

**ELT with Azure Data Factory**

**And**

**Mapping Data Flows**

Hands-on lab step-by-step

Feb 2020

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

© 2018 Microsoft Corporation. All rights reserved.

Microsoft and the trademarks listed at <https://www.microsoft.com/en-us/legal/intellectualproperty/Trademarks/Usage/General.aspx> are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

Contents

[Azure Data Factory hands-on lab 1](#_Toc39586604)

[ELT with Mapping Dataflows 2](#_Toc39586605)

[Task 1: Solution overview 2](#_Toc39586606)

[Task 2: Create Data warehouse tables for SmartFoods in Azure SQLDB 2](#_Toc39586607)

[Task 3: Introduction to Mapping Data Flows 6](#_Toc39586608)

[Slowly changing dimension type 2 withMapping dataflow (customerDim) 9](#_Toc39586609)

[Task 1: Create a new Mapping Dataflow and add source dataset 9](#_Toc39586610)

[Task 2: Add a parameter to Mapping Dataflow 13](#_Toc39586611)

[Task 3: Break the Name field to firstName and lastName fields 14](#_Toc39586612)

[Task 4: How to save (Publish) your Dataflow? 15](#_Toc39586613)

[Task 5: Remove extra columns and Rename columns using “Select” transformation 18](#_Toc39586614)

[Task 6: Calculate MD5 Hash of all non-key columns 19](#_Toc39586615)

[Task 7: Add DW table source 21](#_Toc39586616)

[Task 8: Compare staging records with DW records to identify updates and inserts 23](#_Toc39586617)

[Task 9: Identify Updates/Inserts using “Conditional Split” transformation 24](#_Toc39586618)

[Task 10: Handling New records (New stream) 25](#_Toc39586619)

[Task 11: Handling Changed records (Changed stream) 26](#_Toc39586620)

[Task 12: Putting together all inserts “New” and “Changed” together 30](#_Toc39586621)

[Task 13: Generate Surrogate Keys 31](#_Toc39586622)

[Task 14: Fix surrogate key value 32](#_Toc39586623)

[Task 15: Add Batch columns to our “Insert” dataset 32](#_Toc39586624)

[Task 16: Put Insert and Update records together 33](#_Toc39586625)

[Task 17: Prepare the dataset for sink (Alter row transformation) 34](#_Toc39586626)

[Task 18: Writing to destination DW table 36](#_Toc39586627)

[Task 19: Preview the final dataset (Debug) 37](#_Toc39586628)

[Task 20: Inspect the Dataflow “Script” 40](#_Toc39586629)

[Task 21: Building the pipeline for the dataflow 41](#_Toc39586630)

[Task 22: Debug the pipeline manually 43](#_Toc39586631)

[Task 23: Enhance the pipeline 44](#_Toc39586632)

[Task 24: (Challenge Task) Create an initial load pipeline for this DF 48](#_Toc39586633)

# Azure Data Factory hands-on lab

ELT with Mapping Dataflows

#### Solution overview

**Solution requirements:**

In previous exercise you ingested *customer*, *orders* and *orderlines* data from WWI to blob storage. Also, you ingested *Customer*, *Transactions* and *reference* data from SmartFoods systems to Blob storage. In this exercise you will use ADF Mapping DF to cleans, transform, enrich and store this data to be served using PowerBI to business users. Plus, data needs to be prepared for SmartFoods customer facing application to which displays accumulated loyalty points and comprehensive nutritional information and suggestions.

***Analytics Reporting:***

Since WWI business users are keen to setup a self-serve reporting environment, it means the serving layer storage solution should support the following requirements:

* Role based access control plus row and column level security so data can be made available to all users and controlled at group level which rows and columns will be made available to each user group.
* Dynamic Data Masking, certain PII information can be masked for certain user groups while they still have access to the rest of the data.

With the above requirements and considering this is only a POC, they decided to use *Azure SQL Database* for serving layer storage solution. The team acknowledges that after successful POC they will move this part of the solution to *Azure Synapse Analytics*.

***Data Science:***

In addition, the data science team decided to use *Azure ML services* to build ML/AI applications, particularly to support the nutritional suggestions based on the SmartFoods data. Hence, to avoid the need to export the cleansed data from SQL DB they requested the data to be stored in *Blob storage* as well if possible, at loading time. After considering all this requirement it was decided to use *Parquet files* on Azure Blob Storage (after POC to be replaced with Azure Data Lake Storage Gen2)

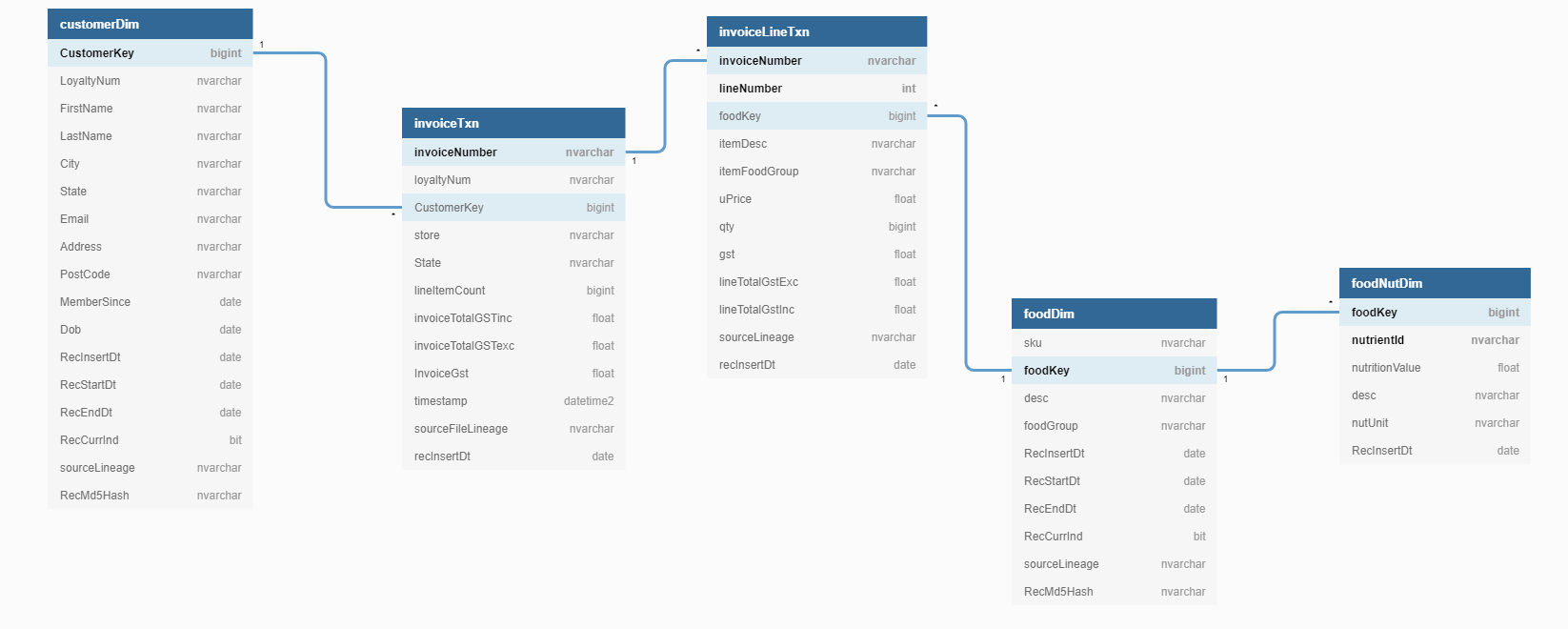
***SmartFoods App:***

Finally, the SmartFoods application (Web and Mobile) will need to access the data through an API and their primary requirements are performance, scalability and availability. After considering all this requirement it was decided to use *Azure CosmosDB* for application data storage.

In the Blob container we copied for SmartFoods there are multiple CSV files which represents SmartFood’s reference data for the transactions that comes through the HTTP API.

#### Create Data warehouse tables for SmartFoods in Azure SQLDB

Here is the initial star schema we are building for SmartFoods DW. Later we will also introduce some aggregate tables for easier reporting.



Either using Query Editor in Azure Portal or using SSMS connect to your Azure SQL DB and create a schema for SmartFoods DW and all the tables by running the following SQL script.

**Note:** You may need to add your Client IP Address to your SQL DB through “Set Firewall” page.

CREATE SCHEMA SmartFoodsDW;

GO

CREATE TABLE [SmartFoodsDW].[customerDim](

[CustomerKey] [bigint],

[LoyaltyNum] [nvarchar](max),

[FirstName] [nvarchar](max) NULL,

[LastName] [nvarchar](max) NULL,

[City] [nvarchar](max) NULL,

[State] [nvarchar](max) NULL,

[Email] [nvarchar](max) NULL,

[Address] [nvarchar](max) NULL,

[PostCode] [nvarchar](max) NULL,

[MemberSince] [date] NULL,

[Dob] [date] NULL,

[RecInsertDt] [date] NULL,

[RecStartDt] [date] NULL,

[RecEndDt] [date] NULL,

[RecCurrInd] [bit] NULL,

[sourceLineage] [nvarchar](max),

[RecMd5Hash] [nvarchar](max)

) ;

GO

CREATE TABLE [SmartFoodsDW].[foodDim](

[sku] [nvarchar](max),

[foodKey] [bigint],

[desc] [nvarchar](max) NULL,

[foodGroup] [nvarchar](max) NULL,

[RecInsertDt] [date] NULL,

[RecStartDt] [date] NULL,

[RecEndDt] [date] NULL,

[RecCurrInd] [bit] NULL,

[sourceLineage] [nvarchar](max),

[RecMd5Hash] [nvarchar](max)

) ;

GO

CREATE TABLE [SmartFoodsDW].[foodNutDim](

[foodKey] [bigint],

[nutrientId] [nvarchar](max),

[nutritionValue] [float] NULL,

[desc] [nvarchar](max) NULL,

[nutUnit] [nvarchar](60) NULL,

[RecInsertDt] [date] NULL

);

GO

CREATE TABLE [SmartFoodsDW].[invoiceLineTxn](

[invoiceNumber] [nvarchar](max),

[lineNumber] [int],

[foodKey] [bigint],

[itemDesc] [nvarchar](max) NULL,

[itemFoodGroup] [nvarchar](max) NULL,

[uPrice] [float],

[qty] [bigint],

[gst] [float],

[lineTotalGstExc] [float],

[lineTotalGstInc] [float],

[sourceLineage] [nvarchar](max),

[recInsertDt] [date]

);

GO

CREATE TABLE [SmartFoodsDW].[invoiceTxn](

[invoiceNumber] [nvarchar](max),

[loyaltyNum] [nvarchar](max) NULL,

[CustomerKey] [bigint] NULL,

[store] [nvarchar](max) NULL,

[State] [nvarchar](max) NULL,

[lineItemCount] [bigint],

[invoiceTotalGSTinc] [float],

[invoiceTotalGSTexc] [float],

[InvoiceGst] [float],

[timestamp] [datetime2](7),

[sourceFileLineage] [nvarchar](max),

[recInsertDt] [date]

) ;

GO

#### Introduction to Mapping Data Flows

Mapping Data Flows is a new feature of Azure Data Factory that allows you to build data transformations in a visual user interface (code-free or very low amount coding).

Mapping data flows are visually designed data transformations in Azure Data Factory. Data flows allow data engineers to develop graphical data transformation logic without writing code. The resulting data flows are executed as activities within Azure Data Factory pipelines that use scaled-out Apache Spark clusters. Data flow activities can be engaged via existing Data Factory scheduling, control, flow, and monitoring capabilities. More info in this [article](https://docs.microsoft.com/en-us/azure/data-factory/concepts-data-flow-overview).

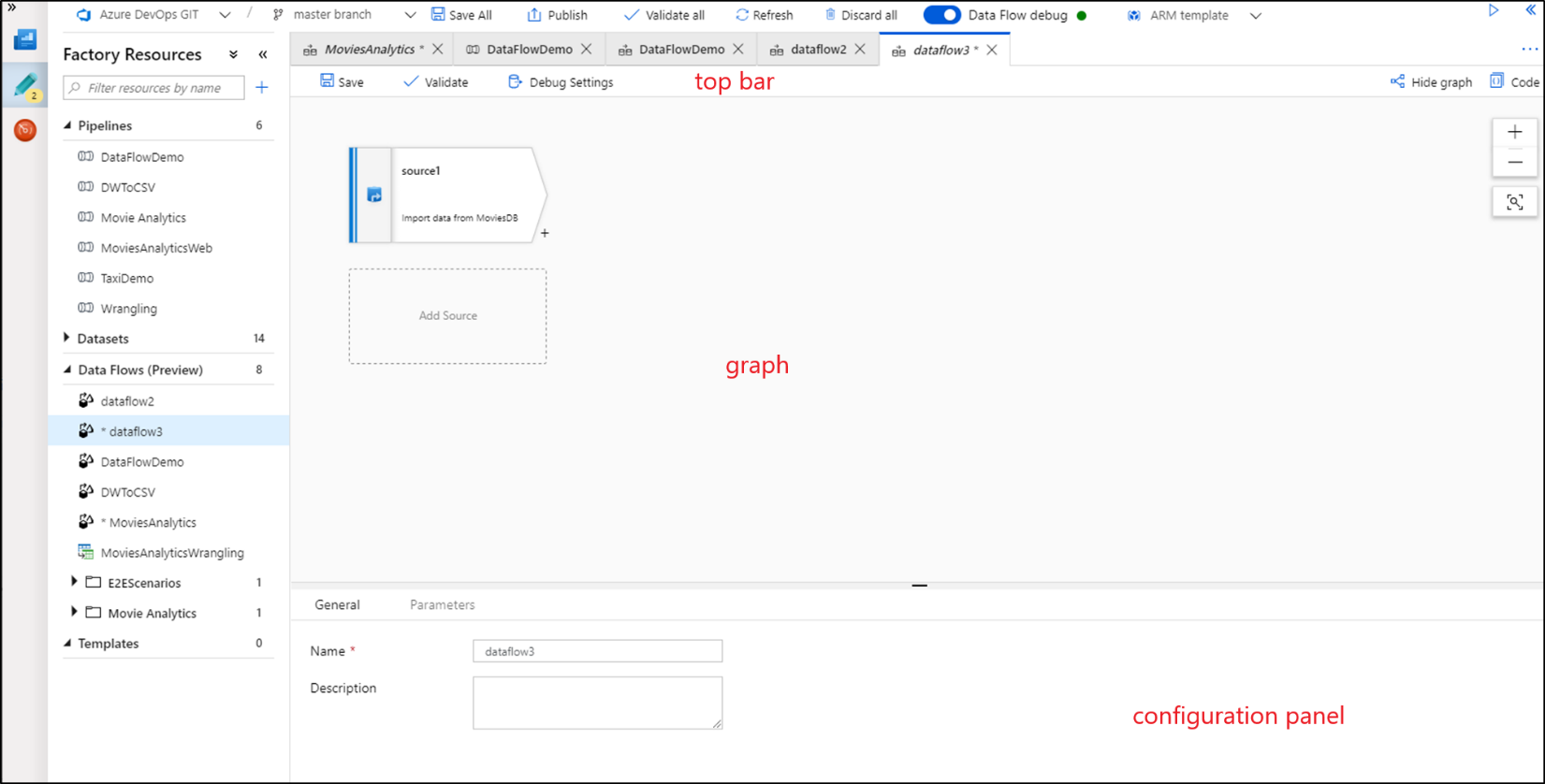
Mapping data flows provide an entirely visual experience with no coding required. Your data flows run on your execution cluster for scaled-out data processing. Azure Data Factory handles all the code translation, path optimization, and execution of your data flow jobs.

ADF translates the flow built in the visual interface to Apache Spark code which will run on serverless Spark cluster than we can configure in terms of count and type of worker nodes.

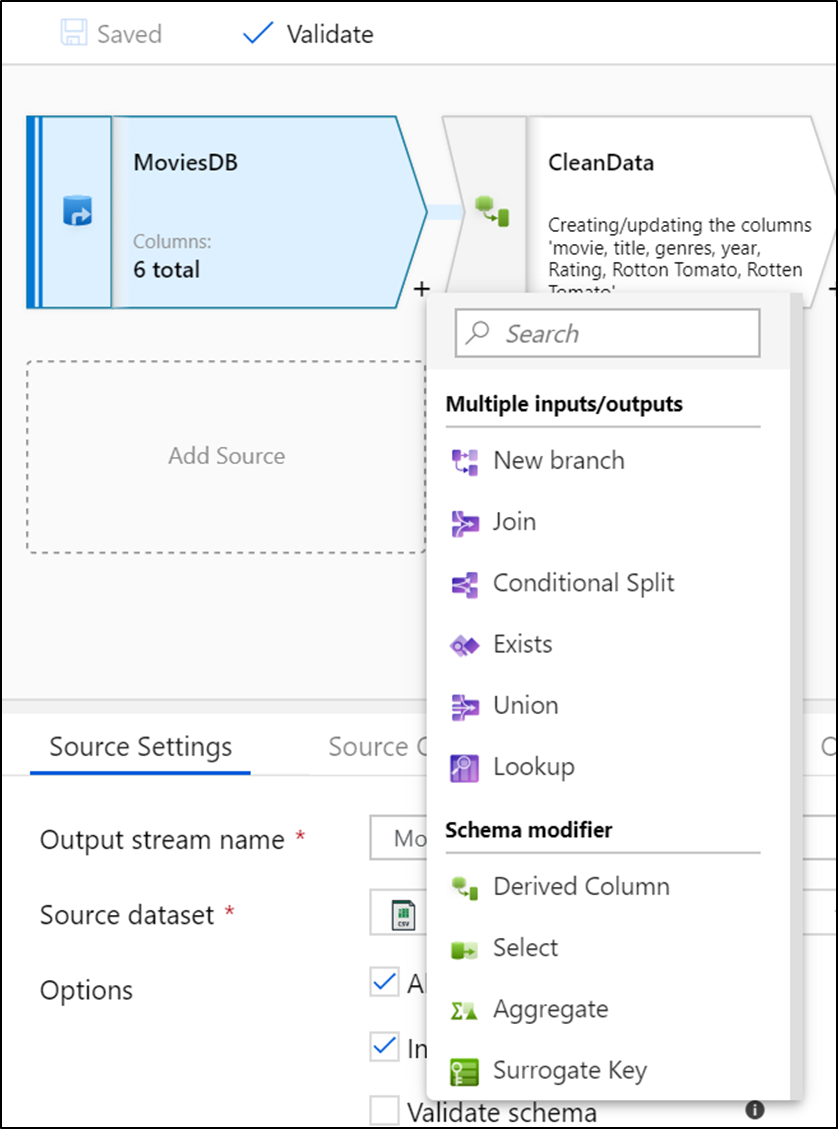
**Serverless Spark cluster**: The Apache Spark cluster will be deployed on Azure Integration Runtime and like Azure IR, which is serverless, the cluster is fully managed by Azure and charged per number of seconds the job takes to run.

**Mapping DF on SH-IR?** ADF Spark clusters are only deployable on Azure and currently there is no option for deploying on-prem.

**Data flow canvas:** Here is what the Data flow canvas look like. It is separated to three parts



**The Graph:** The graph displays the transformation stream. It shows the lineage of source data as it flows into one or more sinks. To add a new source, select Add source. To add a new transformation, select the plus sign on the lower right of an existing transformation.



Slowly changing dimension type 2 withMapping dataflow (customerDim)

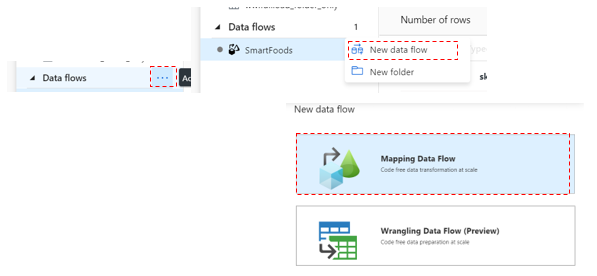
#### Create a new Mapping Dataflow and add source dataset

You loaded SmartFoods’ customer staging data from API to Blob storage in CSV format in the following location “smartfoodsstaging/customer/smartfoods\_customers\_<date>.csv”

The ultimate table will look like below:

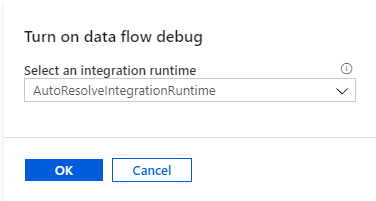
|  |  |  |
| --- | --- | --- |
| Field | Description | Source |
| CustomerKey bigint – primary key | Surrogate Key | Generated by ELT |
| LoyaltyNum nvarchar | Source Key | Existing field in source |
| FirstName nvarchar | From Name field | Generated by ELT |
| LastName nvarchar | From Name field | Generated by ELT |
| City nvarchar |  | Existing field in source |
| State nvarchar |  | Existing field in source |
| Email nvarchar |  | Existing field in source |
| Address nvarchar |  | Existing field in source |
| PostCode nvarchar |  | Existing field in source |
| MemberSince date |  | Existing field in source |
| Dob date |  | Existing field in source |
| RecInsertDt date | Actual ELT running date | Generated by ELT |
| RecStartDt date | Record validity start date = batch date | Generated by ELT |
| RecEndDt date | Record validity end date = batch date | Generated by ELT |
| RecCurrInd Boolean | Record validity indicator | Generated by ELT |
| sourceLineage nvarchar | Name of source file | Generated by ELT |
| RecMd5Hash nvarchar | MD5 Hash of all source fields except natural key | Generated by ELT |

1. Create a mapping Dataflow by clicking on new Data flow button and rename it to “SmartFoodsCustomerELT”



1. At the top of the page turn on the “data flow debug” -> select *AutoResolveIntegrationRuntime*

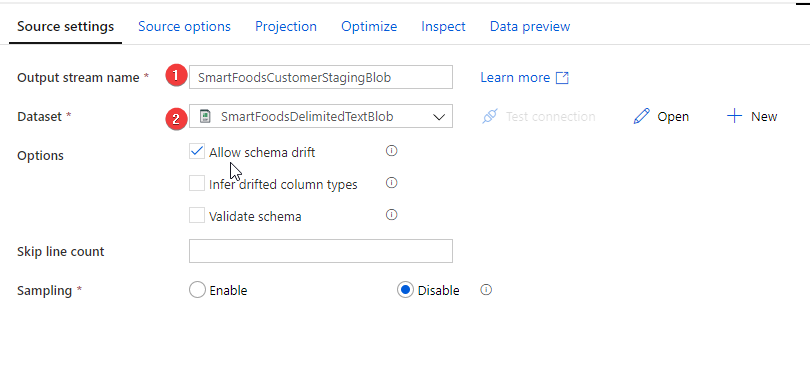




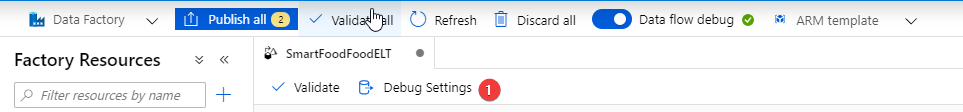
**Debug mode:** Azure Data Factory mapping data flow's debug mode allows you to interactively watch the data shape transform while you build and debug your data flows. The debug session can be used both in Data Flow design sessions as well as during pipeline debug execution of data flows. To turn on debug mode, use the "Data Flow Debug" button at the top of the design surface. For more information check [this](https://docs.microsoft.com/en-us/azure/data-factory/concepts-data-flow-debug-mode) article.

**Note:** By turning on Debug mode, ADF deploys a Spark cluster on the same region as your Integration Runtime.

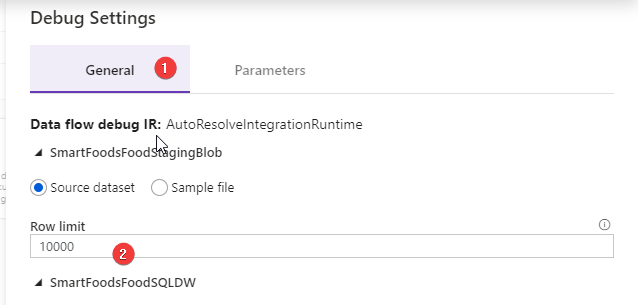
1. Click “Add Source” on canvas
2. Change the output stream name to: “SmartFoodsCustomerStagingBlob”
3. Select the ‘*SmartFoodsDelimitedTextBlob’*



1. Click “Debug Settings” on the top task bar

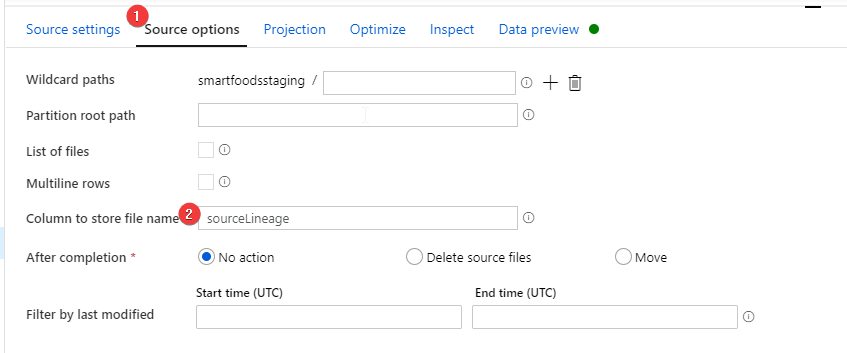


1. Under “General” increase “*Row limit”* to 10,000



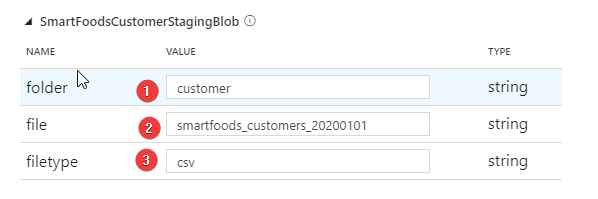
**Row Limit:** This is the maximum number of rows that will be retrieved from the source data set in when we try to preview the transformation results. If you are working with multiple datasets that needs to be joined, it is best to increase this to a higher limit, otherwise the join result in “preview” will have a lot of missing records.

1. Under source options change “Column to store file name” to “sourceLineage”



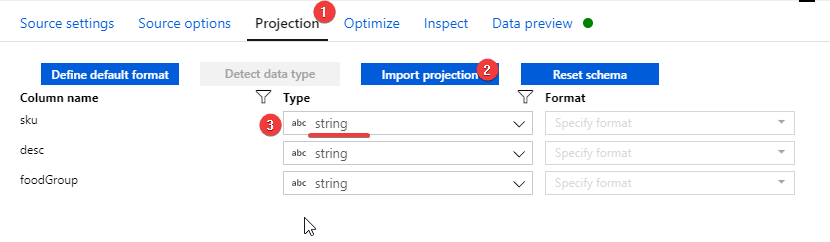
**Note:** This enables the option to take the processing file name and pass it on as a column for every row in the dataset. This is really valuable information for testing, lineage and debugging.

1. Under parameters provide
   * “folder” = customer
   * “file” = smartfoods\_customers\_20200101
   * “fileType” = csv



**Where did the parameters came from?** We are reusing the parametrized dataset created previously for importing data from source systems to blob storage. These are the parameters that dataset needs to operate.

1. Go to “projection tab”
   1. click “*import Projection*”
   2. Change the data type for “Postcode” column from “short” to “string”



**Note:** When we load data from a delimited text or flat file, it is always recommended to double check the schema ADF detected and if needed fix it.

**Why changing SKU from Short to String?** 1. The destination sink (Azure SQLDB) does not recognized values of type short. 2. Postcodes can have leading 0 in it which gets eliminated in non-string type fields

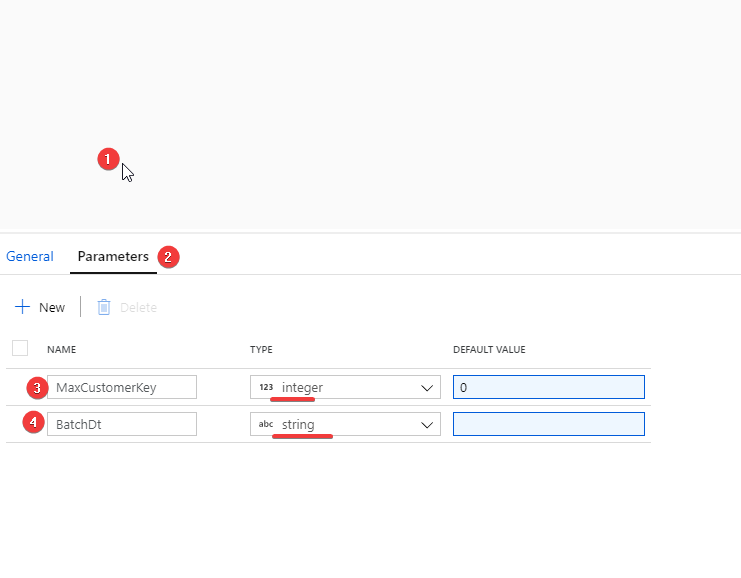
1. Now go to “Data Preview” tab and refresh it to get a preview of the dataset

Debug Cluster: We need “Debug” mode running for 1. Importing data projection (schema) 2. Running preview task.

#### Add a parameter to Mapping Dataflow

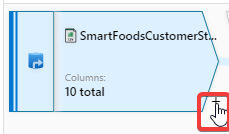
Like ADF pipelines and datasets, Mapping dataflows also provide the option to create parameters. Dataflow parameters can be used within the dataflow itself (more on this later) and can be filled in by an ADF pipeline (more on this later). For now, let’s create two parameters.

1. Click on any white space of the DF canvas
2. Click on parameters tab
3. And create two parameters
   1. MaxCustomerKey -- integer
   2. BatchDt -- string



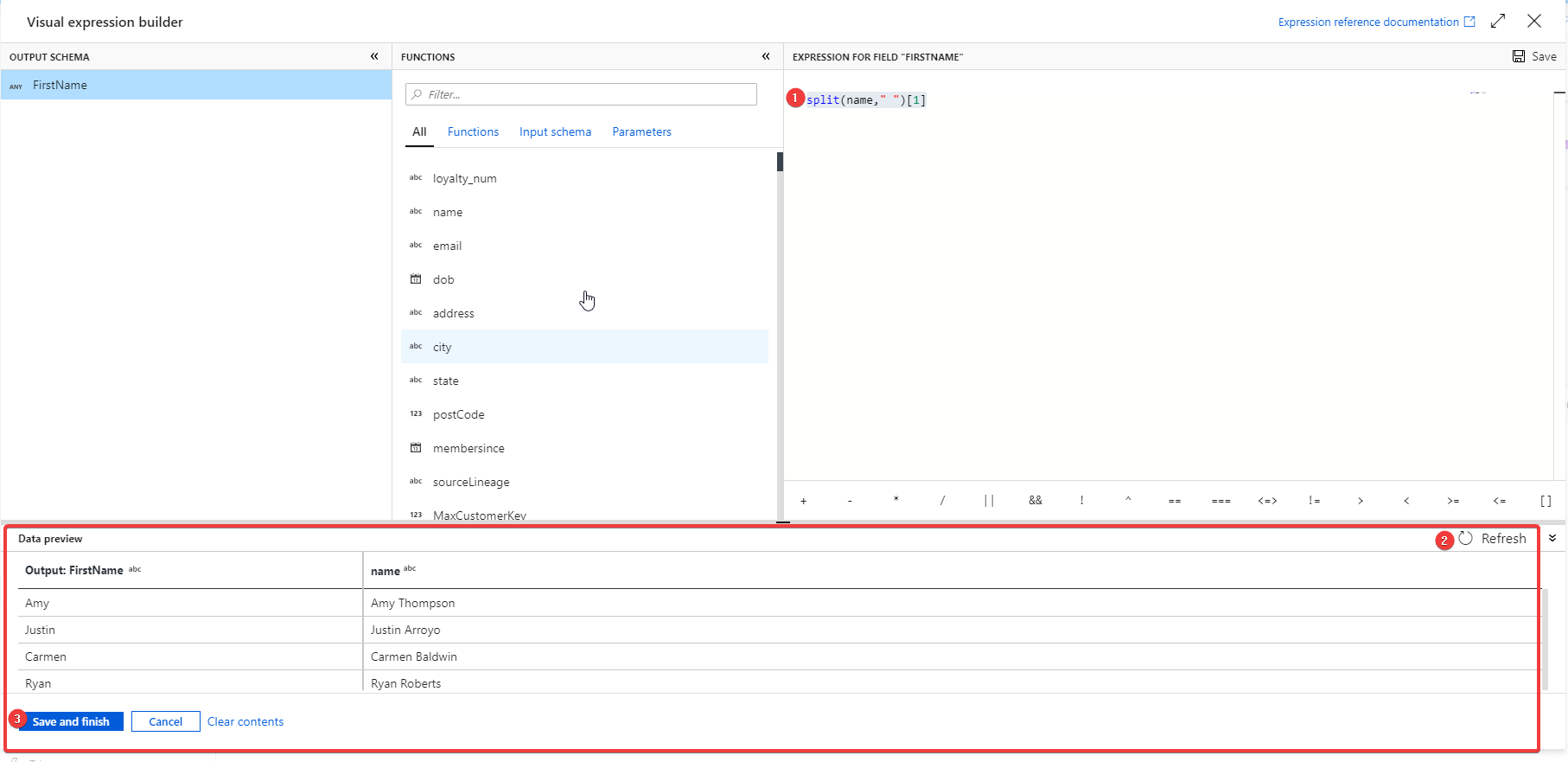
#### Break the Name field to firstName and lastName fields

1. Click the plus sign on the lower right-hand side of source transformation to add the next transformation.



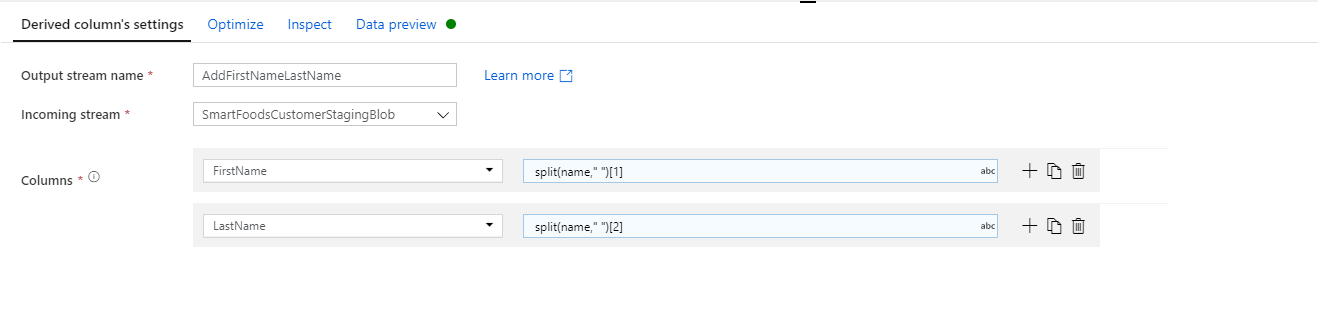
1. Select “Derived column” transformation
2. Rename it to “AddFirstNameLastName”
3. Add a new column and for name type “FirstName” for value click on the right box and it opens the expression editor and enter following expression.
4. Click refresh to see the result of the expression on the data.

split(name," ")[1]



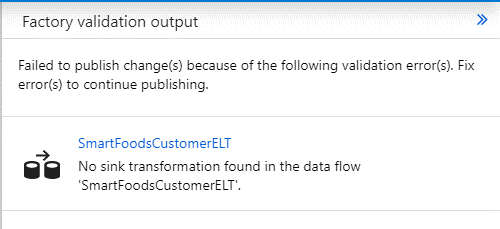
1. Create another column for “LastName” using below expression

split(name," ")[2]



#### How to save (Publish) your Dataflow?

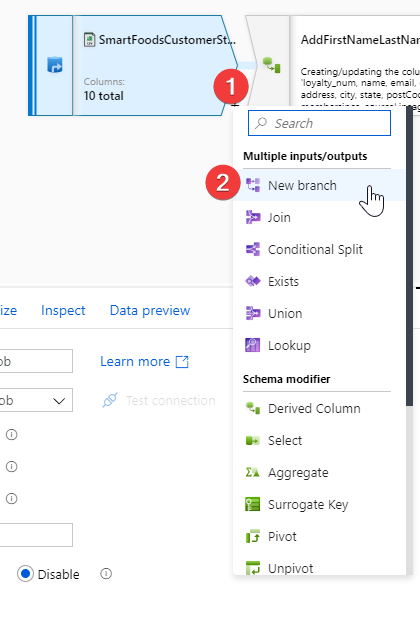
At this stage if you try validating or publishing a dataflow you will get an error like below



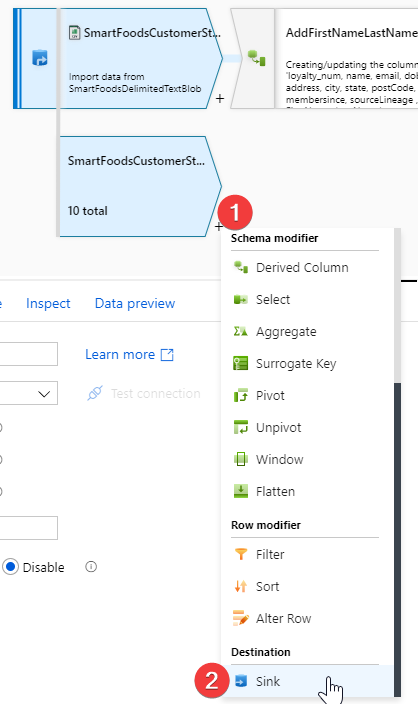
ADF mapping Dataflows does not allow publishing a DF that is still not complete, and every dataflow must have at least a “Sink” transformation to be considered complete.

To get around this issue and be able to save your work progress you can add a dummy “Sink” transformation to the end of you flow, publish your work and then remove it to continue building the flow until you reach a point that you add the actual “Sink” transform.

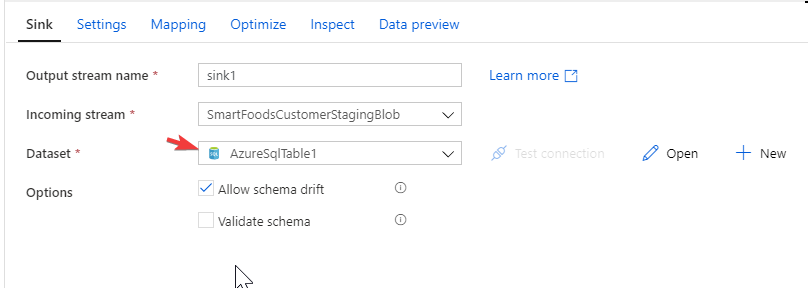
1. After the “Source” Transform add a “New branch” transform



1. Add a “sink” transform after the “New branch”



1. Set up the sink as below:

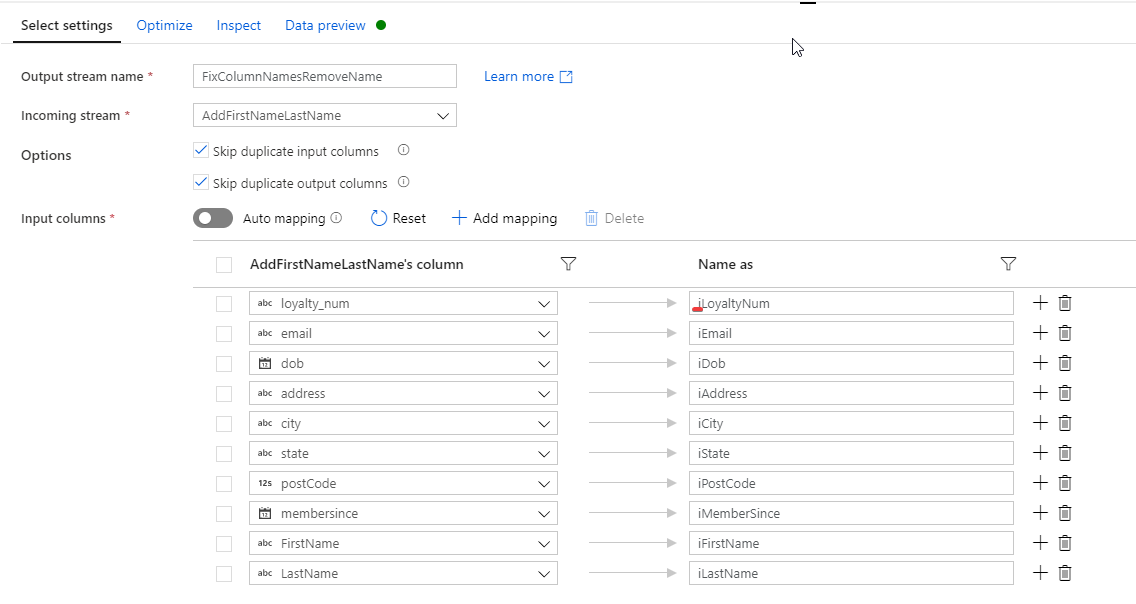


**Note1:** As this AzureSqlTable1 dataset is parameterized, it will add two new parameters to debug settings. Just fill those prams with any random string (it will not affect your dataflow)

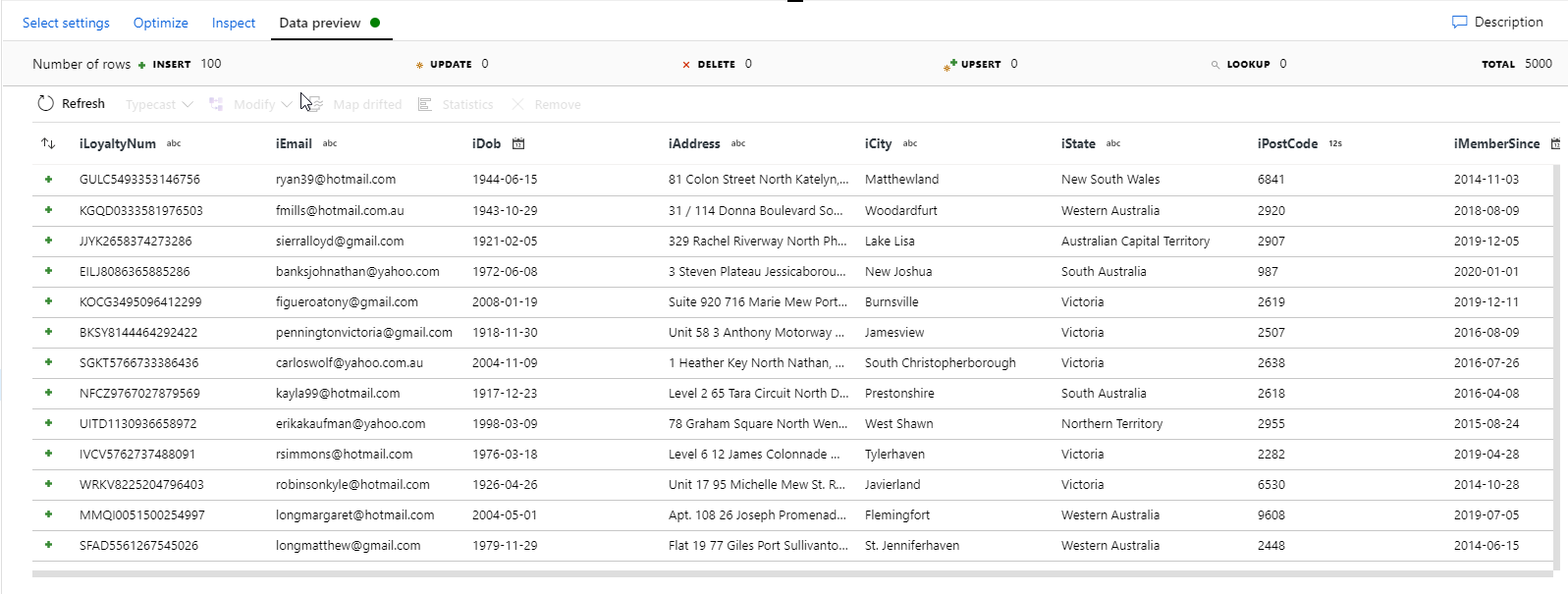
**Note2: DO NOT forget to Remove the “New Branch” and dummy “Sink” transforms once you have the final actual “Sink” transform.**

#### Remove extra columns and Rename columns using “Select” transformation

1. Add a “Select” transformation after the previous transformation to 1. Remove the “name” column in favor or “firstName” and “LastName”. 2. We are going to compare this data with existing data in the dimension to identify changes an newly added rows so for easier identification of columns we add an ‘i’ in front of all columns.
2. Rename your “Select” transformation to “FixColumnNamesRemoveName”
3. Set it up as below screenshot



1. Preview the output of this transformation



#### Calculate MD5 Hash of all non-key columns

**Note:** If you are not familiar with Slowly Changing Dimensions in Data Warehousing read this article in Wikipedia <https://en.wikipedia.org/wiki/Slowly_changing_dimension>

We are trying to build a SCD type 2 for customers table. This means no data will be discarded from the DW table. If a record changes in the OLTP source system this change gets captured by adding a new record with updated values and marking the old record as inactive(usually referred to as closing record)(recCurrInd), plus adding the date/timestamp of the record closure (recEndDate).

There are multiple technics to identify updated records, one is to compare the MD5 Hash of the existing records against the newly received record from source and if they do not match it is considered a change.

1. Add a ‘Derived column’ transformation to calculate MD5 Hash of all non-key columns
2. Rename it to “MD5Hash”
3. Add a column “iRecMd5Hash”
4. For expression use

md5( iif(isNull(iEmail),'',toString(iEmail))+

iif(isNull(iDob),'',toString(iDob))+

iif(isNull(iAddress),'',toString(iAddress))+

iif(isNull(iCity),'',toString(iCity))+

iif(isNull(iState),'',toString(iState))+

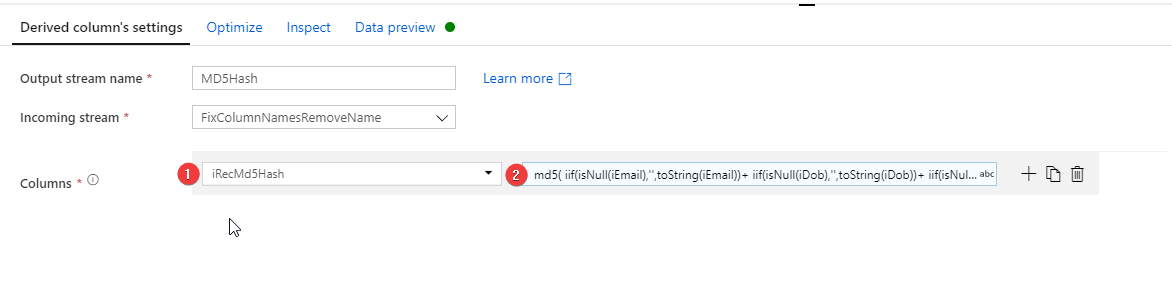
iif(isNull(iPostCode),'',toString(iPostCode))+

iif(isNull(iMemberSince),'',toString(iMemberSince))+

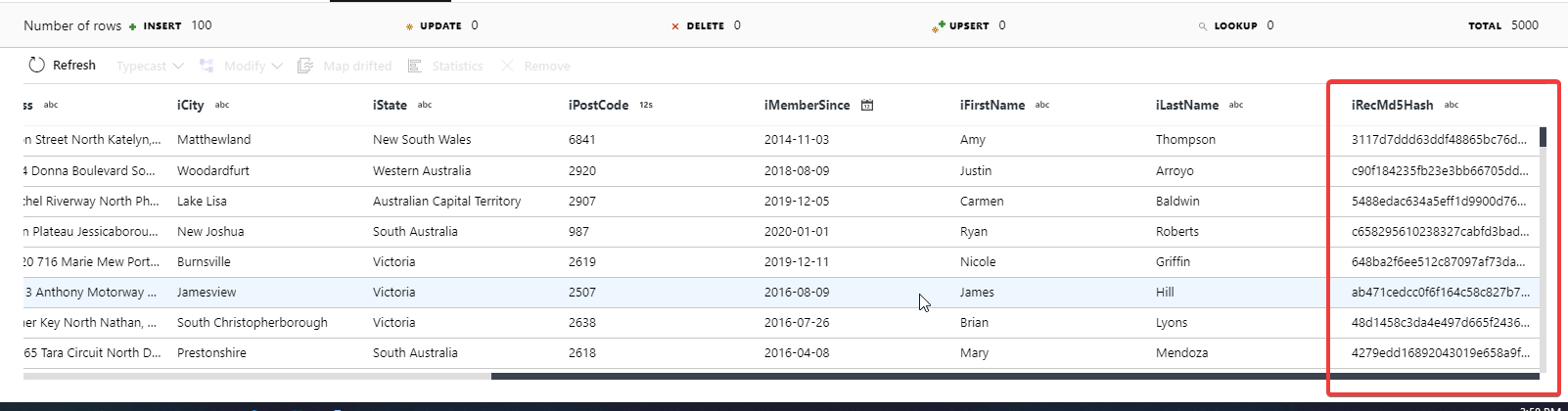
iif(isNull(iFirstName),'',toString(iFirstName))+

iif(isNull(iLastName),'',toString(iLastName)))

**Expression explanation:** For every non-key column first, we replace all Nulls with an empty string, then convert all fields to string and concatenate them together. Finally use the md5 method to calculate the hash of the whole concatenated string.



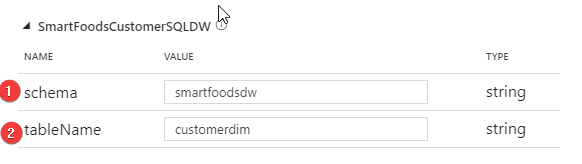
1. Preview the output



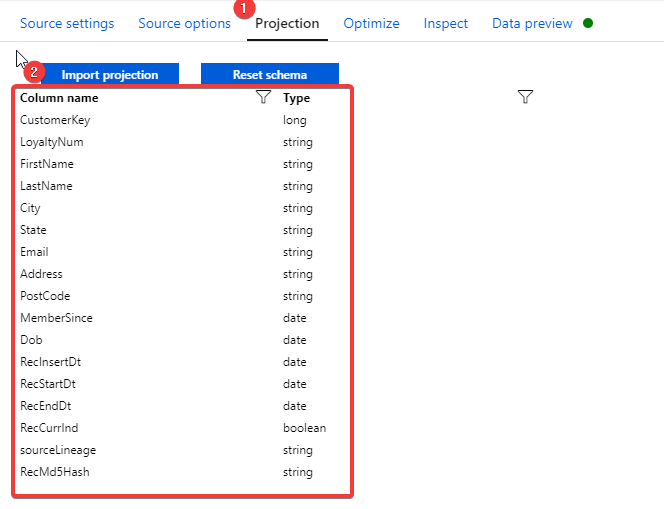
#### Add DW table source

To perform the comparison and identify updated or new records we need to also retrieve the existing data from DW dimension table (at first run the table is empty, hence every row will be determined as new).

1. On the far-left hand side of the canvas under the first source click “Add source” to add a new source to the flow.
2. Rename it to “SmartFoodsCustomerSQLDW”
3. For dataset Select “AzureSqlTable1” (You created this SQL Dataset at the beginning of the lab)
4. Go to “Debug Settings” and provide the below parameters:



1. Go to Projection tab and import the dataset projection

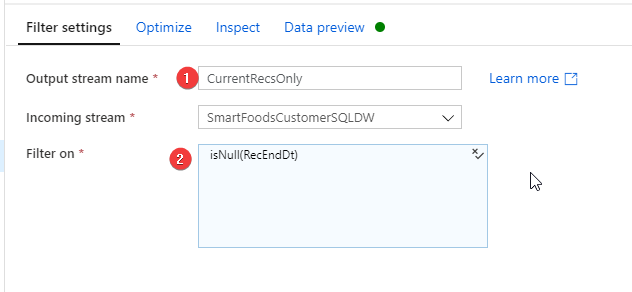


1. Add a filter transformation after the DW source

**Note:** The filter transformation is used to only get ‘active’ rows (RecEndDt is Null) from the table.

* 1. Rename it to “CurrentRecordsOnly”
  2. Filter on:

isNull(RecEndDt)



#### Compare staging records with DW records to identify updates and inserts

Next step is to compare the records from staging and DW. We use a “join” transformation for this purpose. As the staging dataset will have records that do not exists in DW dataset we need to use an “left outer join” the sudo code for the join is:

SELECT \* FROM staging s

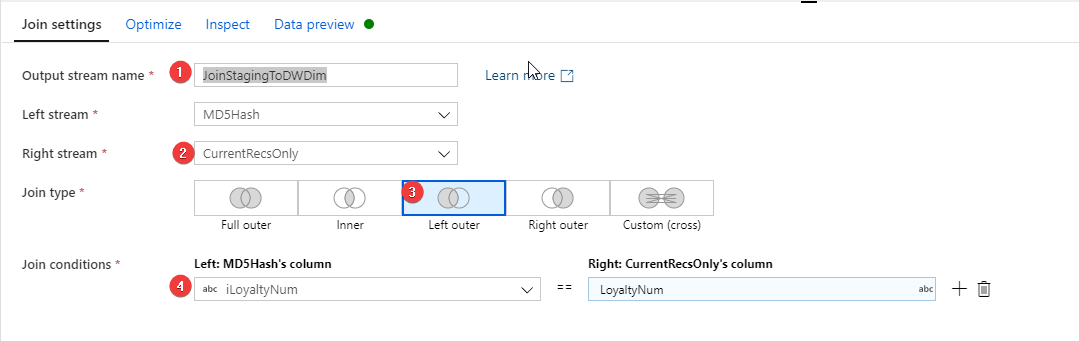
LEFT OUTER JOIN

dw\_table d

ON

s. key = d.key

1. Add a join transformation after “MD5Hash” transformation
2. Rename it to “JoinStagingToDWDim”
3. For right stream select “CurrRecsOnly”
4. Join type: “Left outer”
5. Join conditions: iLoyaltyNum == LoyaltyNum



**Note:** Since we renamed the staging columns with an ‘i’ in front of them it is quite easy to find the right column for joins here.

#### Identify Updates/Inserts using “Conditional Split” transformation

A “Conditional Split” transformation allows us to split an incoming dataset into multiple outgoing datasets based on some logical criteria.

Here we need to find out

* If a record is new (if the right-hand side [records from DW] of left outer join is null)
* If a record has Changed (if the primary key existed within both staging and DW datasets but MD5hashes are not matching).
* If a record has Not Changed (if the primary key existed within both staging and DW datasets and MD5hashes are matching).

1. Add a “Conditional split” transformation after “JoinStagingToDWDim” transformation.
2. Rename it to “SDC2Split”
3. For Split on option set it to “First matching condition”

**Split On**: If we set this to “First matching condition” the first condition a record fits with will be pushed to that stream and condition(s) after that will not be tested on the record. First. This option is more efficient in processing but has two implications: 1. The order of conditions becomes important (stricter conditions should be placed above less strict ones) 2. Every record only gets passed into a single stream. If your workflow logic requires input records to get passed into multiple output streams choose “All matching conditions”

1. Split Condition

* New:

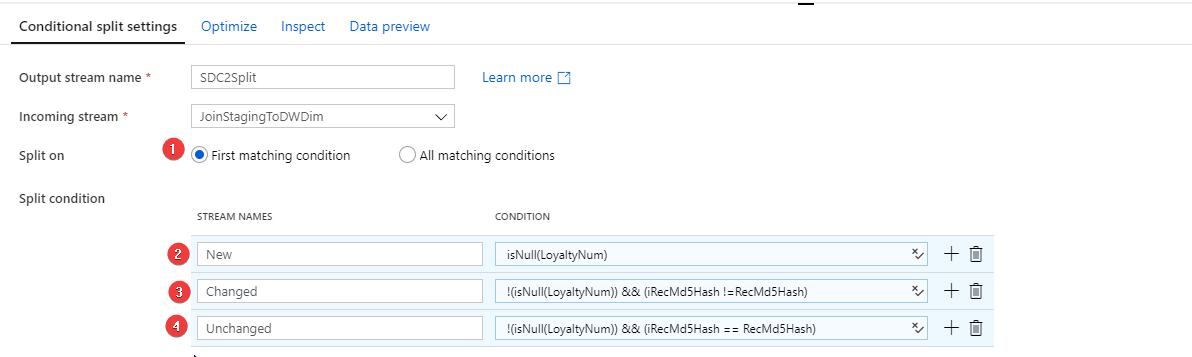
isNull(LoyaltyNum)

* Changed:

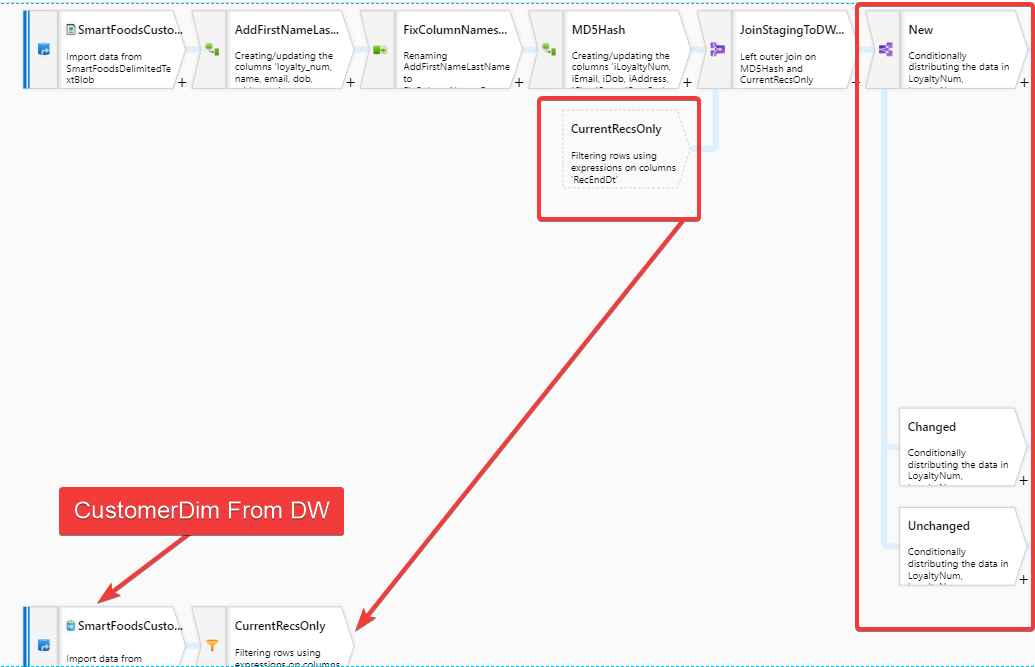
!(isNull(LoyaltyNum)) && (iRecMd5Hash !=RecMd5Hash)

* Unchanged:

!(isNull(LoyaltyNum)) && (iRecMd5Hash == RecMd5Hash)



So far this is how your dataflow should look like. (Conditional split added 3 streams to our flow which we need to manipulate the output from – We will refer to these transformation as New, Changed and Unchanged)



#### Handling New records (New stream)

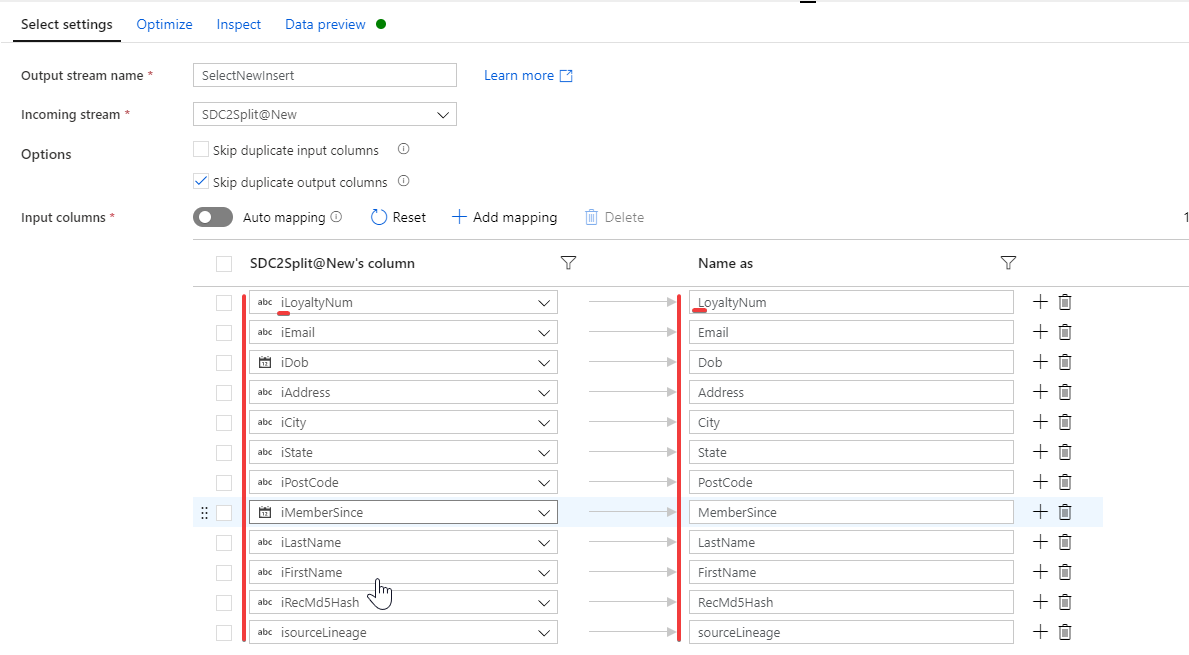
First stream out of condition split is “New”. For every new records we need to only insert a new record to the table.

1. Add a “Select” transformation after “New” stream.

**Note:** This select is going to perform two tasks: 1. Select the columns coming from Staging and remove all other columns (Join transform added column from both staging and DW to our dataset). 2. Remove the extra ‘i’ we added to the front of the staging columns to identify them easier.

1. Rename it to “*SelectNewInsert*”
2. Use the below screenshot as guide on setting the “Select” transformation.

**Note:** Total Number of columns selected is **12.** Only columns with a leading ‘i’ will be selected and the output name will not have a leading ‘i’

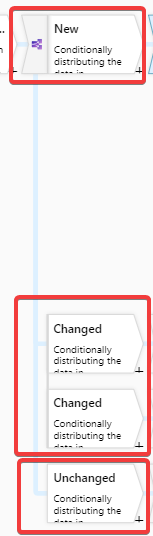


#### Handling Changed records (Changed stream)

Second stream out of condition split is “Changed”. For Changed records we need to insert a new record and update an existing record. So, we need to add a “**New branch”** transformation after “Changed”.

**New Branch:** allows us to replicate the output of a transformation to two streams. Essentially a “New branch” is not a transformation but only duplicating the output of a stream.

This is how our flow will look like:

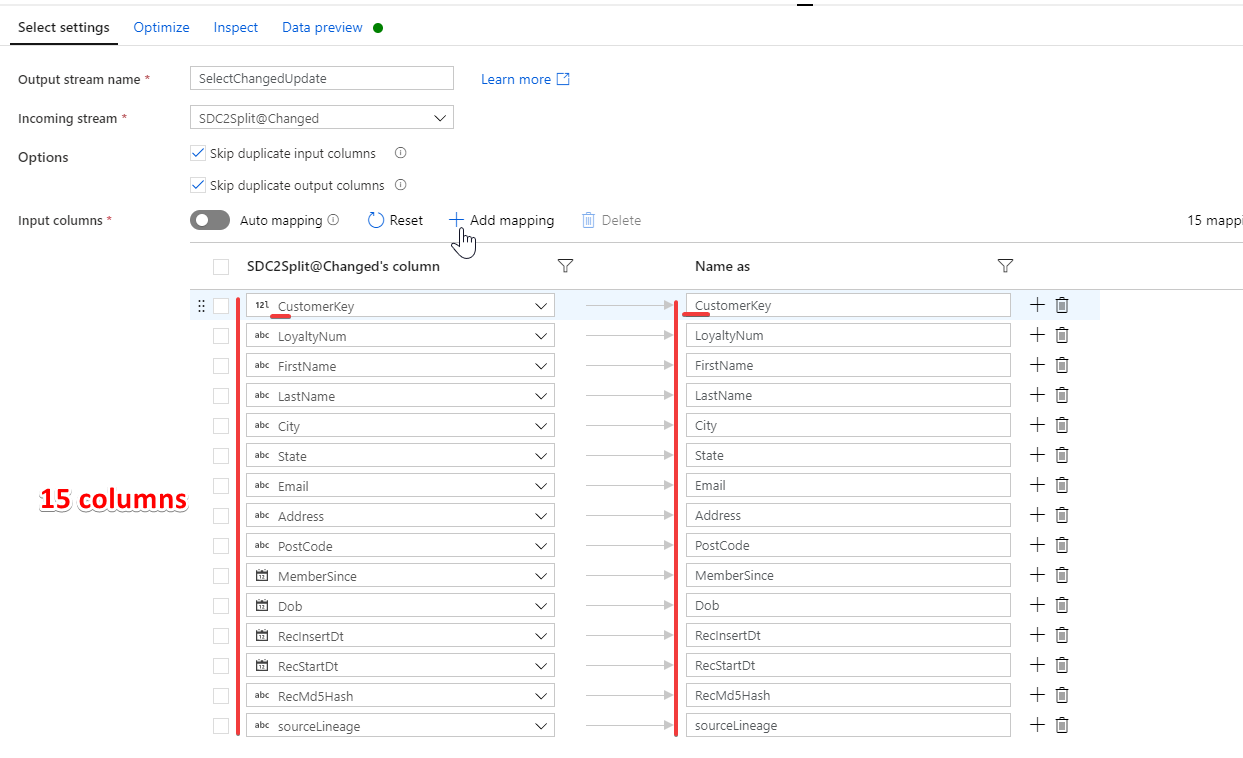


**Updating existing changed records (Closing old records)**

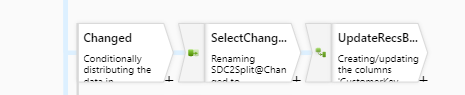
1. After the first “Changed” stream add a “Select” transform and rename it to “SelectChangedUpdate”

This stream is going to perform updated for records.

1. We will select a total of 14 columns in this select as below. Here we select the columns WITHOUT the leading ‘i’ as we want the column values from DW. The only columns we are updating on these records are 1. RecEndDate 2. RecCurrInd



1. After this add a “Derived Column” Transform to finally add the two columns we are updating.
2. Rename it to “*UpdateRecsBatchColumns*”



1. Add “RecEndDt” column:

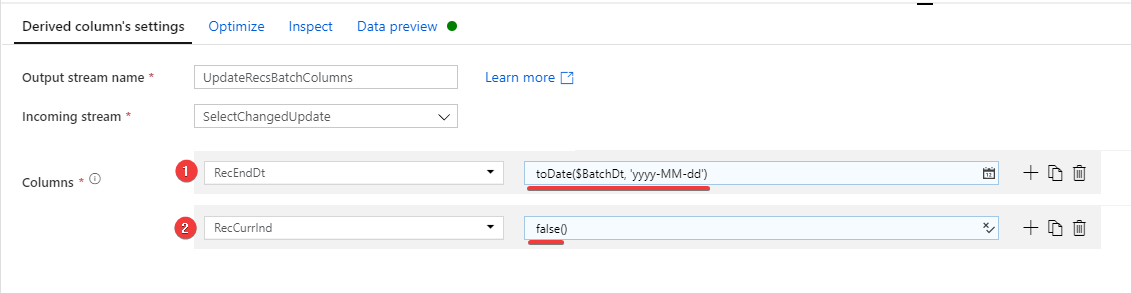
toDate($BatchDt, 'yyyy-MM-dd')

**Expression explanation**: Here we are using the “BatchDt” parameter (In Dataflow expression a ‘$’ is used to plant a parameter in an expression). You created this pram in the beginning of the exercise. Also the parameter is of type ‘String’ so we pass it to ‘toDate’ function to transform it to a date

**Why $BatchDt and not just use ‘currentDate()’ function instead?** The answer is there is no guarantee the batch always runs on the same date that the record change has happened. Specially on initial loading of the data, we would process multiple days on the same day to catchup with the current date.

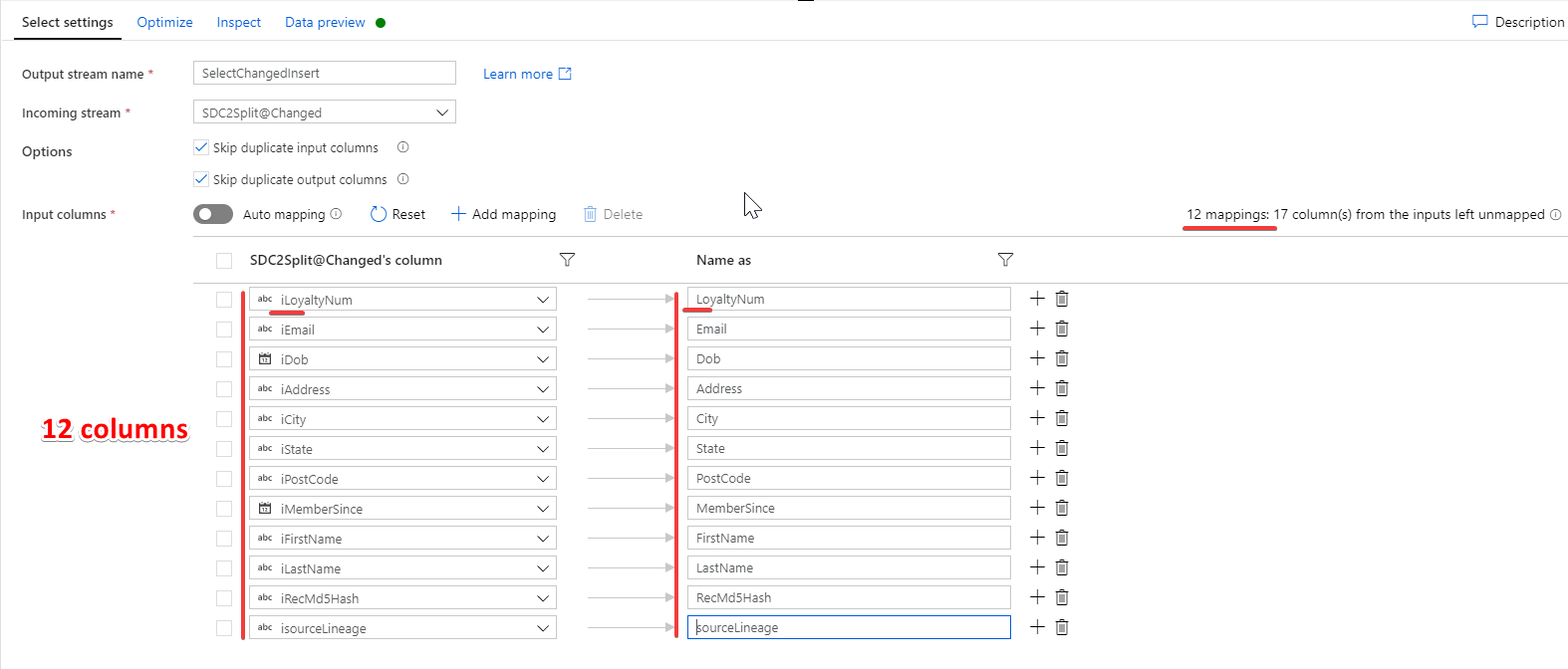
1. Add RecCurrInd column:

false()

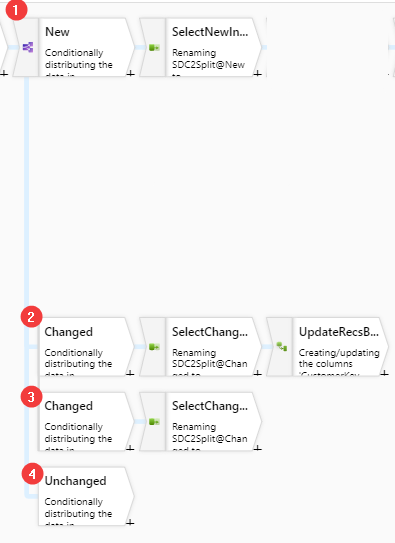


**Adding new version of the changed records**

1. After the second “Changed” stream add a “Select” transform and rename it to “*SelectChangedInsert*”
2. We will select a total of 12 columns in this select as below. Here we select the columns WITH the leading ‘i’ as we want the column values from staging source.



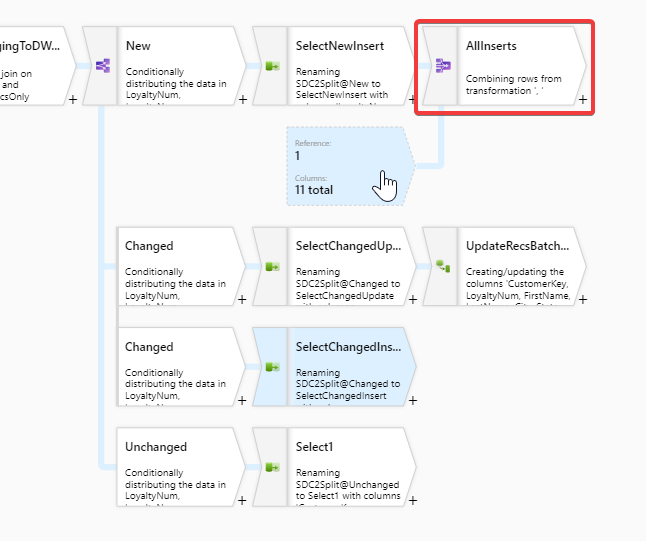
For the “*Unchanged*” stream we leave it without any transformation after it.



#### Putting together all inserts “New” and “Changed” together

Out of stream “New” (marked 1 in above figure) and stream “Changed” (marked 3 in above figure) we have two sets of records which needs to be inserted in to the DW table. The next logical step is to merge the two set into one using a “Union” transformation.

1. After “*SelectNewInsert*” transform add a “Union” transformation and rename it to “*ALLInserts*”



1. Select Union by: Name
2. Union With: SelectChangedInsert

**Union transformation options:** Union transformation allows merging two data sets either by matching column names (Union by name) or merge columns from two datasets in the order of columns they are. And “Union With” is where we select the other dataflow stream(s) that we want to union with the current stream. You can add multiple streams by using the Plus sign to add more streams.

1. Preview the output of the transform.

#### Generate Surrogate Keys

For all the records to be inserted in the DW table we need to generate a surrogate key (more information [here](https://en.wikipedia.org/wiki/Surrogate_key)).

Surrogate key transformation essentially is a sequential number generator. The “Key Column” sets the name of the column in the output dataset for surrogate key and the “Start Value” designates the integer number that the sequential number starts at.

1. After “AllInserts” transform add a “SurrogateKey” transromation
2. Rename it to “SurrogateKey”
3. Key column: CustomerKey
4. Start value: 1

**The Surrogate key issue**: With the above settings every time the dataflow runs, a number will be generated for each record which passes through the “surrogate key” transform, starting at 1. So, if we leave things as is the surrogate key generated will be worthless as it will duplicated without any relation to existing records in the DW table.

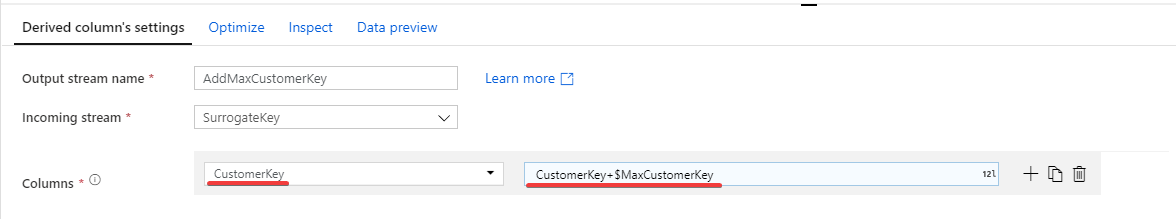
**The solution:** To solve this problem we need to know the maximum value of surrogate key in the DB and add it to the generated number.

#### Fix surrogate key value

At the beginning of this exercise, you created a parameter named “*MaxCustomerKey*”. The purpose of this parameter is to hold the biggest surrogate key currently in the DW table and add it to the generated number.

The parameter gets filled in by the ADF pipeline that dataflow runs in it. Once dataflow is ready to be place in pipeline we will precede it with a “lookup” activity which retrieves the maximum of “CustomerKey” in the table and pass it on to the DF through the parameter (More on this later)

1. After the “SurrogateKey” transromation add a “Derived Column” transformation
2. Rename it to “AddMaxCustomerKey”
3. Set it like the screenshot



CustomerKey:

CustomerKey+$MaxCustomerKey

This transformation takes the generated surrogate key sequential number and add the maximum surrogate key value to it.

#### Add Batch columns to our “Insert” dataset

Our “Insert” dataset is still missing the following 4 columns. In this task a “Derived Column” activity is used to generate them and add them to the dataset.

|  |  |  |
| --- | --- | --- |
| RecInsertDt date | Actual ELT running date | Generated by ELT |
| RecStartDt date | Record validity start date = batch date | Generated by ELT |
| RecEndDt date | Record validity end date = batch date | Generated by ELT |
| RecCurrInd Boolean | Record validity indicator | Generated by ELT |

1. Add a “Derived Column” transformation
2. Rename it to InsertRecsBatchColumns
3. Columns:

* RecInsertDt:

currentDate()

* RecCurrInd

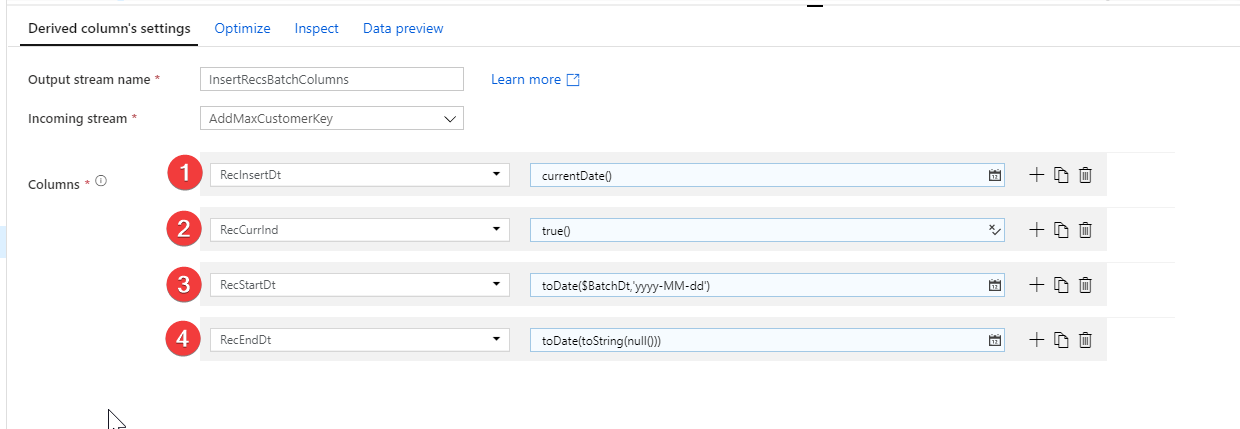
true()

* RecStartDt

toDate($BatchDt,'yyyy-MM-dd')

* RecEnddt

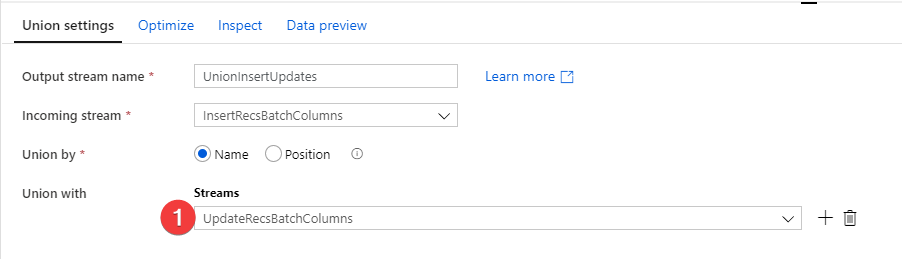
toDate(toString(null()))



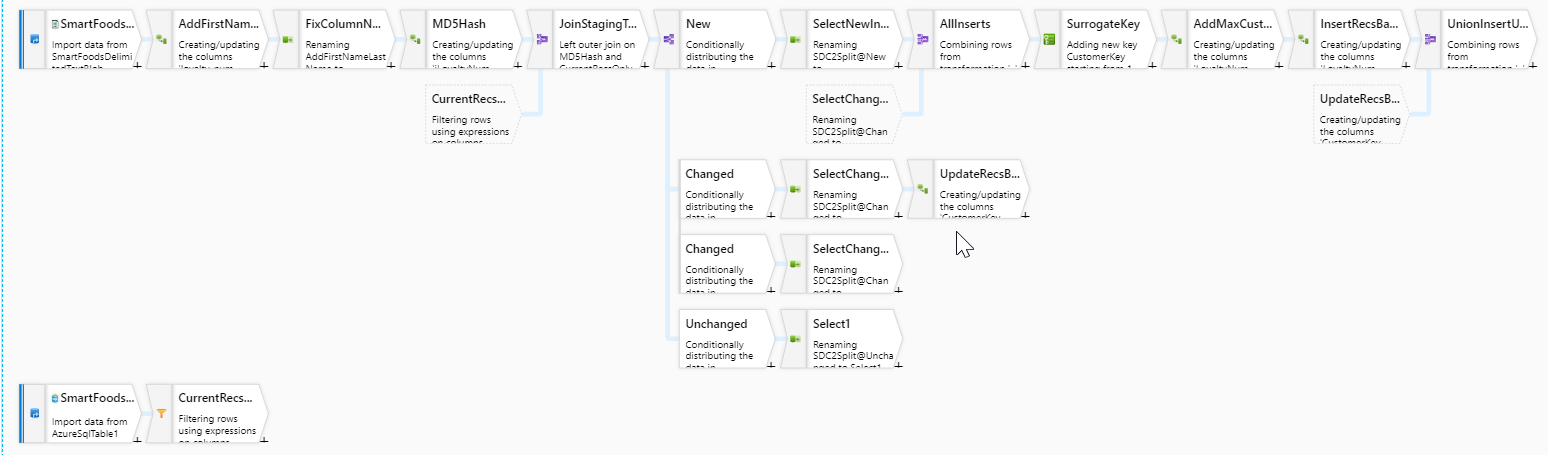
#### Put Insert and Update records together

Now we need to merge the insert stream and update stream together in to a single stream in preparation for pushing it to the destination.

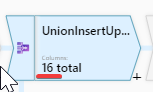
1. Add a “Union” transform
2. Rename it to “*UnionInsertUpdates*”
3. Union with: “*UpdateRecsBatchColumns*”



So far, your Dataflow should look like below

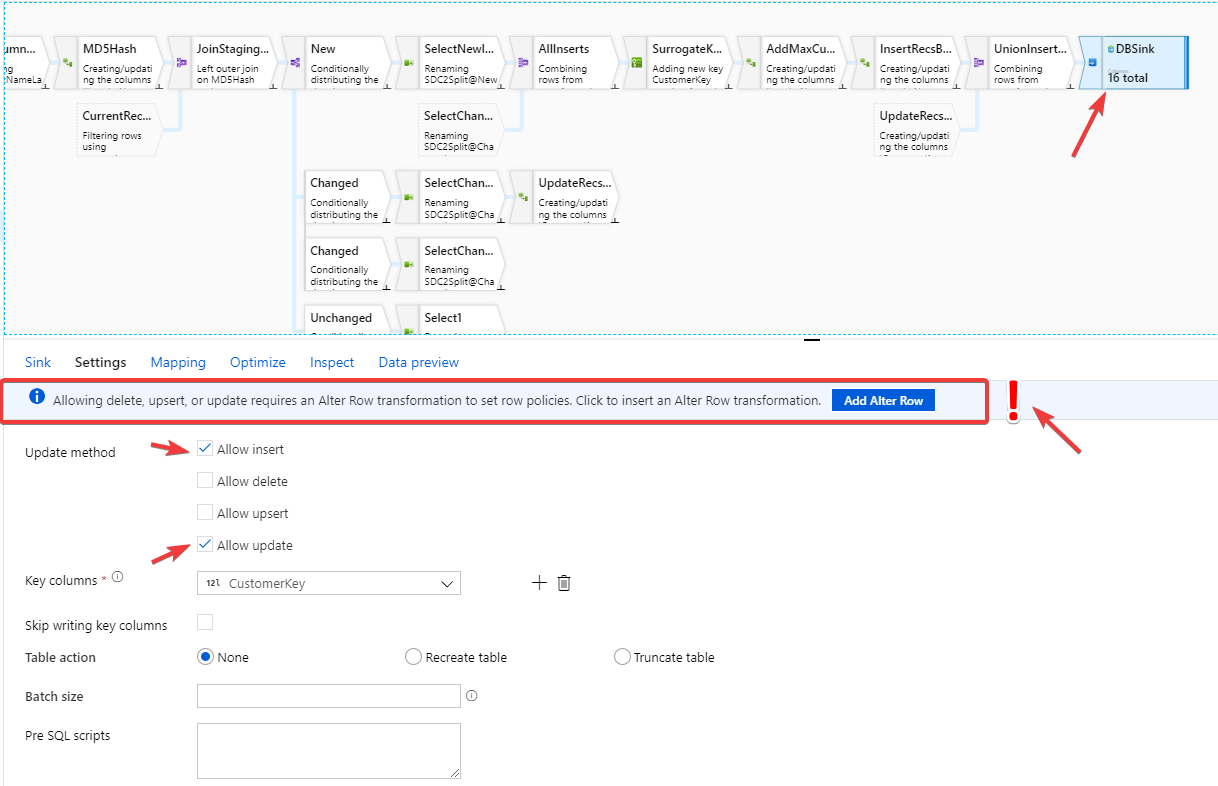


**Note:** If you hover over any transformation it tells you how many output columns are coming out of that transformation. To double check you work the last transformation should have **16** columns



#### Prepare the dataset for sink (Alter row transformation)

Now that our dataset is ready to be written to the DW table. Next logical step is to add a “Sink” transform and complete the data flow. let’s see what happens if we go ahead and add sink transform



A sink transformation that is writing to a Database (or DW) and requires to perform anything other than insert (update, upsert or delete) on the destination table, requires an additional transformation to mark each row with the type of operation “Sink” transform is expected to perform. That transformation is called “Alter Row”.

**Alter row:** Alter row transformation is used when sink is an RDBMS (DB or DW) and is expected to perform Update, Upsert or Delete.

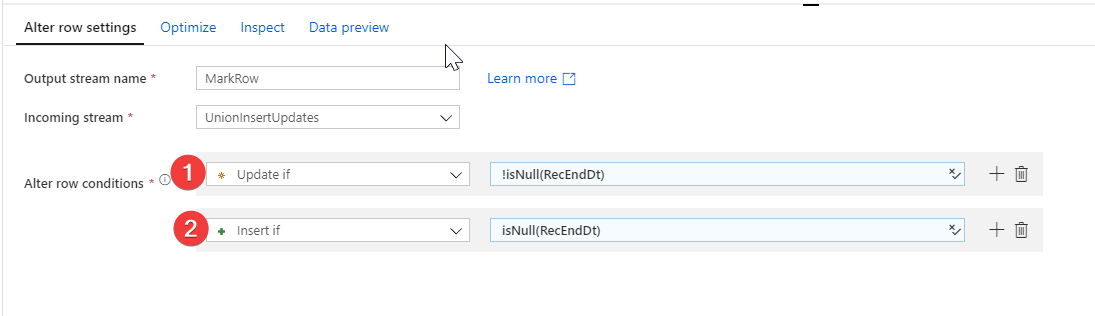
1. After “*UnionInsertUpdates”* transform add an “Alter row” transform
2. Rename it to “*MarkRow*”
3. Add two Alter row conditions:

* Update if

!isNull(RecEndDt)

* Insert if

isNull(RecEndDt)

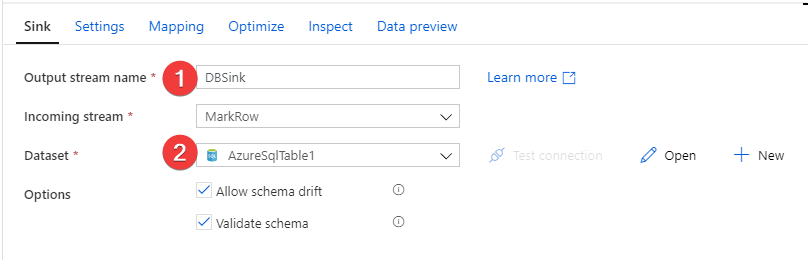


**Update if**: this is a new record (RecEndDt IS NOT Null)  
**Insert if**: This is an existing record being closed (RecEndDt IS Null)

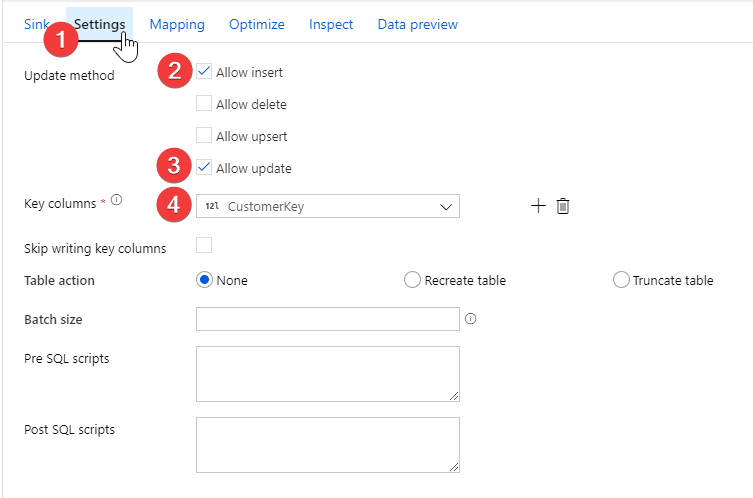
#### Writing to destination DW table

Now the dataflow is ready to add the Sink Transformation and write to destination table.

1. After “*MarkRow”* transform ad a “Sink” transform
2. Rename it to “DBSink”
3. Dataset: AzureSqlTable1



1. Go to “Settings” Tab
2. Tick “Allow Insert”
3. Tick “Allow Update”
4. Key Columns: “CustomerKey”

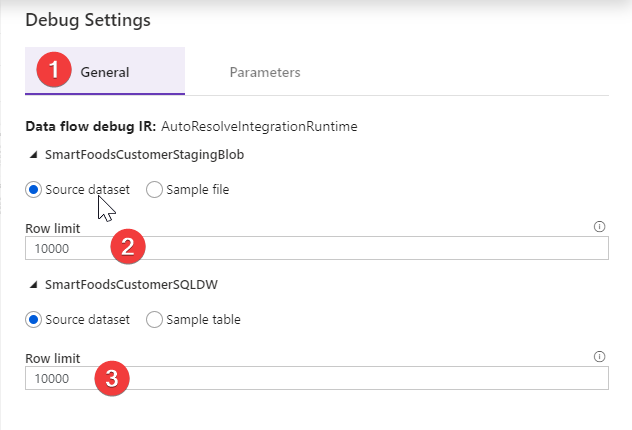


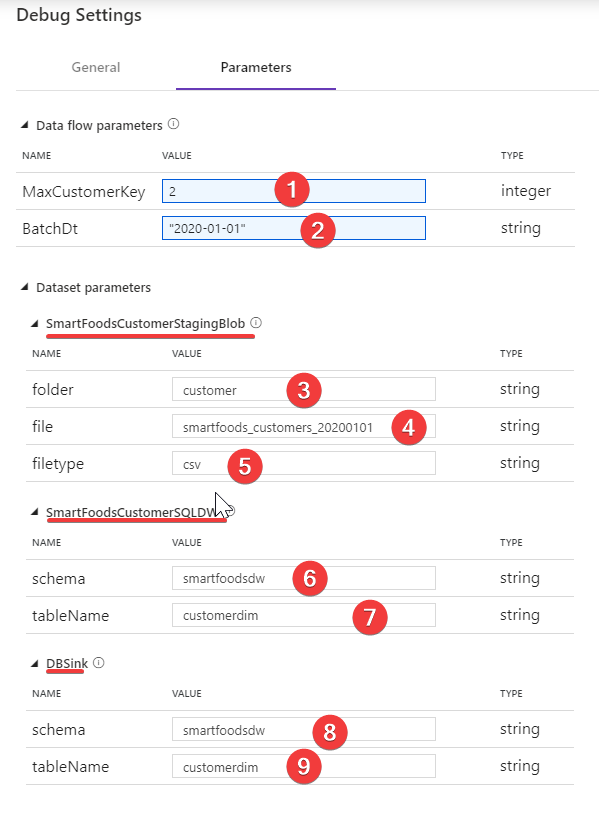
**Reminder:** Delete the “New Branch” and dummy “Sink” transforms!

#### Preview the final dataset (Debug)

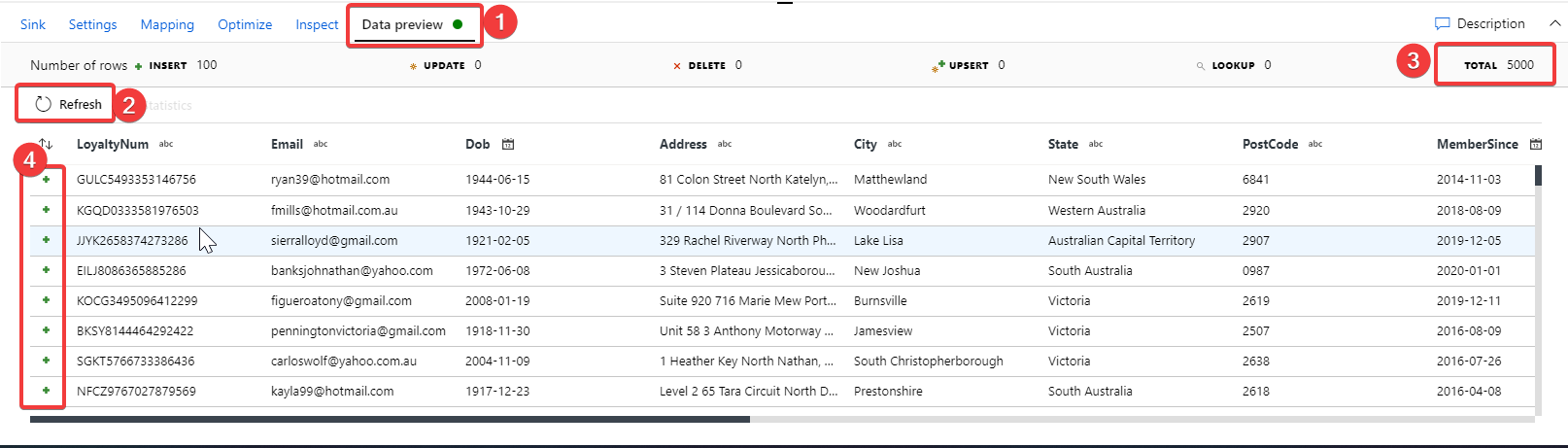
The final task before publishing the dataflow is to have a look at the final dataset through “Data preview” in the “Sink” transform.

1. Go to “Debug Settings” and make sure your parameters matches below screenshots.





1. In “DBSink” Under “Data Preview” click “Refresh”

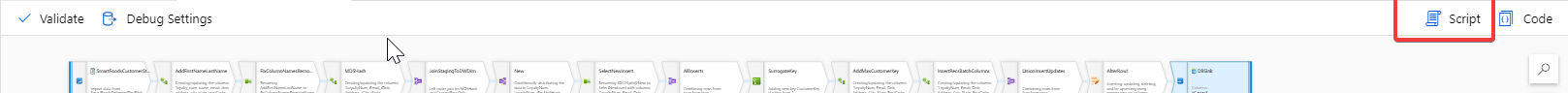


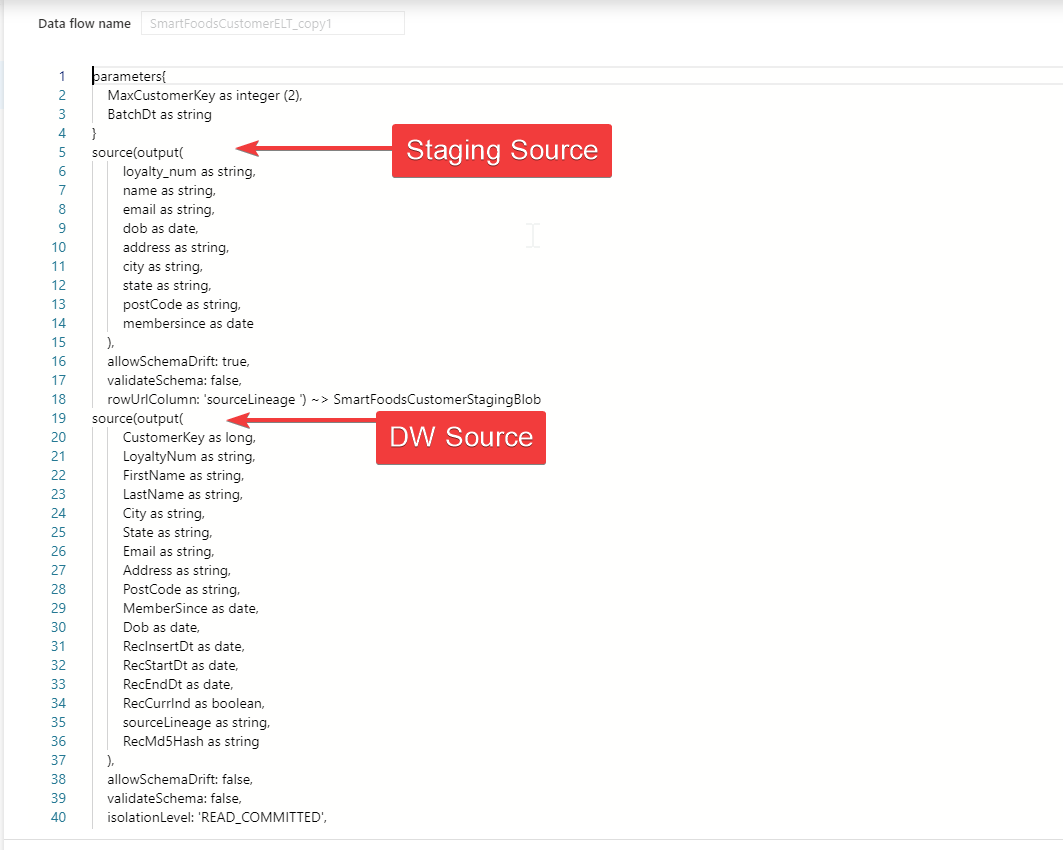
In the figure above number 3 shows the total number of rows that going to go through this transformation. In “Debug” mode this number is capped at either the “Row Limit” or the actual maximum number of rows in the source. Here our “Row limit” is 10,000 and the file has 5,000 rows in it. Also Number 4 indicates what type of action is being perform on each of the preview rows, the green plus sign indicates an “insert”.

**Note:** Running “Data preview” on “Sink” transform WILL NOT actually write anything to the destination. It only previews what the write would look like. ADF Mapping DF “Debug” will not perform any action on “Sink” transformation until the DF is placed in an ADF pipeline and debugged from there.

#### Inspect the Dataflow “Script”

ADF mapping DF, generates a script for the GUI changes we make. You can access and update this script by clicking the “Script” button on the top ribbon.



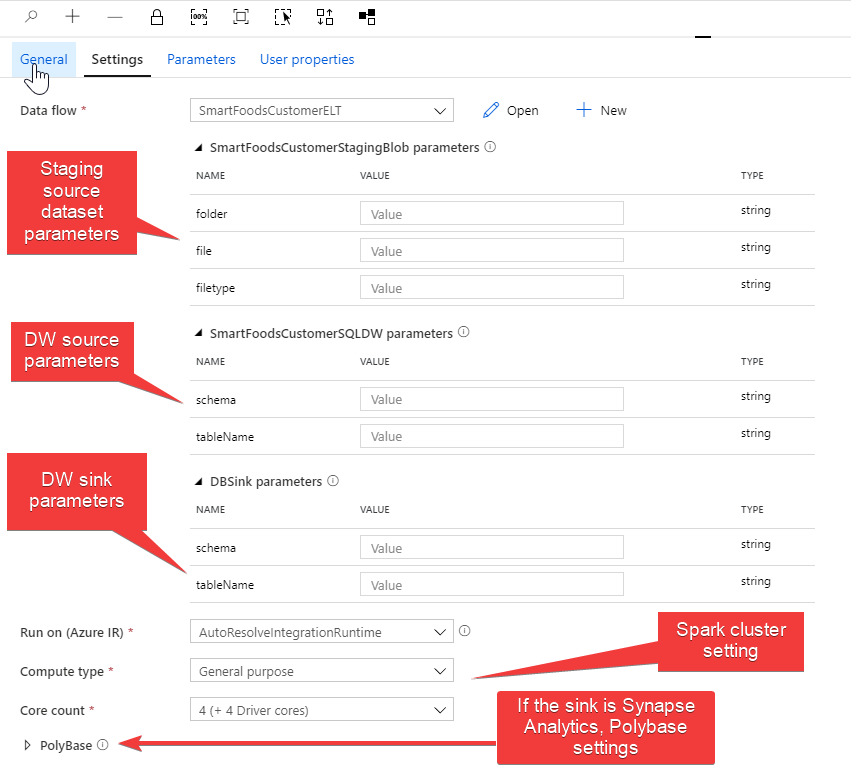


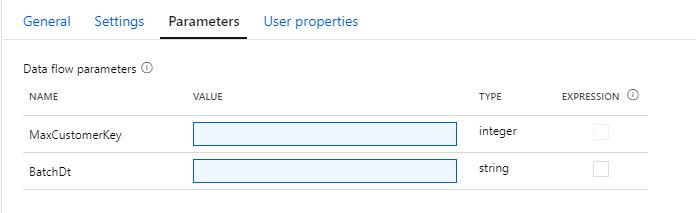
**Note:** In the Github repo under ./Hands-on-lab/Part2/Appendix there is a file named “SmartFoodsCustomerELTScript.txt” with the complete solution script. There are instructions inside the file on how to use it.

#### Building the pipeline for the dataflow

Now that the dataflow is published the next task is to create a pipeline and add the dataflow to it.

1. Create a pipeline
2. Rename it to “SmartFoodsCustomerELTBlobToSql”
3. Under “Move & Transfrom” drag a “Data flow” activity to the pipeline.
4. Select “SmartFoodsTransactionELT” from list of Dataflows
5. The setting tab of the DF activity should look like below figure. All the dataset parameters surfaced on the “Settings” tab and dataflow’s parameters are under “parameters” tab

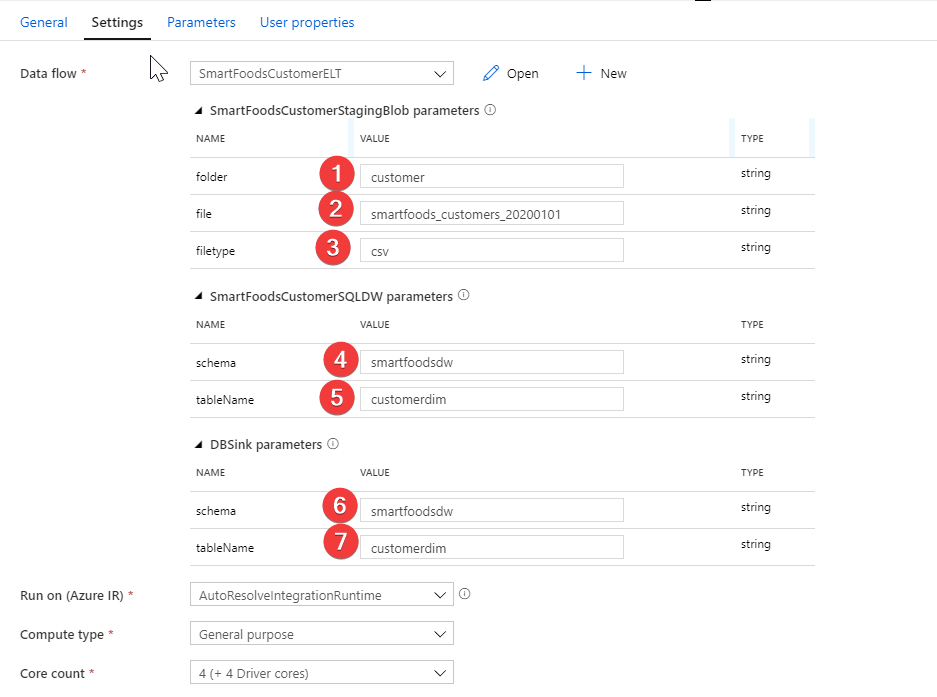




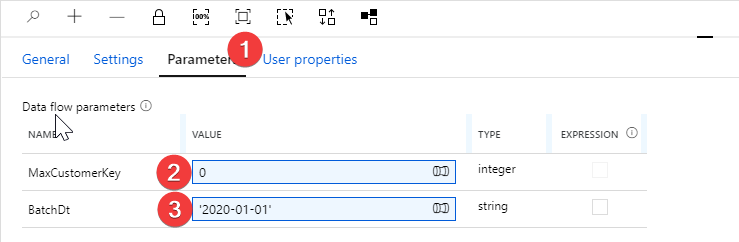
#### Debug the pipeline manually

As a first step, we will debug the pipeline and enter the parameters manually. Then we modify the pipeline to a more production ready state.

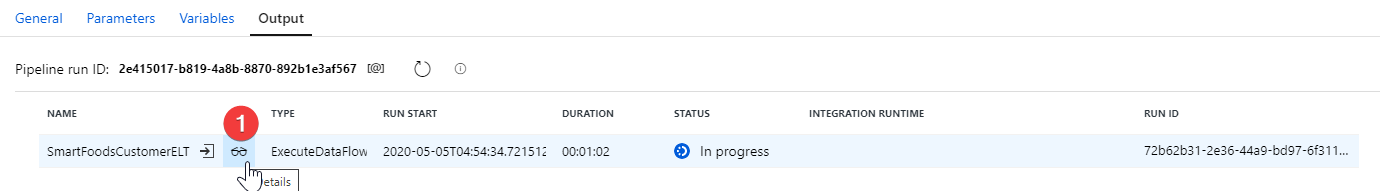
1. From parameters tab click on “MaxCustomerKey” value and select pipeline expression
2. Fill in the dataset parameters as:



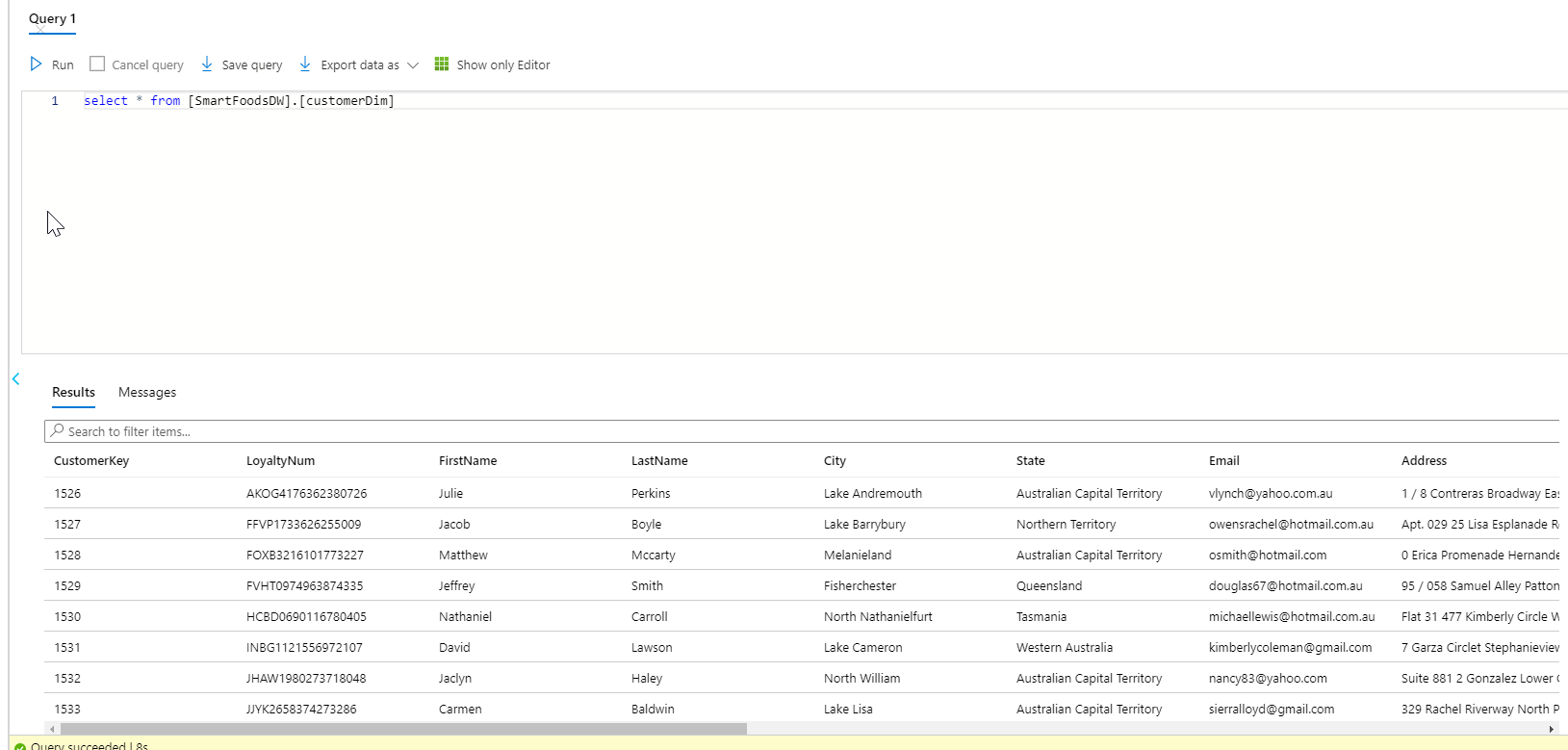
1. Fill in dataflows patamers as:



1. Click “Debug”.
2. Examine the dataflow monitoring by click the “eye glass” icon – (More on Dataflow monitoring later)



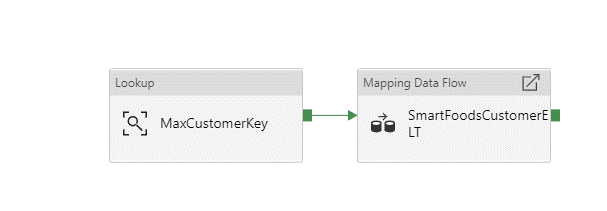
1. Once the debug finishes successfully examine the table contents either through SQLDB “Query editor” or using SSMS.



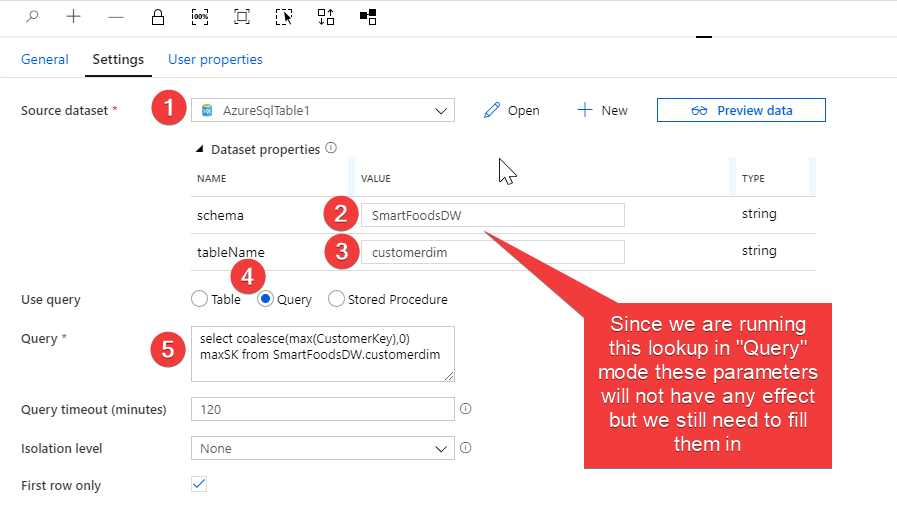
#### Enhance the pipeline

Once we debugged the pipeline with manual parameters successfully next step is to automate those parameters filling like what we have done for pipelines loading data from the source.

1. Add a lookup activity before the dataflow activity to retrieve the maximum Customerkey
2. Rename it to “*MaxCustomerKey*”



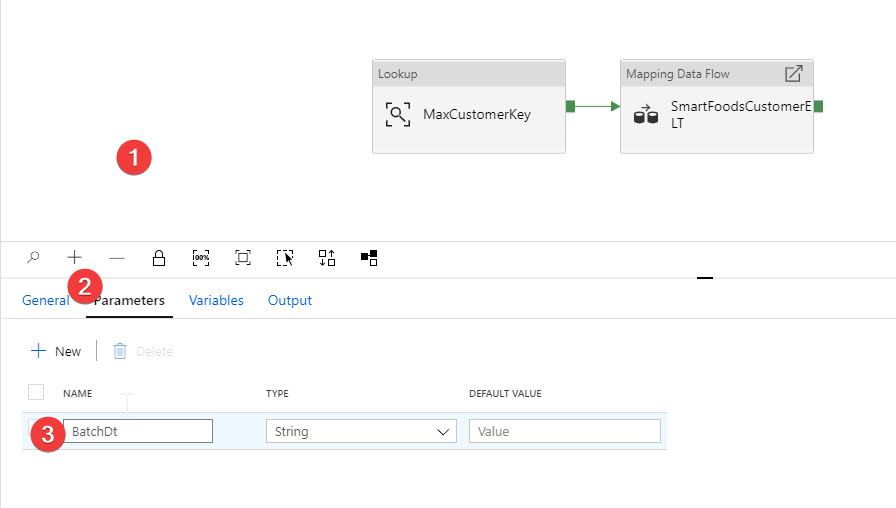
1. Set up the lookup as below:



Query:

select coalesce(max(CustomerKey),0) maxSK from SmartFoodsDW.customerdim

1. Create a pipeline parameter as BatchDt

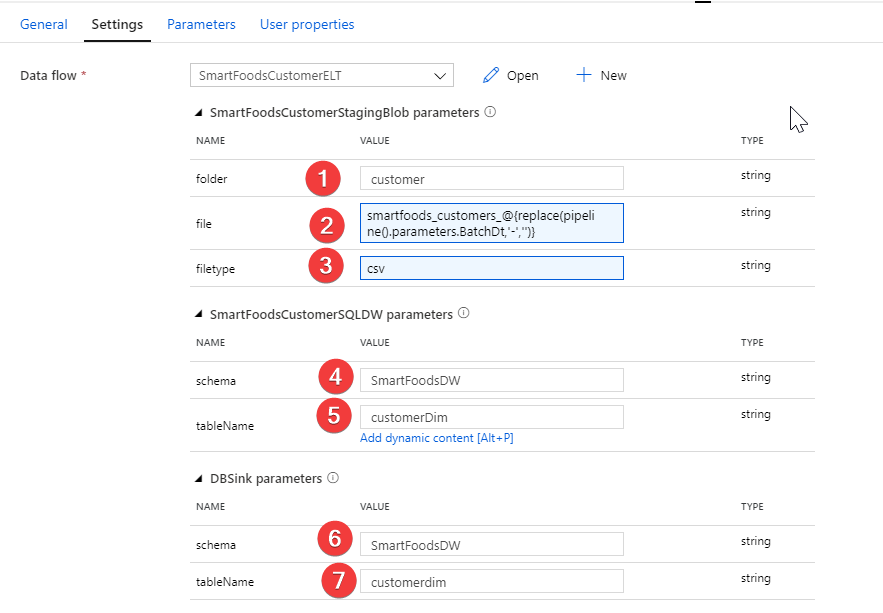


1. Go to DF activity and setup dataset parameters as

* Folder: customer
* File:

[smartfoods\_customers\_@{replace(pipeline().parameters.BatchDt,'-','')}](mailto:smartfoods_customers_@%7breplace(pipeline().parameters.BatchDt,'-','')%7d)

* Filetype: csv
* Schema: SmartFoodsDW
* tableName: customerDim
* Schema: SmartFoodsDW
* tableName: customerDim



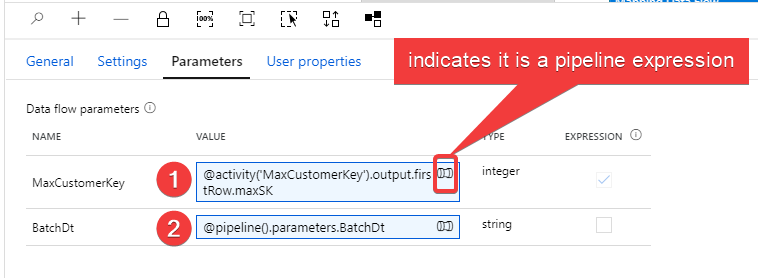
1. Under parameters tab setup dataflow parameters as:

* MaxCustomerKey -> pipeline expression

@activity('MaxCustomerKey').output.firstRow.maxSK

* BatchDt -> pipeline expression

@pipeline().parameters.BatchDt



1. Now try debugging this pipeline and the only parameter that needs to be passed is “BatchDt” pipeline parameter.

#### (Challenge Task) Create an initial load pipeline for this DF

Create another pipeline to run multiple dates of the Customer ELT pipeline in a loop

Hint1: create an array variable with values:

["2020-01-01","2020-01-02","2020-01-03","2020-01-04","2020-01-05","2020-01-06"]

Hint2: Use ForEach Loop.

Hint3: ForEach loop must run sequentially not in parallel. (why?)

Hint4: Inside the loop use execute pipeline activity.

Hint5: If you needed extra help check the screenshots below.

