# Lab 04 – Building a Mapping Data Flow

This lab assumes you've created Lab 3 and have imported the following tables into your ADLS Gen 2 instance:

- SalesLT.Product
- SalesLT.ProductCategory
- SalesLT.ProductModel

Note: If your data factory instance is fairly old, it might not have the preview Mapping Data Flow activity. For the purpose of this lab – create a new data factory following Lab 01.

We're going to take a fairly basic warehousing example – take three tables and combine them into a basic "dimension" table.

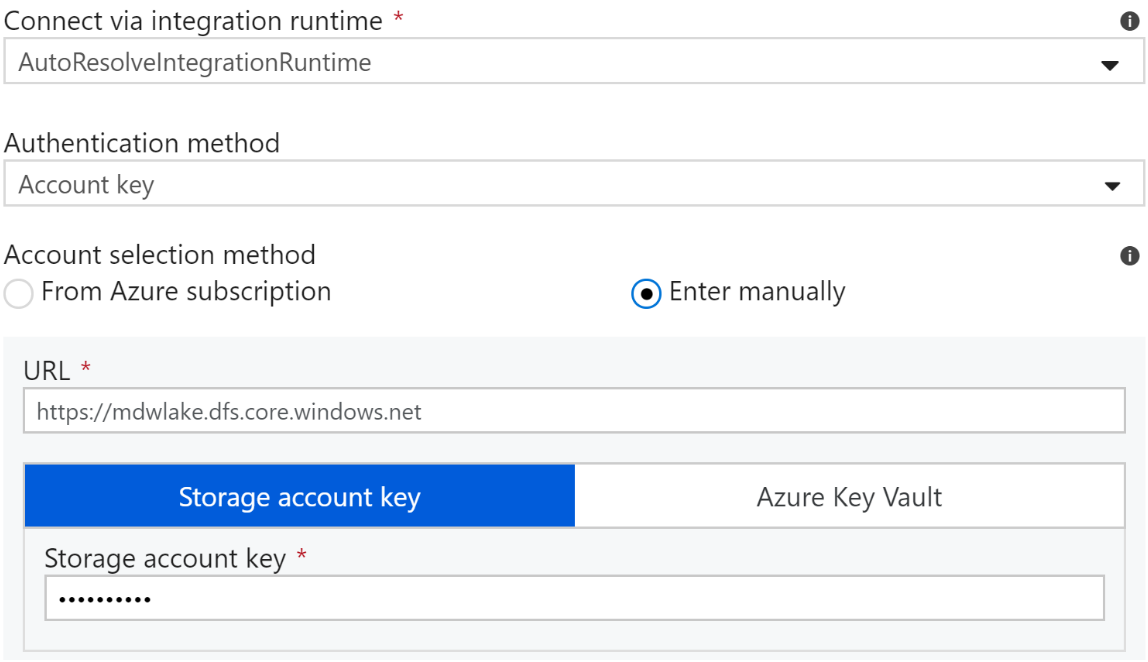
We will use the lake copy of the Adventureworks database we used in previous examples – let's assume we're pulling the "name" category from the SalesLT.Product, SalesLT.ProductCategory and SalesLT.ProductModel tables into a single "dimension"-style table.

Basically, we're duplicating this query:

```
select P.ProductID,
    P.Name ProductName,
    PC.Name ProductCategory,
    PM.Name ProductModel
from SalesLT.Product P
    inner join SalesLT.ProductCategory PC on P.ProductCategoryID =
PC.ProductCategoryID
    inner join SalesLT.ProductModel PM on P.ProductModelID = PM.ProductModelID
```

## LAB 04.A Create Datasets

1. Data Flows are a little awkward as they're still in preview – they can't connect to an ADLS Gen 2 lake using a service principal just yet. So create a new Linked Service that uses the "Account Key" style of connection to your lake, just for this lab



Connect via integration runtime *

AutoResolveIntegrationRuntime

Authentication method

Account key

Account selection method

○ From Azure subscription ● Enter manually

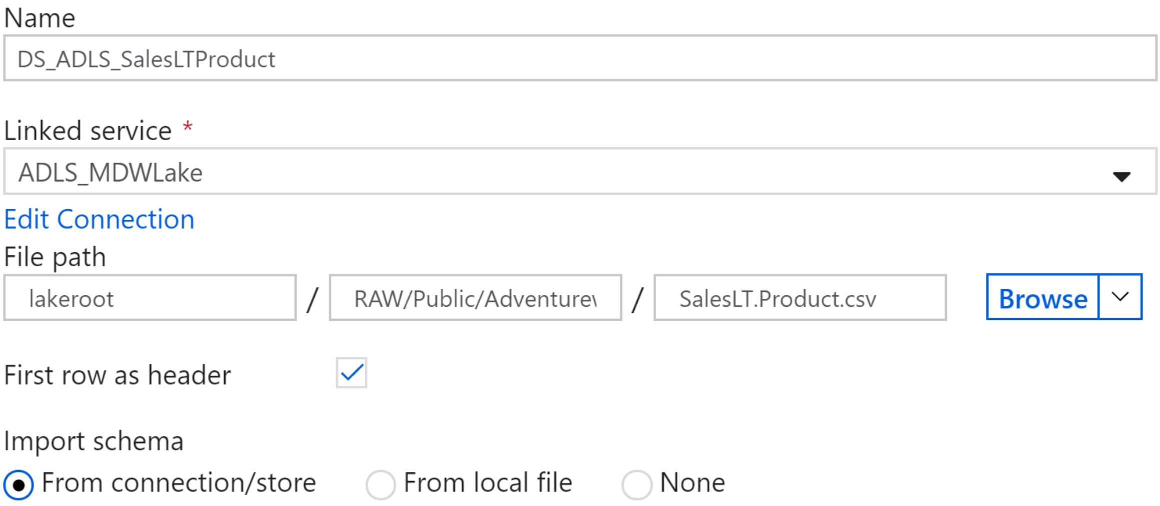
URL *

https://mdwlake.dfs.core.windows.net

Storage account key *

••••••••••

2. Just as we have before, we need to create three datasets pointing to files within our late. Create a Data Lake Store Gen 2 dataset, for a CSV file and select the Product tables we imported earlier:



Name

DS_ADLS_SalesLTProduct

Linked service *

ADLS_MDWLake

File path

lakeroot / RAW/Public/Adventure\ / SalesLT.Product.csv

First row as header ☑

Import schema

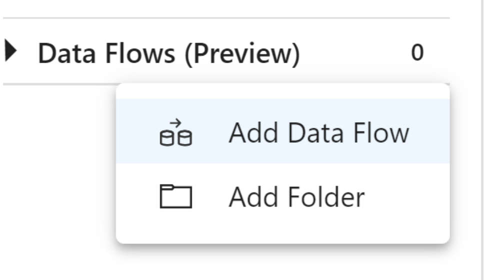
● From connection/store ○ From local file ○ None

3. Repeat the above steps for ProductCategory and ProductModel

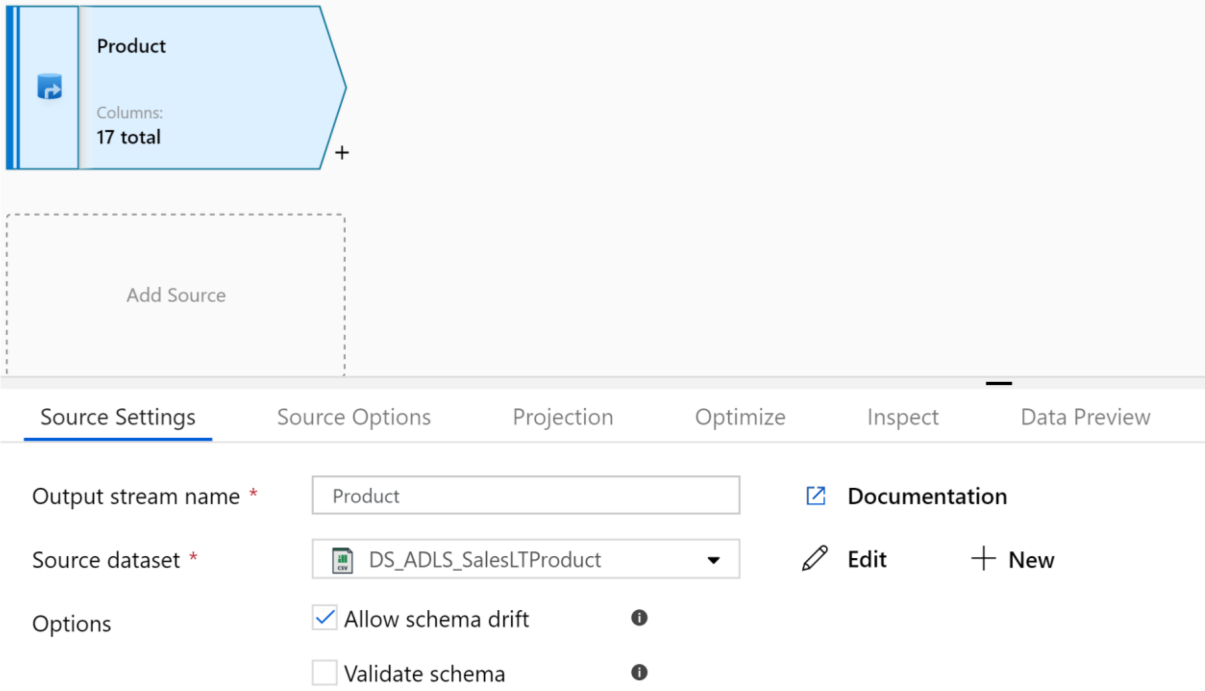
## LAB 04.B – Create a Data Flow

1. Create a new Data Flow

## Lab 04 – Building a Mapping Data Flow



2. You'll be guided to add a source – click on the empty square and it will create a "source" stream component. Name this "Product" and select the DS_ADLS_SalesLTProduct dataset



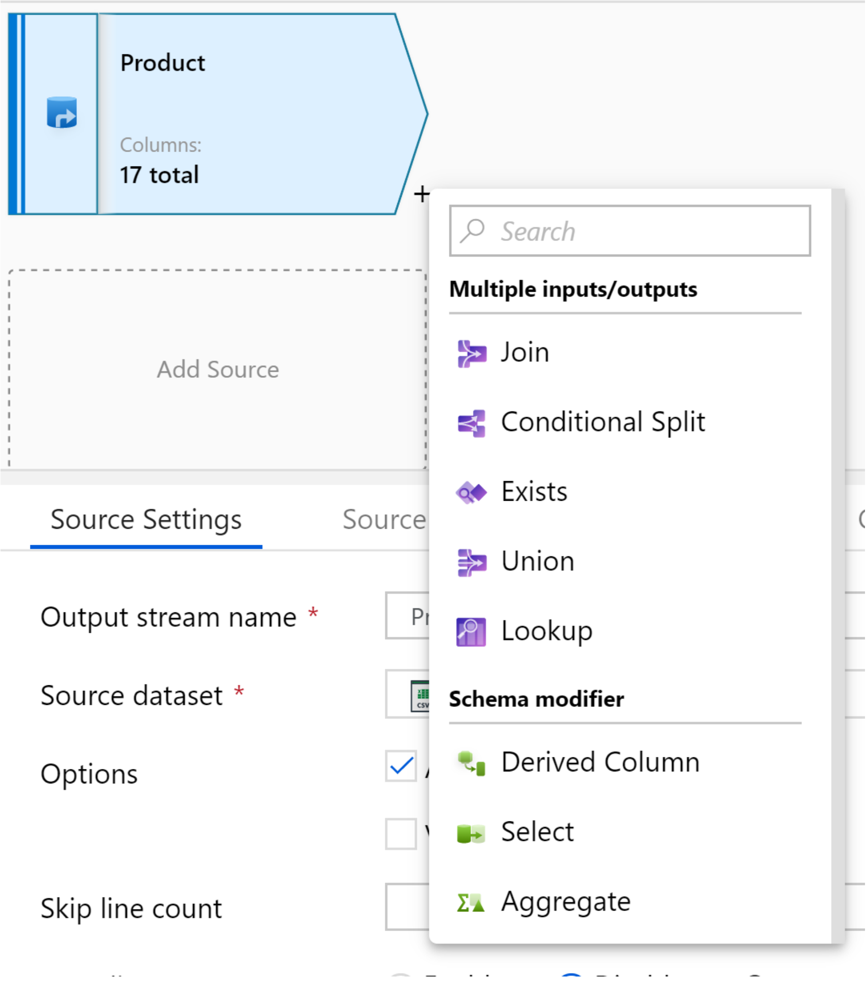
Output stream name * Product

Source dataset * DS_ADLS_SalesLTProduct

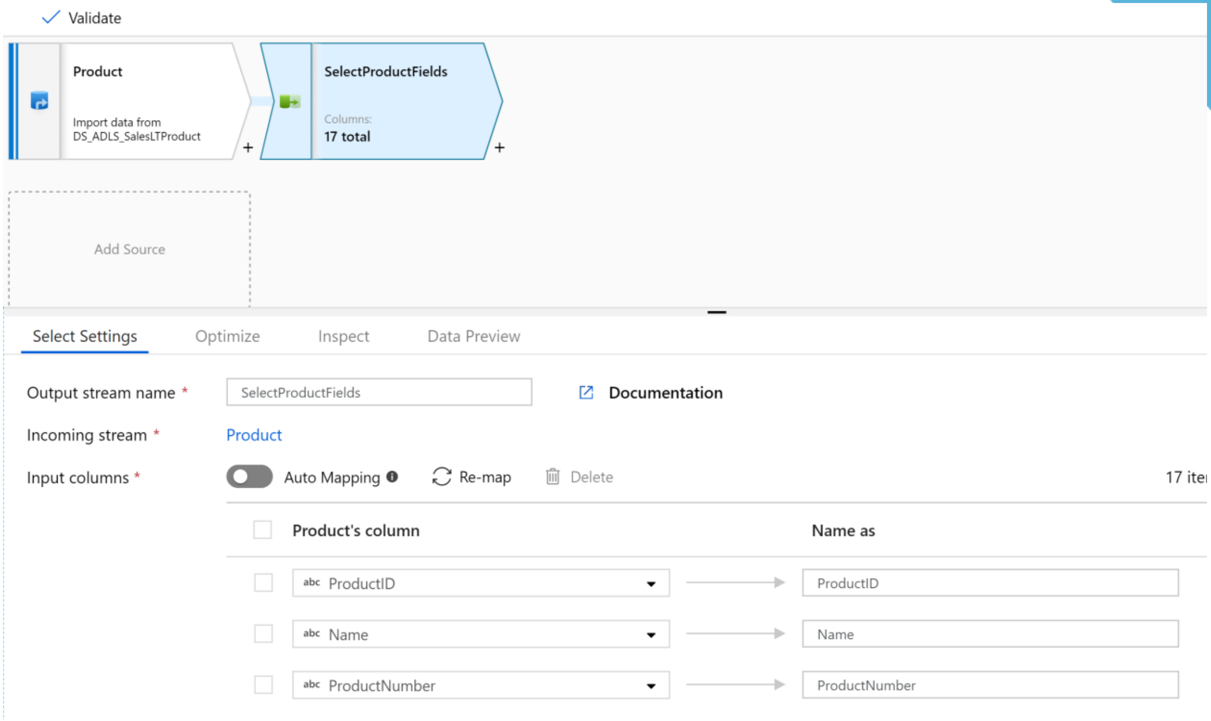
Options ☑ Allow schema drift

□ Validate schema

3. Click the small + icon next to the source stream to add a transformation:



4. We want to get rid of excess columns, so let’s use a “Select” transformation. Give the transformation step a name:



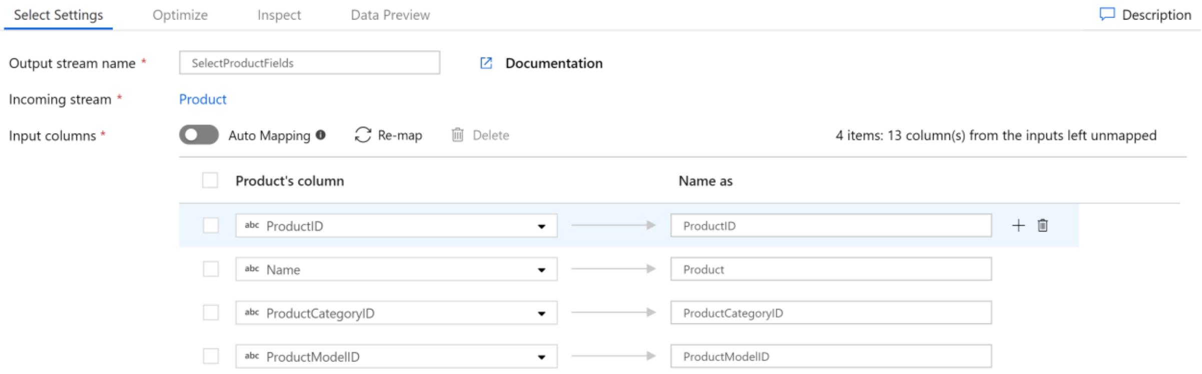
Output stream name * SelectProductFields

Incoming stream * Product

Input columns * 17 ite

| Product's column | Name as |
|---|---|
| abc ProductID | ProductID |
| abc Name | Name |
| abc ProductNumber | ProductNumber |

5. We want to rename “Name” to “Product” and remove all columns except for ProductID, ProductCategoryID, ProductModelID and our renamed “Product” attribute.



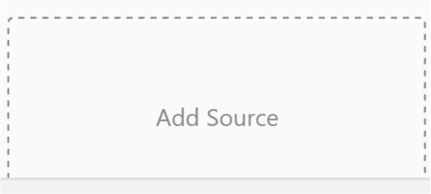
Output stream name * SelectProductFields

Incoming stream * Product

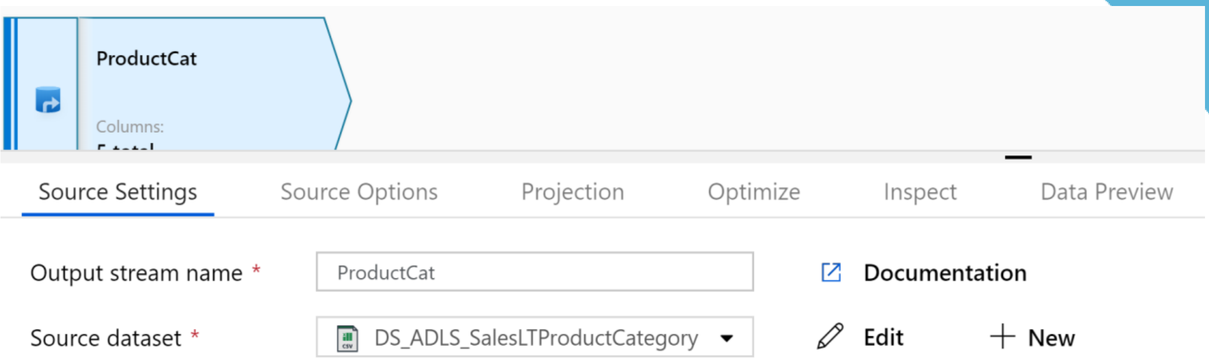
Input columns * 4 items: 13 column(s) from the inputs left unmapped

| Product's column | Name as |
|---|---|
| abc ProductID | ProductID |
| abc Name | Product |
| abc ProductCategoryID | ProductCategoryID |
| abc ProductModelID | ProductModelID |

6. We now want to click on the “Add Source” button to add a source for our next table:



7. Configure this new source to use our ProductCategory dataset:



ProductCat

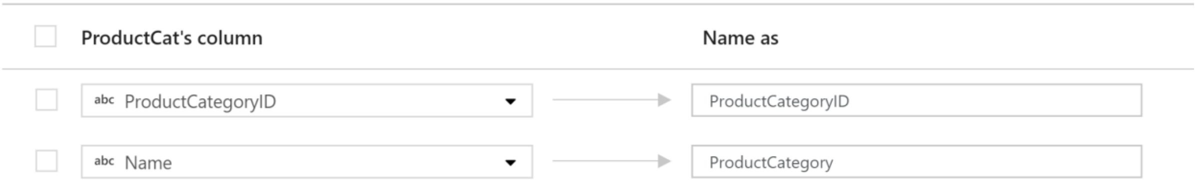
Columns:

Source Settings | Source Options | Projection | Optimize | Inspect | Data Preview

Output stream name * ProductCat

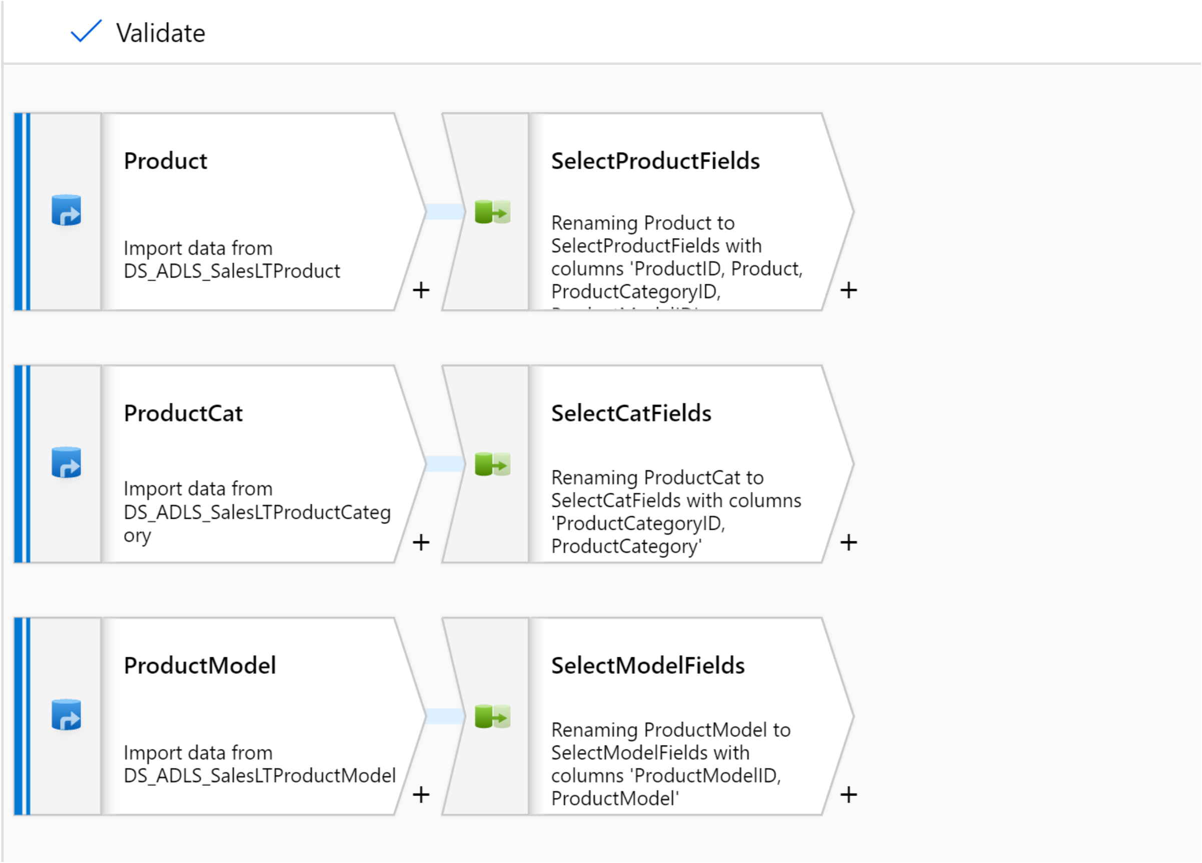
Source dataset * DS_ADLS_SalesLTProductCategory

8. Add Select Transform to rename “Name” to “ProductCategory” and remove all columns except for the ProductID



| □ ProductCat's column | Name as |
|---|---|
| □ abc ProductCategoryID | ProductCategoryID |
| □ abc Name | ProductCategory |

9. Do the same for the ProductModel, again stripping out columns aside from the ProductModelID and the name (renamed as ProductModel)



Product

Import data from DS_ADLS_SalesLTProduct

SelectProductFields

Renaming Product to SelectProductFields with columns 'ProductID, Product, ProductCategoryID,

ProductCat

Import data from DS_ADLS_SalesLTProductCategory

SelectCatFields

Renaming ProductCat to SelectCatFields with columns 'ProductCategoryID, ProductCategory'

ProductModel

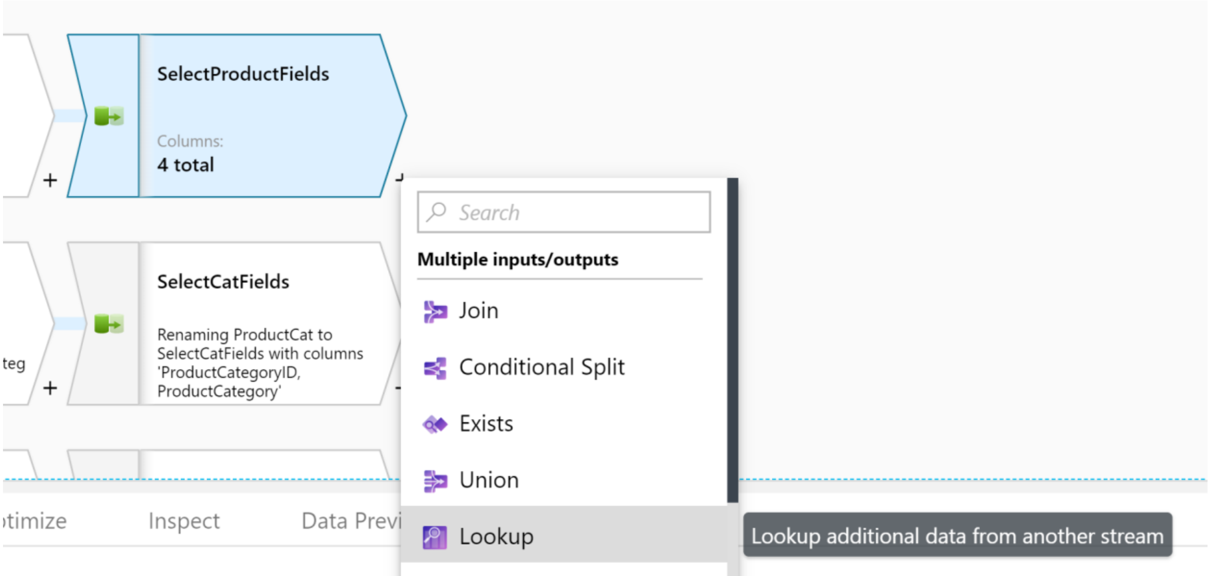
Import data from DS_ADLS_SalesLTProductModel

SelectModelFields

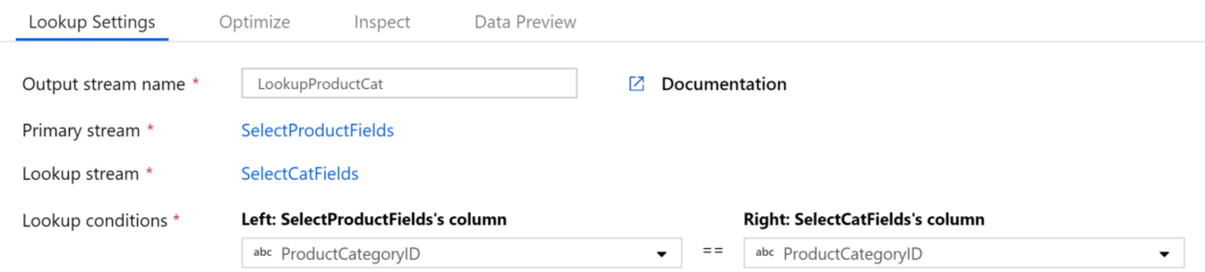
Renaming ProductModel to SelectModelFields with columns 'ProductModelID, ProductModel'

We now have our source data, nicely trimmed of excess fields that we can combine into a single dataset.

10. Add a new transformation after the “select” on the main Product stream and choose the “Lookup” type – this is where we will lookup information from the other streams



11. Select the “SelectCatFields” stream as the reference stream and configure it to use the ProductCategoryID to perform the reference join, like so:


Lookup Settings   Optimize   Inspect   Data Preview

Output stream name * LookupProductCat   Documentation

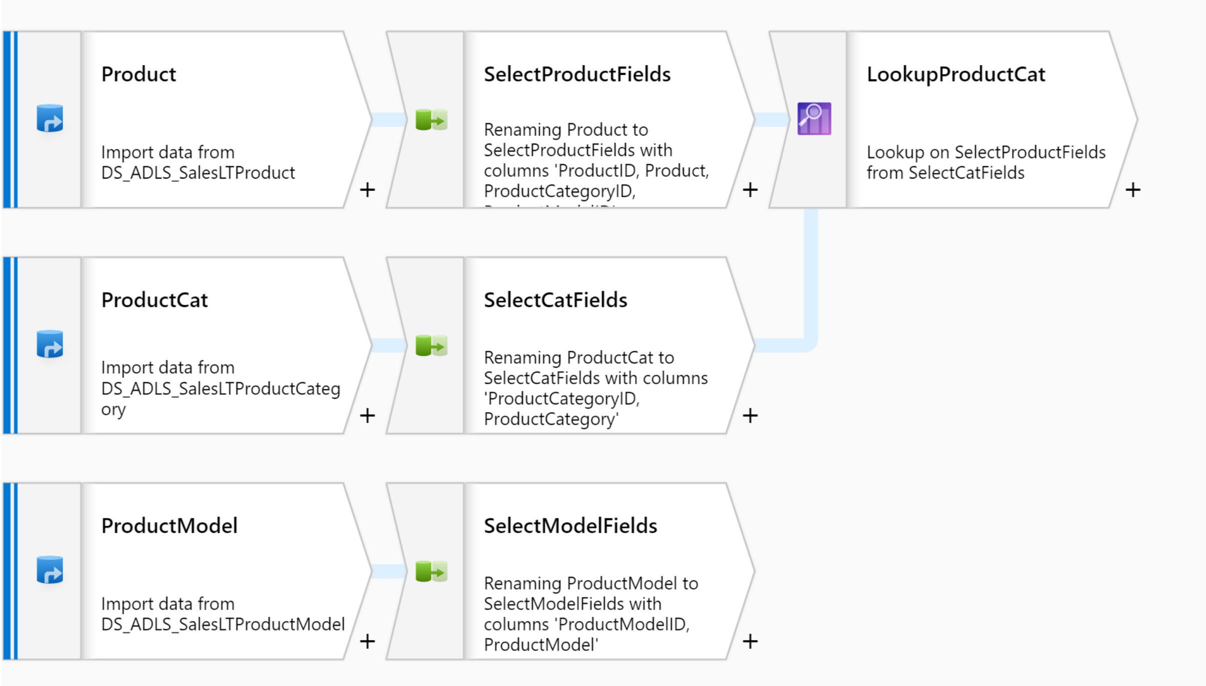
Primary stream * SelectProductFields

Lookup stream * SelectCatFields

Lookup conditions * Left: SelectProductFields's column abc ProductCategoryID == Right: SelectCatFields's column abc ProductCategoryID

Be careful here – you can actually select to use the output of any of the transformations, including the original source ones, before we had stripped columns and renamed them.

When your lookup is configured, it will automatically update your diagram reflect the relationship.



Product

Import data from DS_ADLS_SalesLTProduct

SelectProductFields

Renaming Product to SelectProductFields with columns 'ProductID, Product, ProductCategoryID,

LookupProductCat

Lookup on SelectProductFields from SelectCatFields

ProductCat

Import data from DS_ADLS_SalesLTProductCategory

SelectCatFields

Renaming ProductCat to SelectCatFields with columns 'ProductCategoryID, ProductCategory'

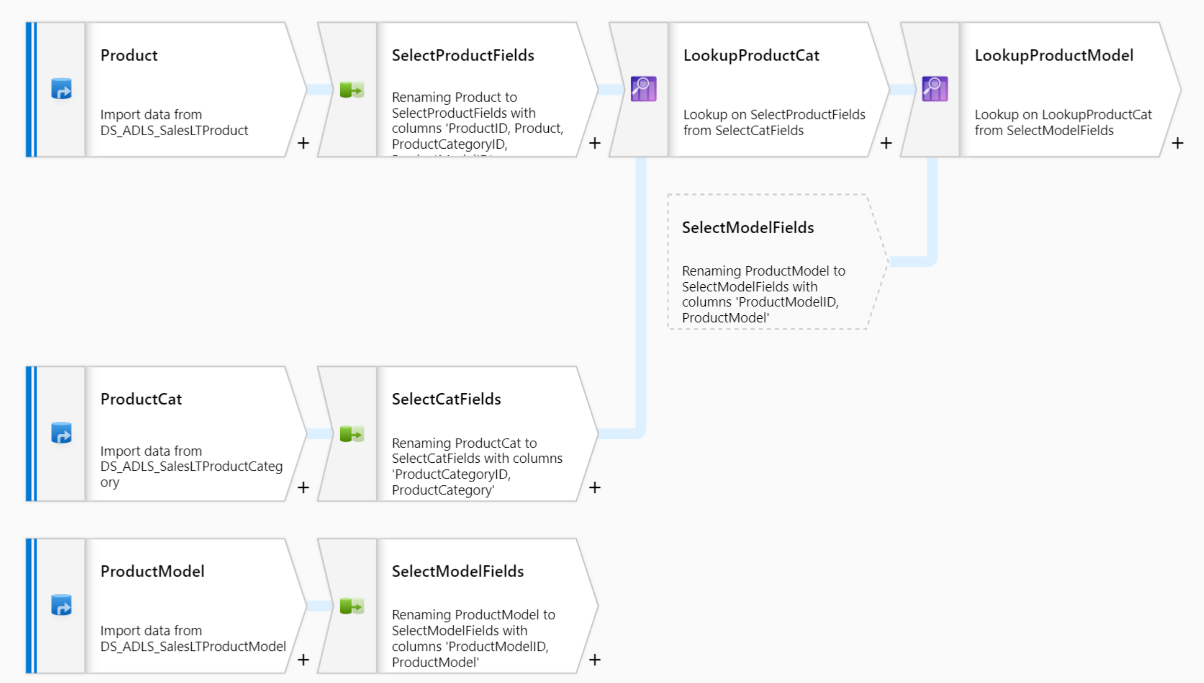
ProductModel

Import data from DS_ADLS_SalesLTProductModel

SelectModelFields

Renaming ProductModel to SelectModelFields with columns 'ProductModelID, ProductModel'

Do the same for the Product Model output stream:



Product

Import data from DS_ADLS_SalesLTProduct

SelectProductFields

Renaming Product to SelectProductFields with columns 'ProductID, Product, ProductCategoryID,

LookupProductCat

Lookup on SelectProductFields from SelectCatFields

LookupProductModel

Lookup on LookupProductCat from SelectModelFields

SelectModelFields

Renaming ProductModel to SelectModelFields with columns 'ProductModelID, ProductModel'

ProductCat

Import data from DS_ADLS_SalesLTProductCategory

SelectCatFields

Renaming ProductCat to SelectCatFields with columns 'ProductCategoryID, ProductCategory'

ProductModel

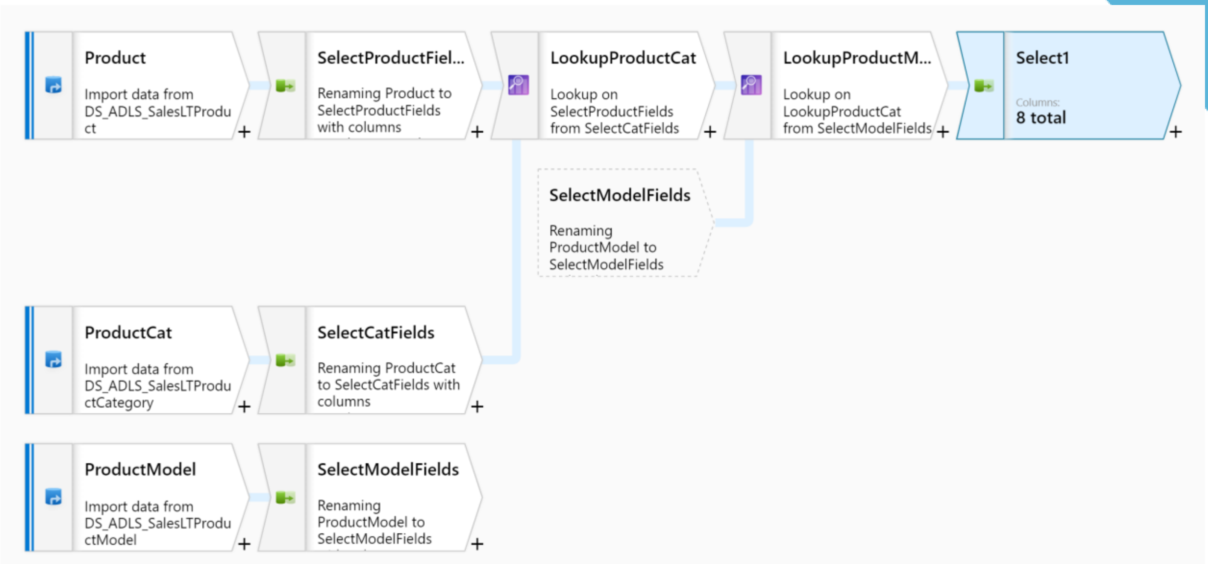
Import data from DS_ADLS_SalesLTProductModel

SelectModelFields

Renaming ProductModel to SelectModelFields with columns 'ProductModelID, ProductModel'

You may notice, we didn't select the fields to be added at any point, just those that are the join constraints. By default, it will bring the whole table into the aggregation, including duplicates of the keys.

12. Let's add a final select transformation to get rid of those duplicate keys:



Product
Import data from DS_ADLS_SalesLTProduct

SelectProductFiel...
Renaming Product to SelectProductFields with columns

LookupProductCat
Lookup on SelectProductFields from SelectCatFields

LookupProductM...
Lookup on LookupProductCat from SelectModelFields

Select1
Columns:
8 total

SelectModelFields
Renaming ProductModel to SelectModelFields

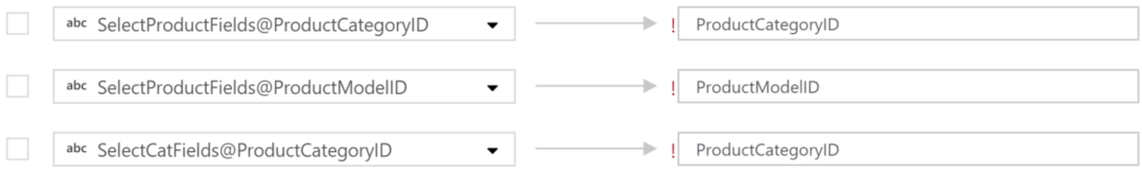
ProductCat
Import data from DS_ADLS_SalesLTProductCategory

SelectCatFields
Renaming ProductCat to SelectCatFields with columns

ProductModel
Import data from DS_ADLS_SalesLTProductModel

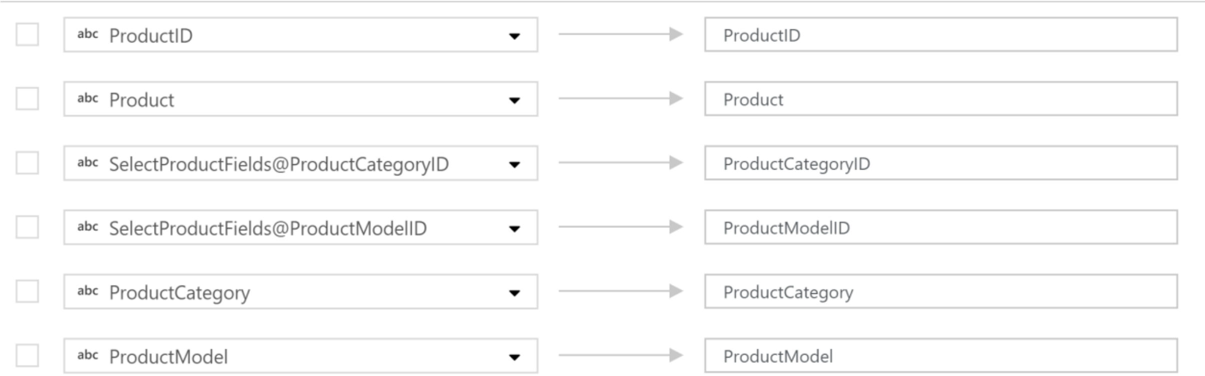
SelectModelFields
Renaming ProductModel to SelectModelFields

The duplicate columns will be highlighted and use a SELECT syntax to denote which stream they orginally came from:



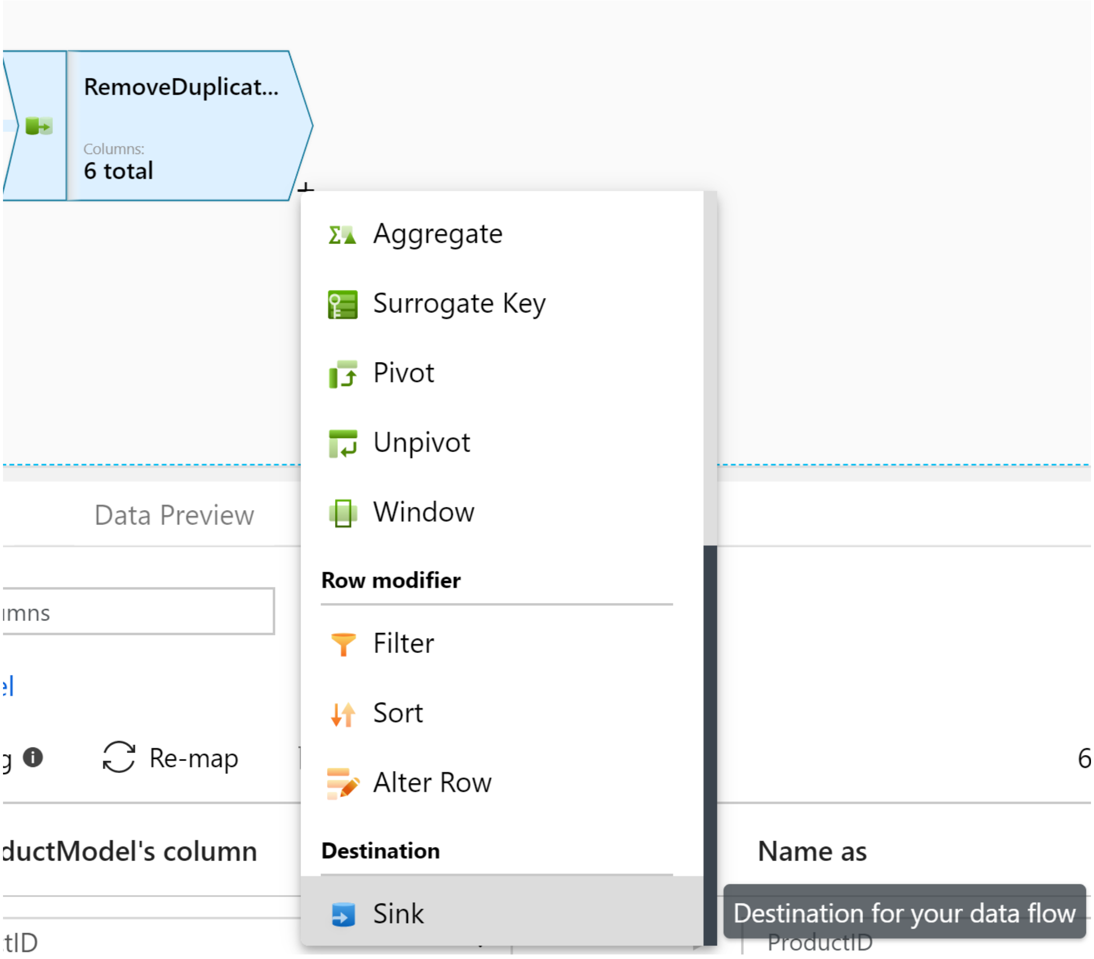
| | | | |
|---|---|---|---|
| □ | abc SelectProductFields@ProductCategoryID | → ! | ProductCategoryID |
| □ | abc SelectProductFields@ProductModelID | → ! | ProductModelID |
| □ | abc SelectCatFields@ProductCategoryID | → ! | ProductCategoryID |

Delete the columns that are duplicates sourced from our lookup tables, and we should see something like:



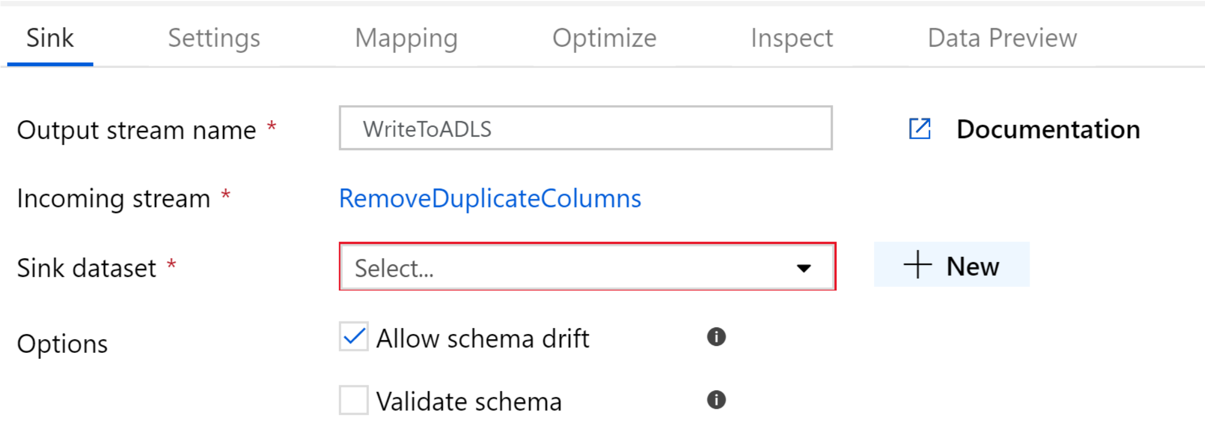
| | | | |
|---|---|---|---|
| □ | abc ProductID | → | ProductID |
| □ | abc Product | → | Product |
| □ | abc SelectProductFields@ProductCategoryID | → | ProductCategoryID |
| □ | abc SelectProductFields@ProductModelID | → | ProductModelID |
| □ | abc ProductCategory | → | ProductCategory |
| □ | abc ProductModel | → | ProductModel |

13. Finally, we want to write our data back to our lake in it's new form – we do this by adding a "Sink" transformation at the point where the data is in the right state:



We can choose an existing dataset if we had set one up in advance, or we can create one now, using the schema of our stream as reference

14. Name your sink transformation, then click the "New" button to create a new dataset to hold our data



Output stream name * WriteToADLS

Incoming stream * RemoveDuplicateColumns

Sink dataset * Select...

Options ☑ Allow schema drift

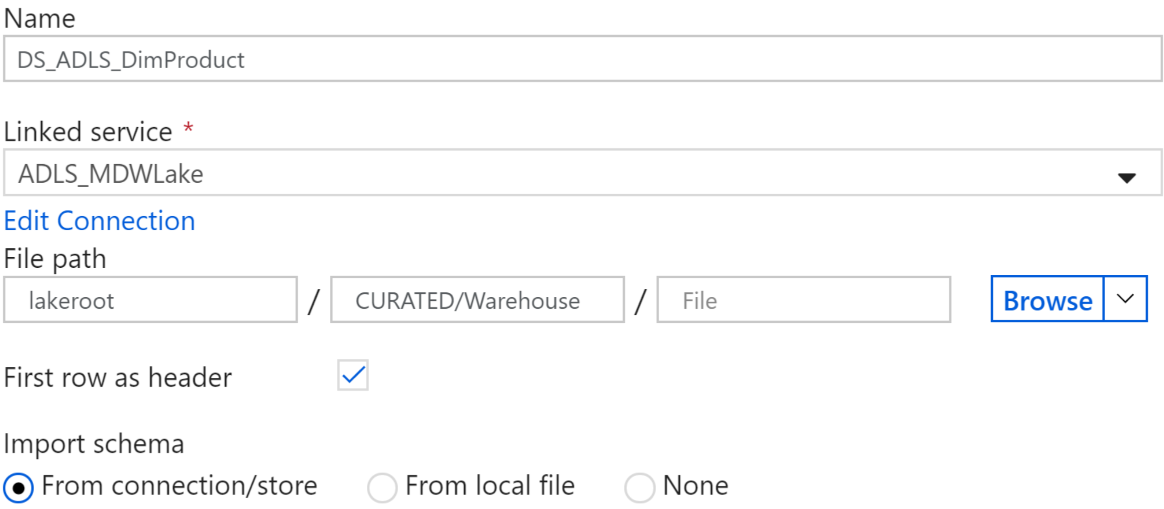
□ Validate schema

15. Select Data Lake Store Gen 2 as the destination type, then select the file format.

    For most scenarios, we would write this as a parquet file for performance – but for ease of testing our transformation, let’s leave it as a CSV

    You’ll notice that several other big data formats are shown (ORC, Avro etc) but they are not yet enabled

16. Give the file a name and configure where it should be created within your lake. By convention, I’ve separated mine from the RAW data into a CURATED data layer:



Name: DS_ADLS_DimProduct

Linked service *: ADLS_MDWLake

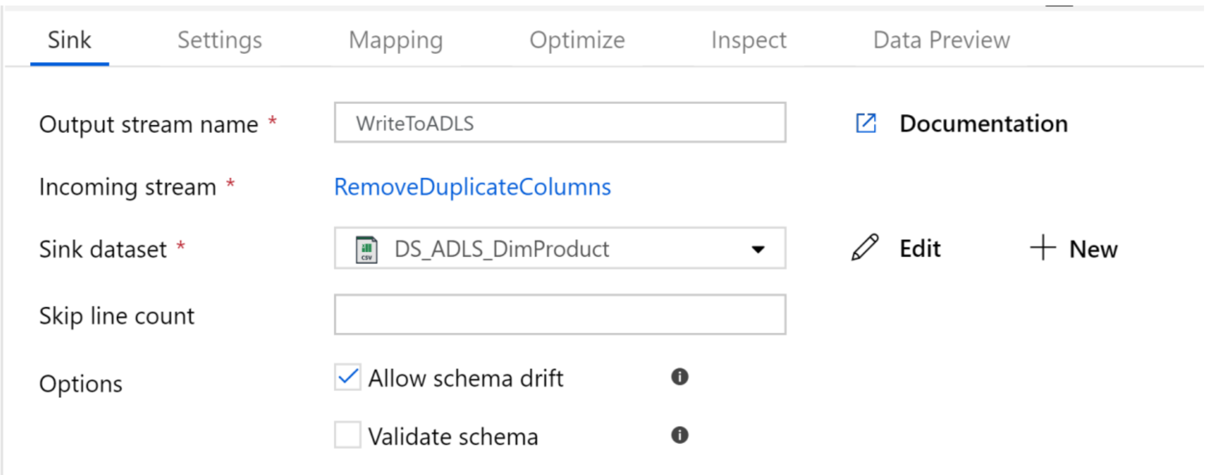
Edit Connection

File path: lakeroot / CURATED/Warehouse / File

First row as header ☑

Import schema: ● From connection/store ○ From local file ○ None

Finish creating your dataset and navigate back to your data flow, you’ll see it now updated with your sink:



Output stream name *: WriteToADLS

Incoming stream *: RemoveDuplicateColumns

Sink dataset *: DS_ADLS_DimProduct

Skip line count:

Options: ☑ Allow schema drift ☐ Validate schema

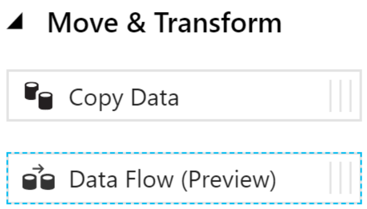
And that’s it! That’s our working Data Flow, ready to go!

# Lab 04 – Building a Mapping Data Flow

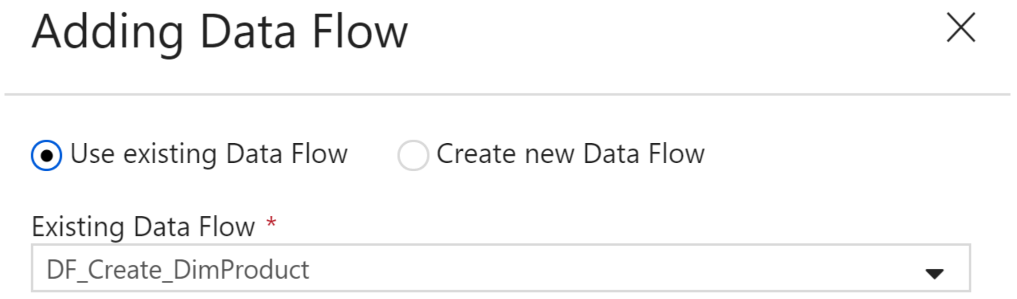
## Lab 04.C – Create a pipeline for your Data Flow

So… we've created a data flow and hooked it up to source data, destinations etc… but we've got nothing to actually run the data flow logic. That's where we need a pipeline.

1. Create a new pipeline and drag on a "Data Flow" activity, found in the "Move & Transform" menu:



2. Unlike other transformations, this will immediately open up a config window, where you'll need to select the name of the data flow you created earlier:

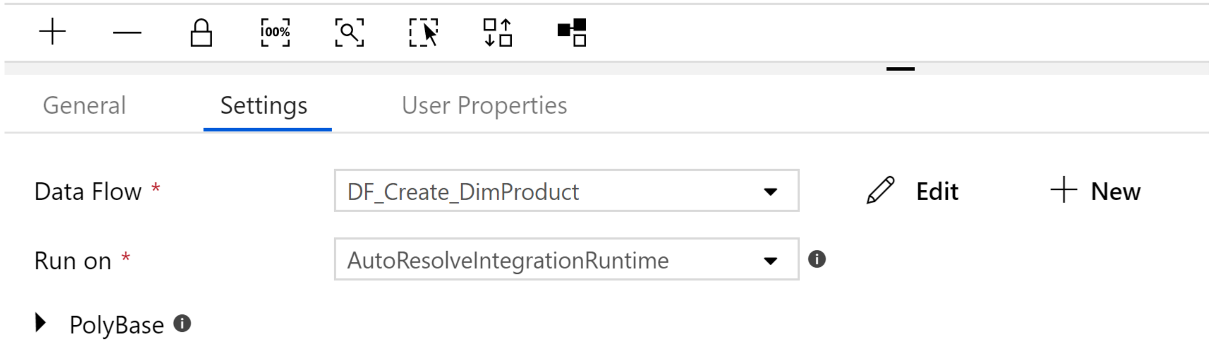


Adding Data Flow

● Use existing Data Flow ○ Create new Data Flow

Existing Data Flow *: DF_Create_DimProduct

If you look at the settings for your new activity, there isn't much to do. By default (and as the only option in preview), the data flow will work on an internal ADF Databricks cluster and perform it's own sizing

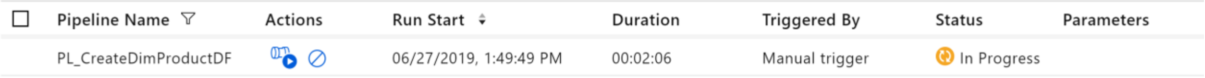


Data Flow *: DF_Create_DimProduct

Run on *: AutoResolveIntegrationRuntime

In future, it is expected that we'll be able to tweak the performance by changing the size/scale of the spark cluster our data flow is running on.

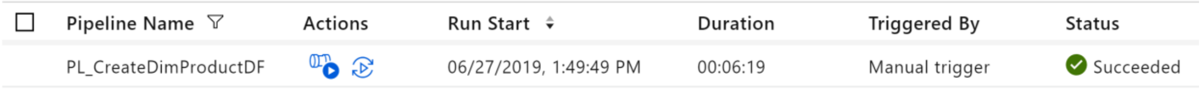
3. Now we can test our new creation! Hit the publish button and trigger your pipeline to see how it goes



| □ | Pipeline Name | Actions | Run Start | Duration | Triggered By | Status | Parameters |
|---|---|---|---|---|---|---|---|
| | PL_CreateDimProductDF | ▶ ⊘ | 06/27/2019, 1:49:49 PM | 00:02:06 | Manual trigger | In Progress | |

It'll probably take a couple of minutes before it does anything – that's because it's provisioning the spark cluster, which has an overhead. They're looking to reduce this, but for now bear in mind that these flows are generally meant for fairly large data processing tasks.

Eventually, we'll see this turn green as the transformation succeeds:

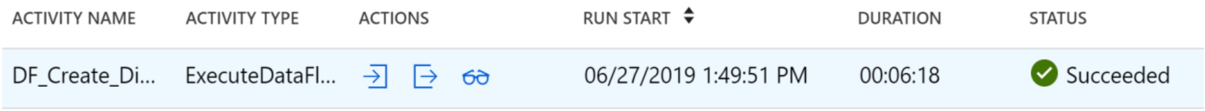


| □ | Pipeline Name | Actions | Run Start | Duration | Triggered By | Status |
|---|---|---|---|---|---|---|
| | PL_CreateDimProductDF | ▶ ⟳ | 06/27/2019, 1:49:49 PM | 00:06:19 | Manual trigger | ✓ Succeeded |

What's good to learn is the debug/audit information available

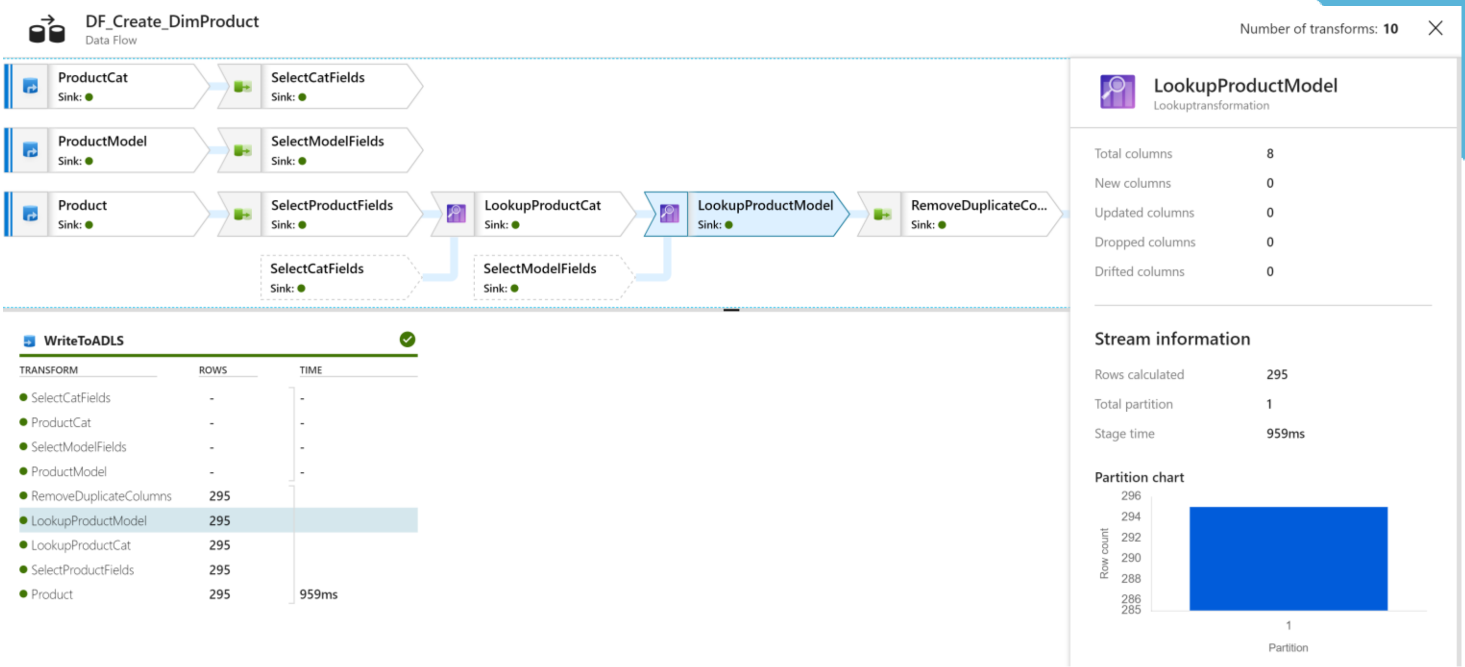
4. Click on the ▶ icon to view the activity-level results for your pipeline

   Here we can see how long each activity in our pipeline took to execute, which is useful if you have a long chain of transformations, but what we're interested in is the details behind the actual dataflow:



| ACTIVITY NAME | ACTIVITY TYPE | ACTIONS | RUN START | DURATION | STATUS |
|---|---|---|---|---|---|
| DF_Create_Di... | ExecuteDataFl... | ⇥ ⇤ 👓 | 06/27/2019 1:49:51 PM | 00:06:18 | ✓ Succeeded |

5. Click the 👓 to open the data flow results page, which breaks down each transformation and gives partitioning data as various spark transformations were made:



DF_Create_DimProduct
Data Flow

Number of transforms: 10

WriteToADLS ✓

| TRANSFORM | ROWS | TIME |
|---|---|---|
| ● SelectCatFields | - | - |
| ● ProductCat | - | - |
| ● SelectModelFields | - | - |
| ● ProductModel | - | - |
| ● RemoveDuplicateColumns | 295 | |
| ● LookupProductModel | 295 | |
| ● LookupProductCat | 295 | |
| ● SelectProductFields | 295 | |
| ● Product | 295 | 959ms |

LookupProductModel
Lookuptransformation

| | |
|---|---|
| Total columns | 8 |
| New columns | 0 |
| Updated columns | 0 |
| Dropped columns | 0 |
| Drifted columns | 0 |

Stream information

| | |
|---|---|
| Rows calculated | 295 |
| Total partition | 1 |
| Stage time | 959ms |

This page contains a wealth of information – highlight different transformations to see the number of rows processed, how long that stage lasted and how the data was partitioned. In this example, the data was very small and so we could perform everything on a single box. For larger examples, we can configure how datasets are distributed to optimise spark executor partitioning, which is very powerful indeed!

You've now got the basics for creating a Data Factory data flow, but there's a lot more to learn! Try out some of the other transformation types and, when you're ready, try using the DerivedColumn transformation to see the large number of functions available!