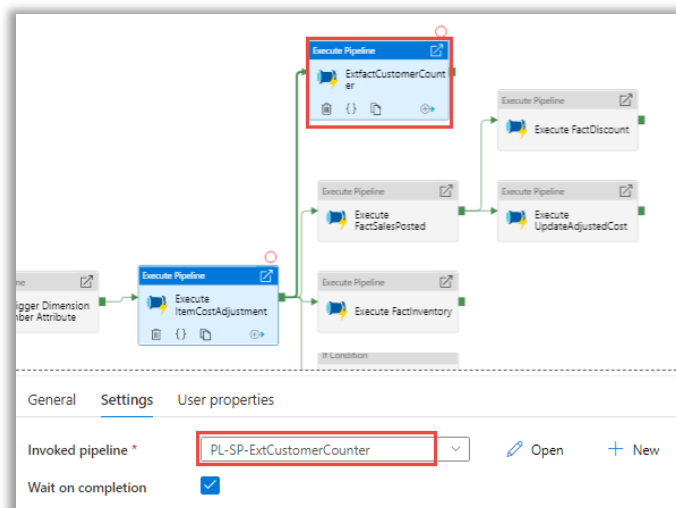
On your next scheduled run the new table will be available in the LS Insight DW and you can add to new data to your reports.

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

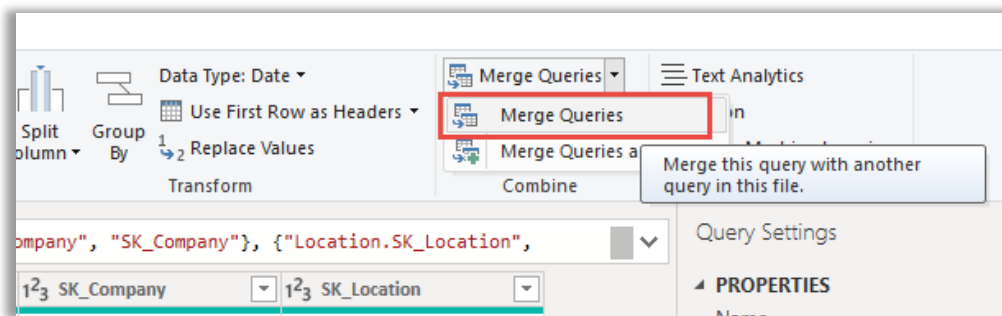In Power BI desktop, open *Home – Transform data – Transform data:*
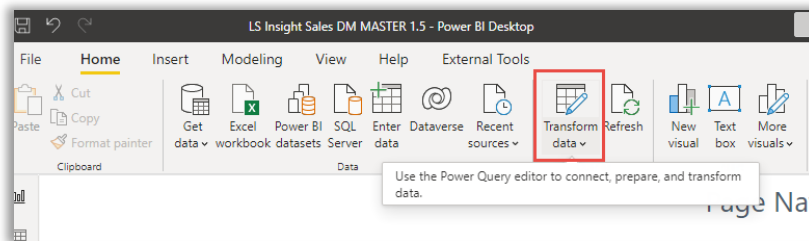
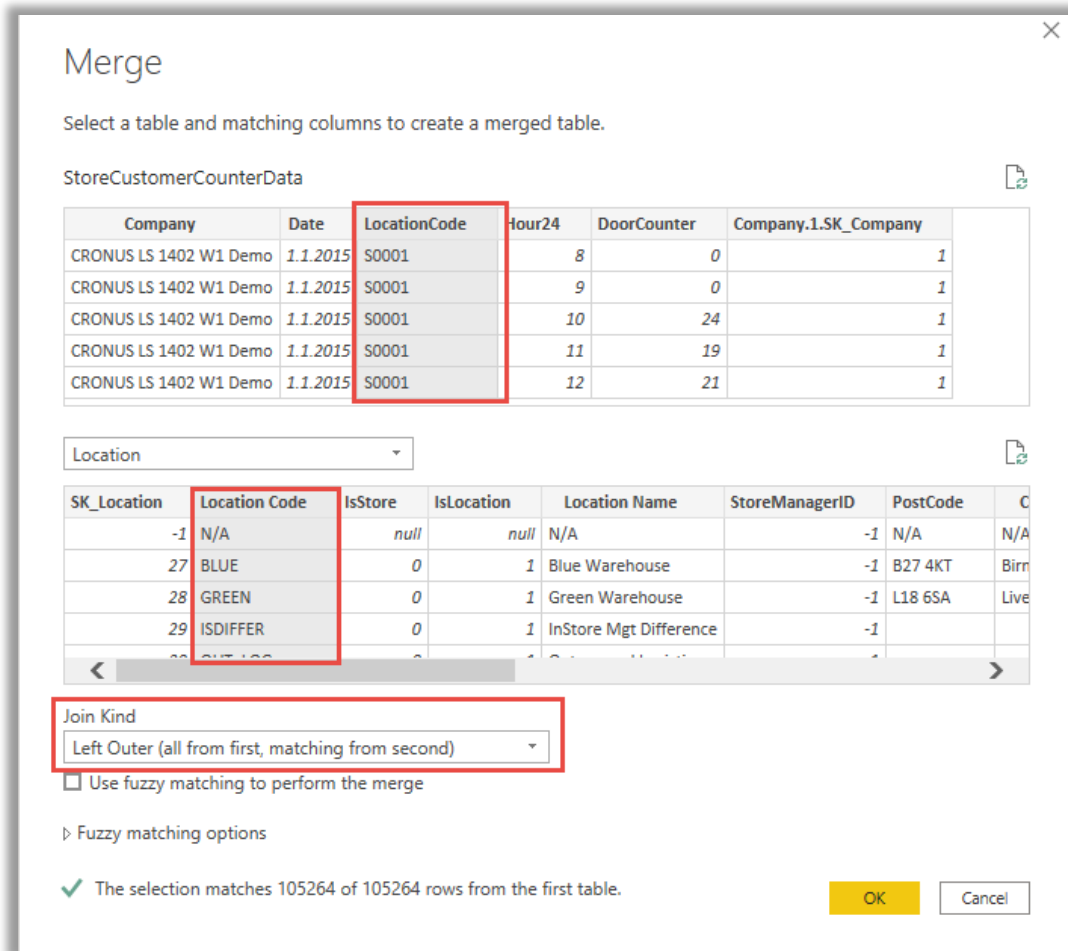Select *New Source* and in this case *Text/csv:*
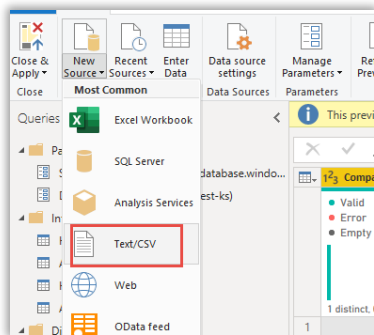
Select the source data and verify columns before loading to the data model.

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.
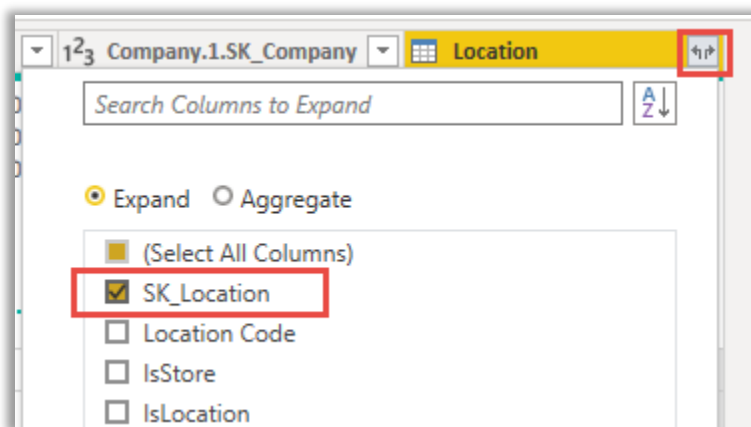
Select the keys that match and click "OK":

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.
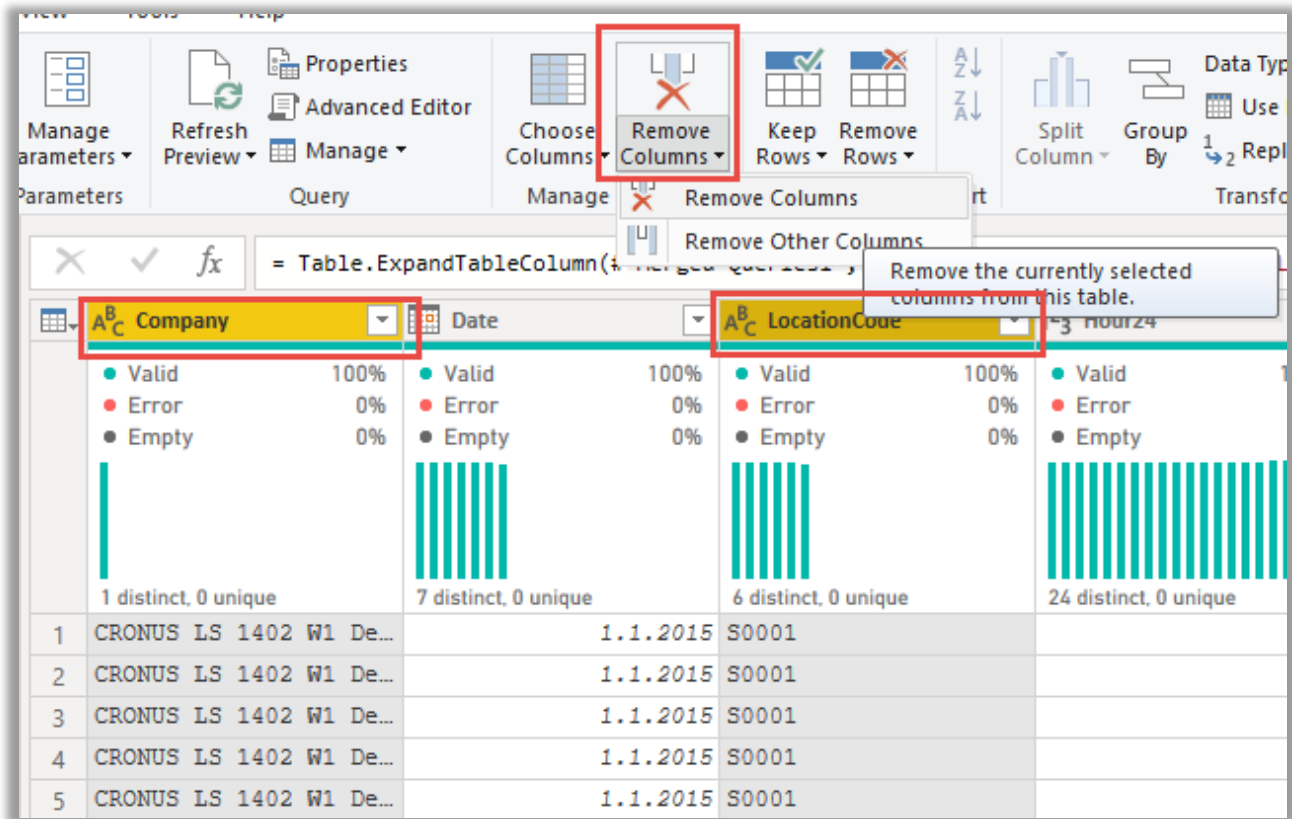
M query

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 LS Insight 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.
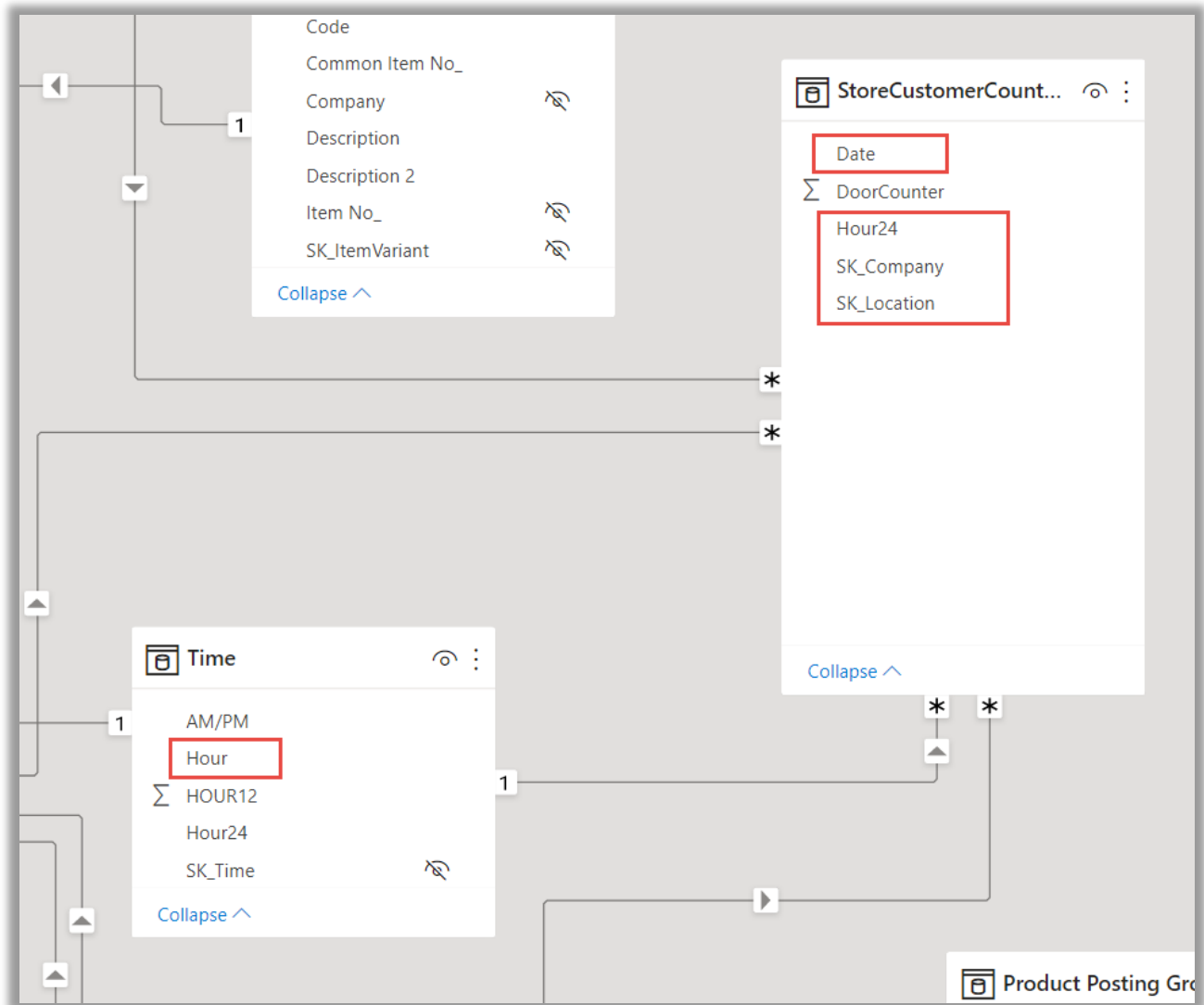
Use the new data in a report page

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: