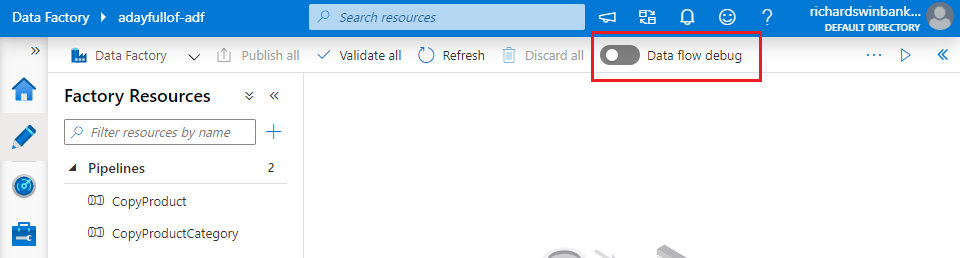
Lab 4 – Build a Mapping Data Flow

In this lab you will use Azure Data Factory’s Mapping Data Flows feature to implement a familiar data warehousing process: maintaining a dimension.

# Lab 4.1 – Enable data flow debugging

Mapping data flows are executed – and debugged – on Apache Spark clusters. A cluster takes several minutes to warm up, so start this lab by switching data flow debug on for your ADF UX session.

1. In the ADF UX, toggle the “Data flow debug” slider to “On”.



1. When the ADF UX prompts you for confirmation, click “OK”.

While the debug cluster is warming up, continue with Labs 4.2 & 4.3.

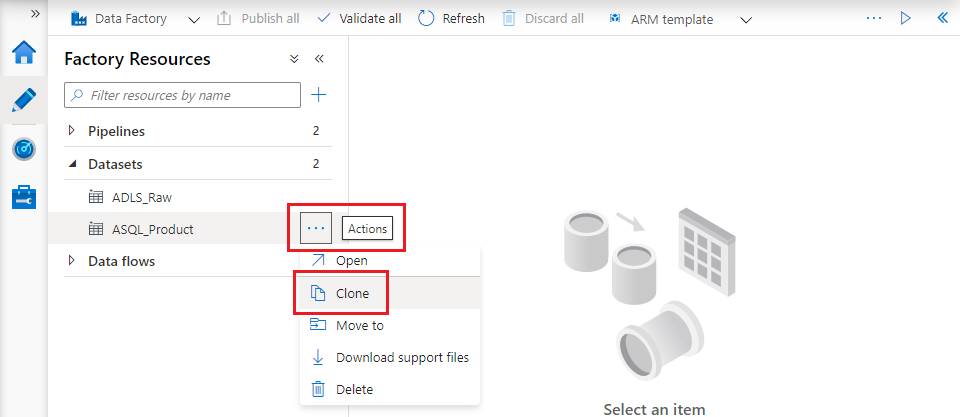
# Lab 4.2 – Copy source data to the data lake

The product dimension will be built using data from three source tables:

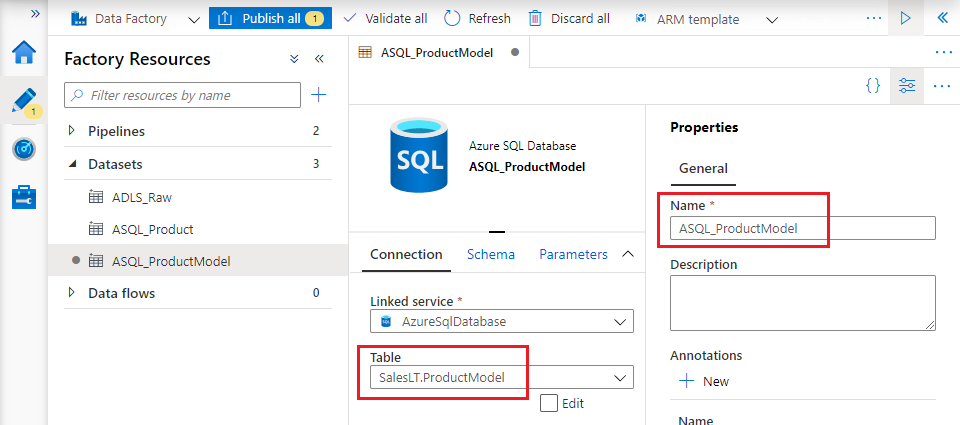
* [SalesLT].[Product]
* [SalesLT].[ProductCategory]
* [SalesLT].[ProductModel]

In Labs 2 & 3 you imported data from the first two tables into the data lake. Import data for [SalesLT].[ProductModel] now.

1. Create a copy of the “ASQL\_Product” dataset by clicking its ellipsis “Actions” button in the “Factory Resources” list, then selecting “Clone”.



1. The cloned dataset opens automatically with its “Properties” pane displayed. Change its name to “ASQL\_ProductModel”, then on the “Connections” tab choose the corresponding [AdventureWorks] table.

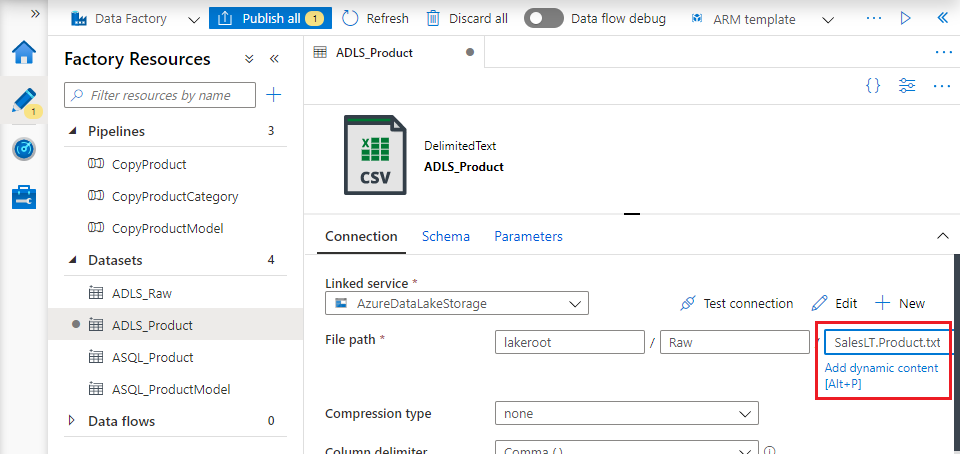


1. Create a new pipeline in the same way as Lab 2.3, using the new “ASQL\_ProductModel” dataset as source and your Azure Data Lake Storage dataset as sink. Save your changes.
2. Run the pipeline in debug mode and verify that file “SalesLT.ProductModel.txt” has been created in the “lakeroot” container’s “Raw” folder.

# Lab 4.3 – Create ADLS datasets

To use data from the three files now created in “/lakeroot/Raw”, you need datasets to represent them.

1. Create a new dataset by cloning your existing Azure Data Lake Storage dataset. Name it “ADLS\_Product” and add file name “SalesLT.Product.txt” to the “File path” specified on the dataset’s “Connection” tab.



1. Repeat step 1 to create two further datasets:
   * “ASQL\_ProductCategory”, to represent file “/lakeroot/Raw/SalesLT.ProductCategory.txt”
   * “ASQL\_ProductModel”, to represent file “/lakeroot/Raw/SalesLT.ProductModel.txt”

# Lab 4.4 – Combine ADLS datasets

The product dimension combines product model and category information to support different aggregations of facts that have a product attribute. This SQL query combines this information within the [AdventureWorks] database:

SELECT

p.ProductID

, p.[Name] AS Product

, pm.[Name] AS ProductModel

, pc.[Name] AS ProductCategory

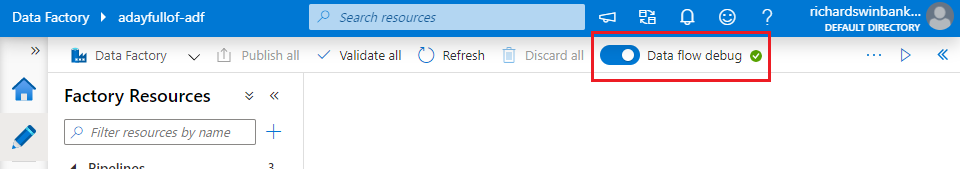
FROM SalesLT.Product p

INNER JOIN SalesLT.ProductModel pm ON pm.ProductModelID = p.ProductModelID

INNER JOIN SalesLT.ProductCategory pc ON pc.ProductCategoryID = p.ProductCategoryID

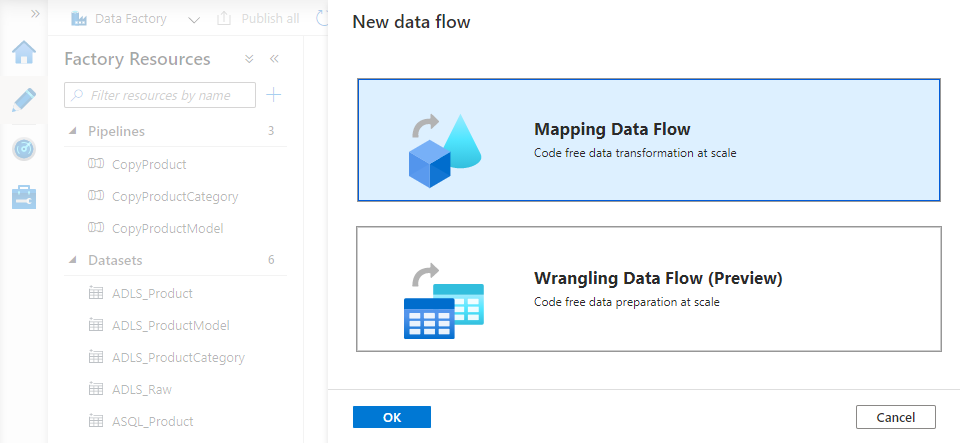
In this section you will use a Mapping Data Flow to reproduce this effect in Azure Data Factory.

1. Check that the debug cluster has finished warming up. When the cluster is available, a tick mark in a green circle appears to the right of the “Data flow debug” slider.

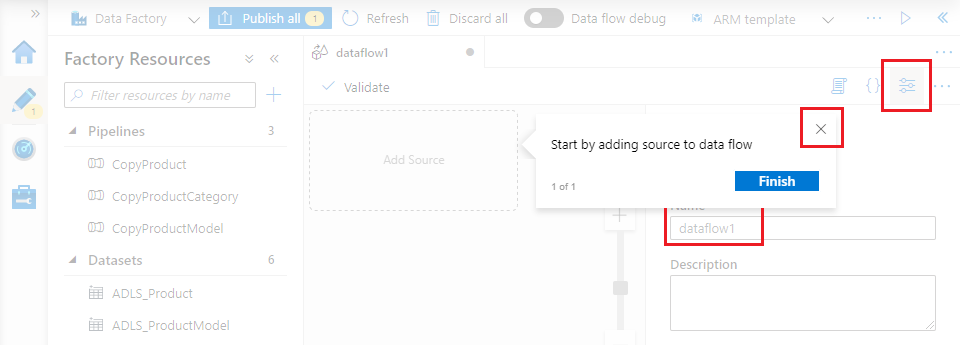


If the cluster is not ready yet, wait for it to finish warming up.

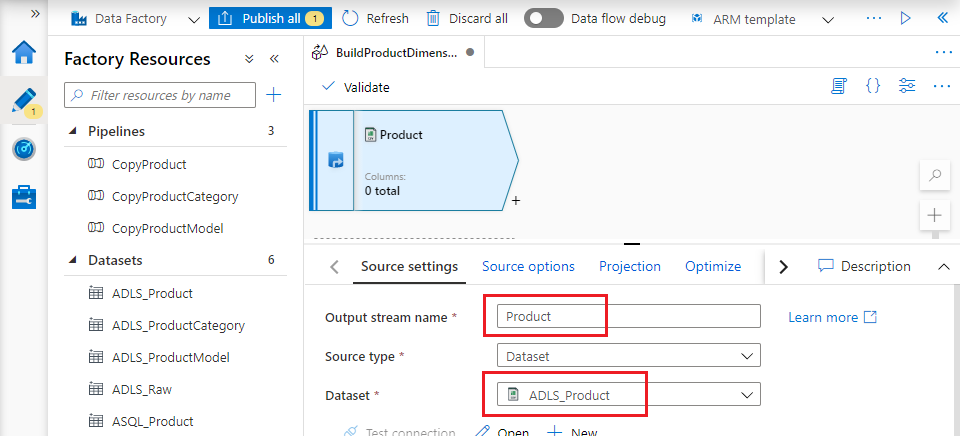
1. In the “Factory resources” sidebar, click the “+” button to the right of “Filter resources by name”, then choose “Data flow”.
2. On the “New data flow” blade, select the “Mapping Data Flow” tile. Click “OK”.



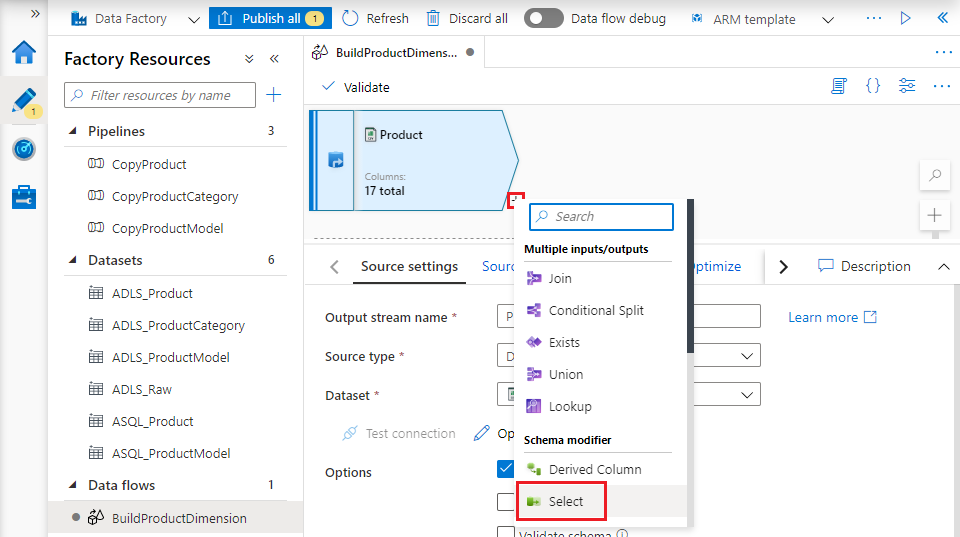
1. The data flow canvas opens, displaying the callout “Start by adding source to data flow”. Dismiss the callout using its close button, then replace the data flow default name (“dataflow1”) with something more descriptive. Use the “Properties” slider button to close the data flow properties blade.



1. Click the “Add source” tile on the data flow canvas and close the callout that appears.
   * On the source transformation’s **Source settings** tab, change its “Output stream name” to “Product” and select the corresponding “ADLS\_Product” dataset.
   * On the **Projection** tab, click “Import projection” to infer and import the source file’s schema

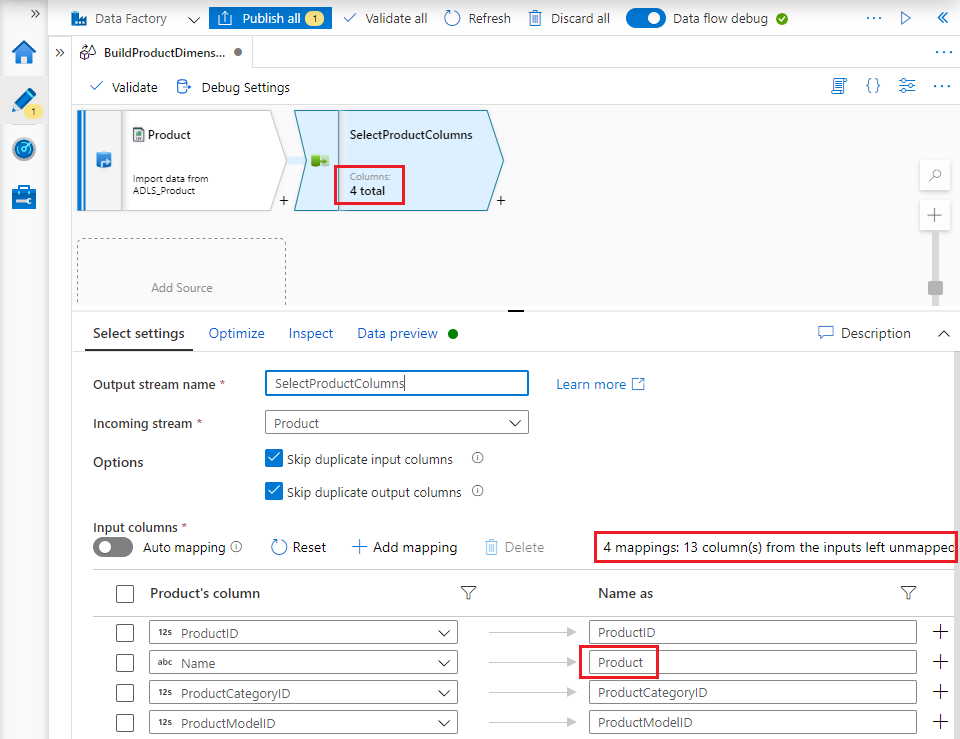


1. The source transformation provides a stream of rows for consumption by downstream transformations. To add a transformation to consume the source stream, click on the small “+” button to the bottom right of the source transformation. Choose the “Select” transformation from the popup menu of available options.

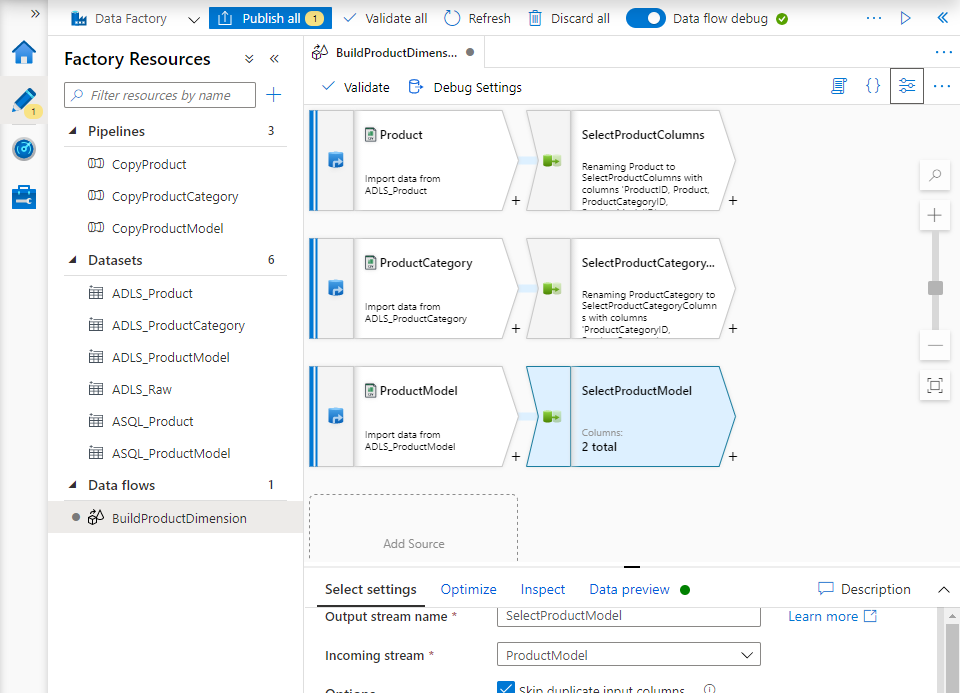


1. On the Select transformation’s **Select settings** tab, change its “Output stream name” to “SelectProductColumns”, then scroll down to the “Input columns” section.

The Select transformation enables you to rename, reorder or remove columns from a stream. Remove all columns except “ProductID”, “Name”, “ProductCategoryID” and “ProductModelID”. Rename the “Name” column by setting its “Name as” value to “Product”.

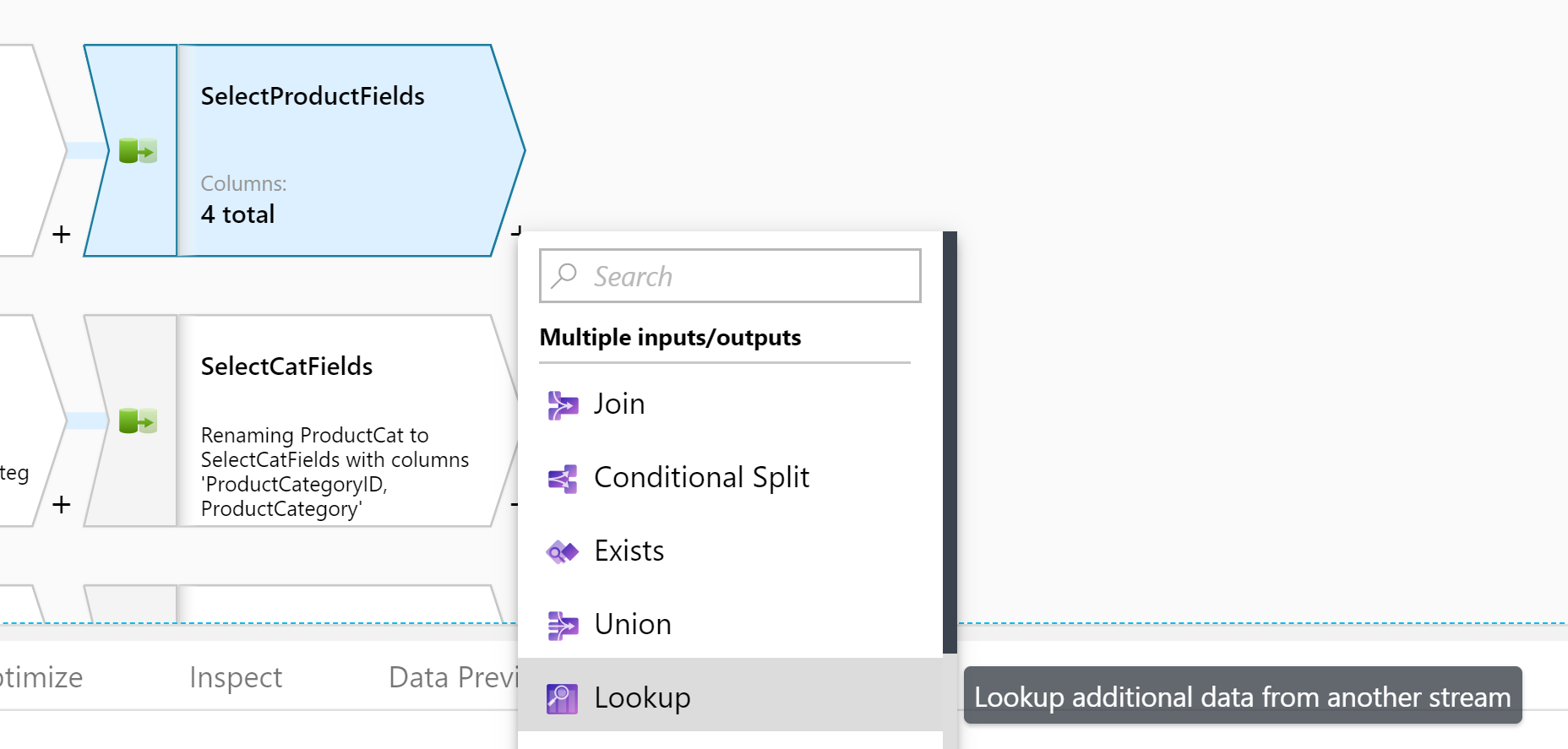


1. Repeat steps 5-7 for the “ADLS\_ProductCategory” dataset:
   * Add a source (using the “Add Source” tile displayed on the data flow canvas beneath the Product source transformation)
   * Set its dataset to “ADLS\_ProductCategory” and import the file’s schema
   * Add a select transformation
   * Remove all columns except “ProductCategoryID” and “Name”. Rename “Name” to “ProductCategory”.
2. Repeat steps 5-7 using the “ADLS\_ProductModel” dataset. Remove all columns except “ProductModelID” and “Name”. Rename “Name” to “ProductModel”. You will now have three parallel streams, loading and modifying data from the three source files.

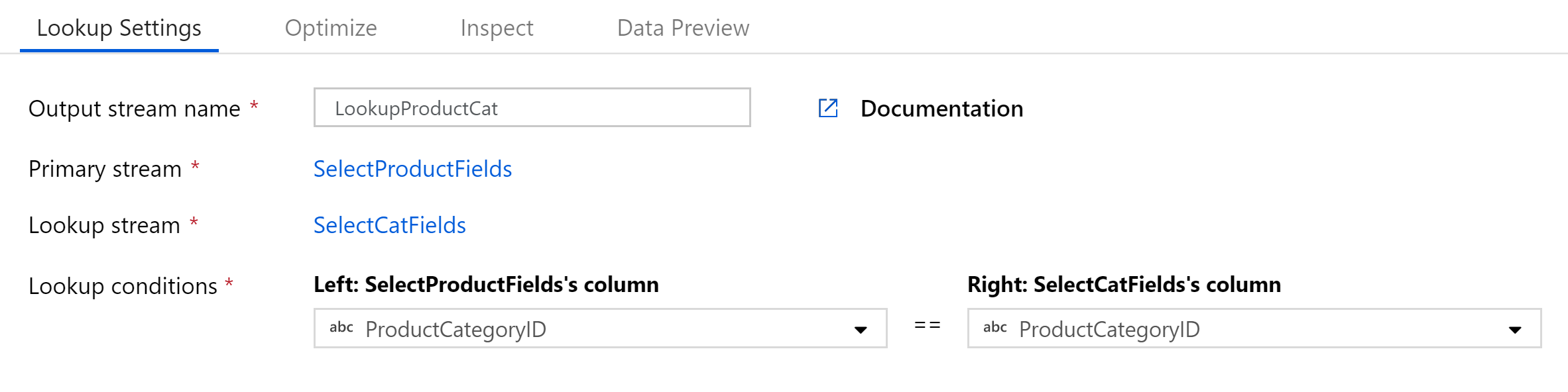


We now have our source data, nicely trimmed of excess fields that we can combine into a single dataset.

1. 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

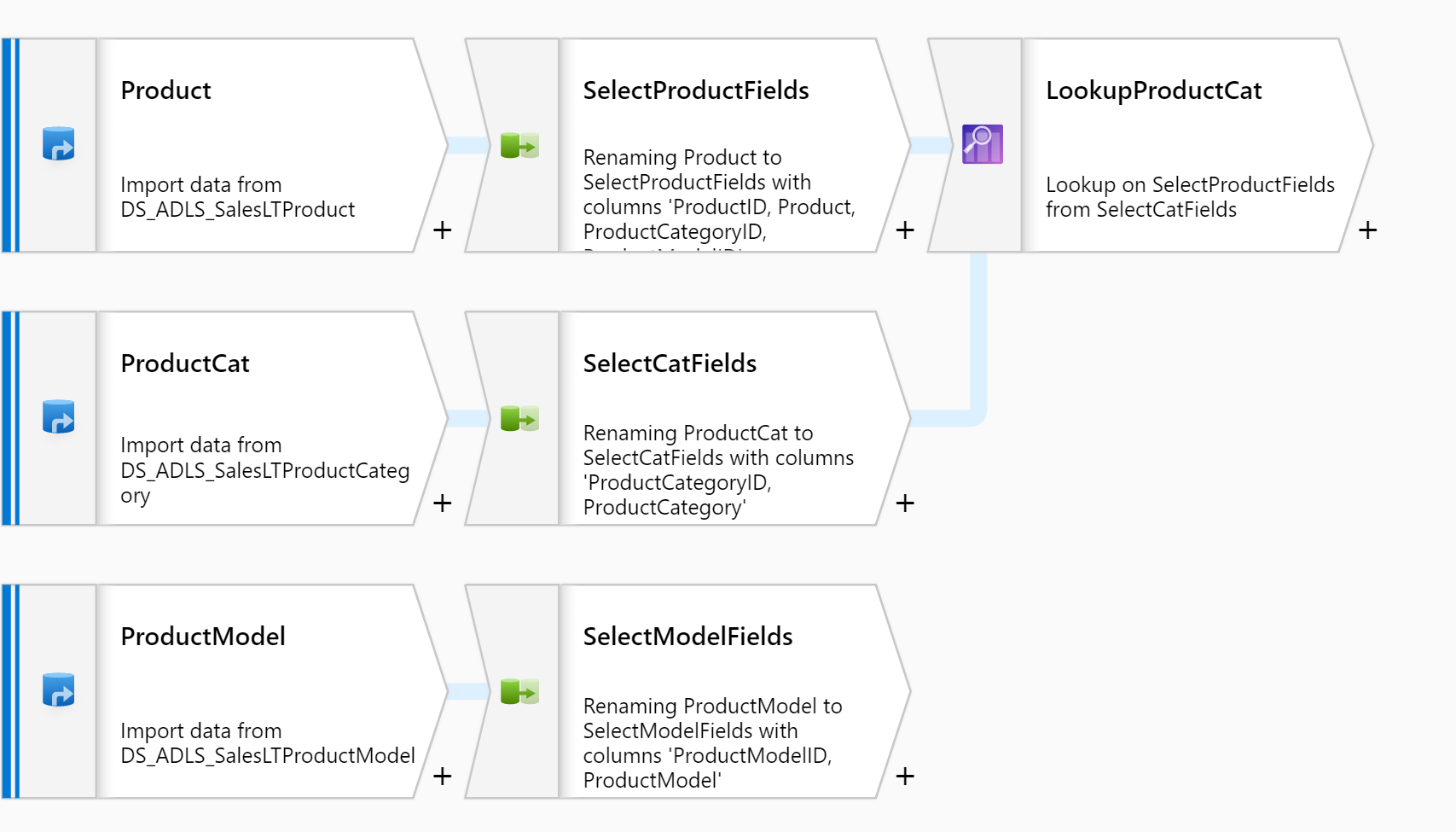


1. Select the “SelectCatFields” stream as the reference stream and configure it to use the ProductCategoryID to perform the reference join, like so:

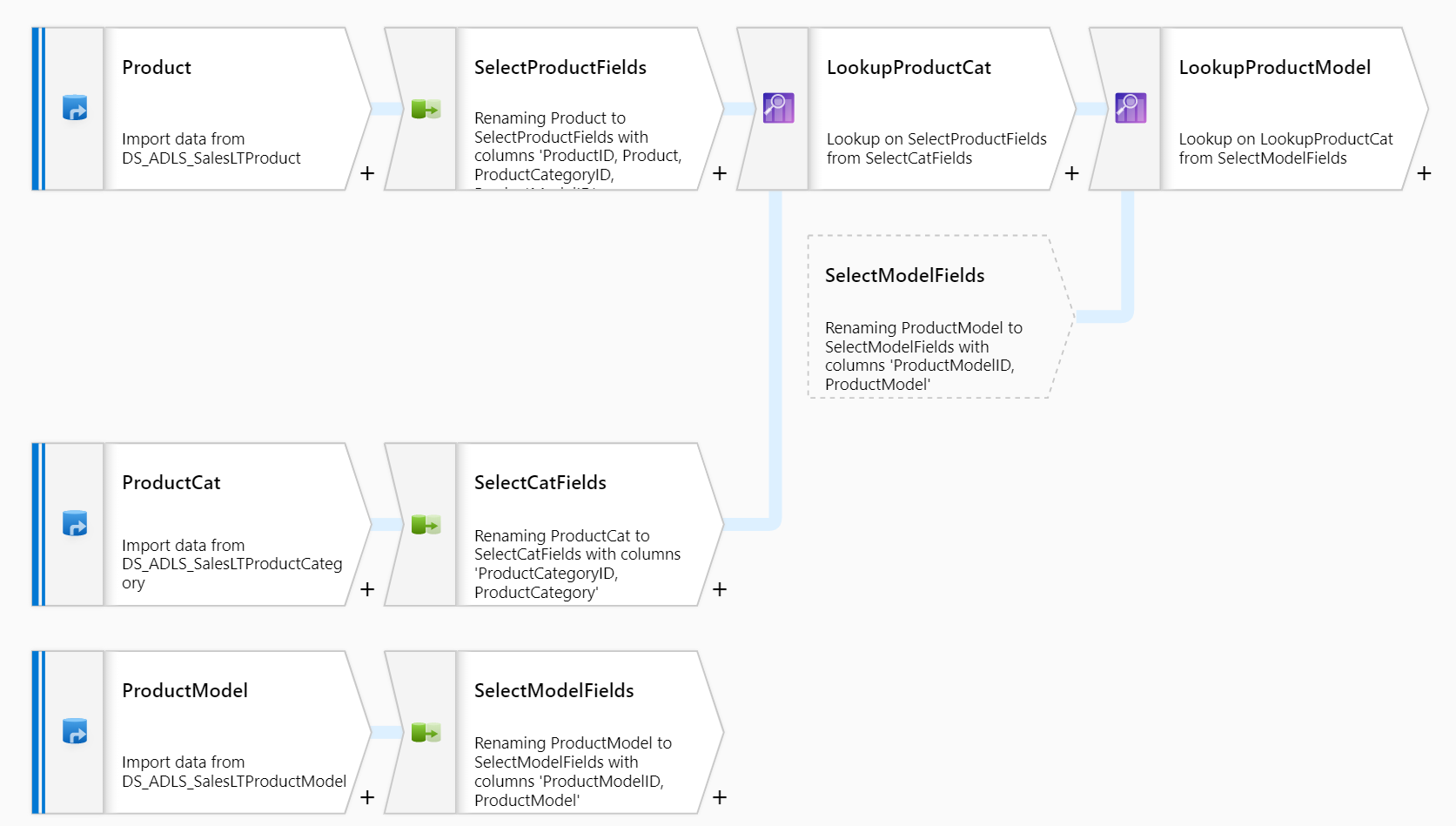


Be careful here – you can actually select to use the output of any of the transformations, including the original source ones, before we had stripped columns and renamed them.

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

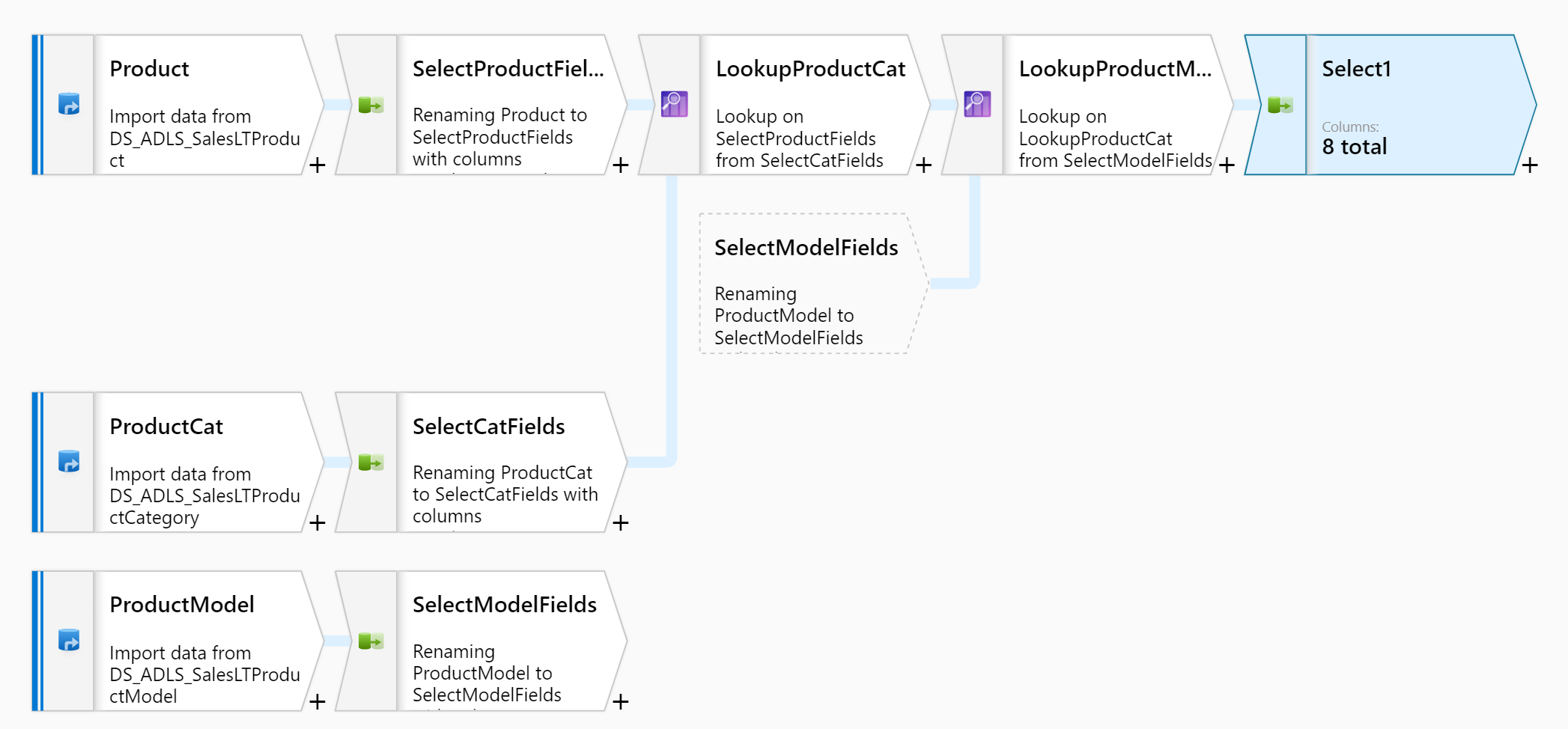


Do the same for the Product Model output stream:

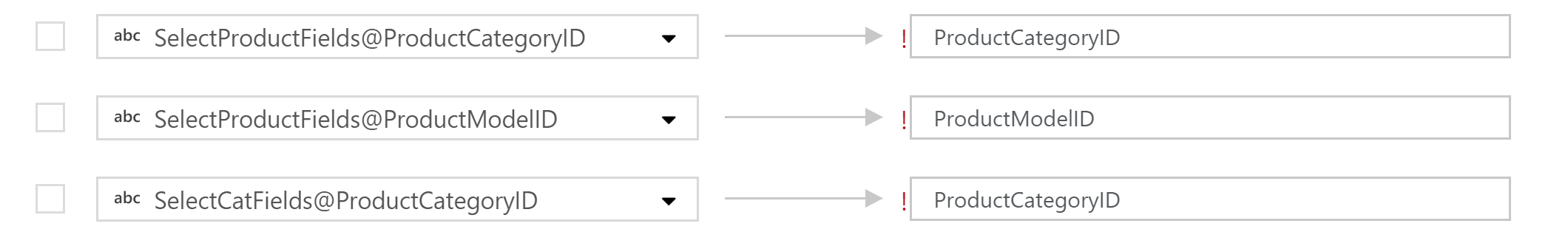


You may notice, we didn’t select the fields to be added at any point, just those that are the join constraints. By default, it will bring the whole table into the aggregation, including duplicates of the keys.

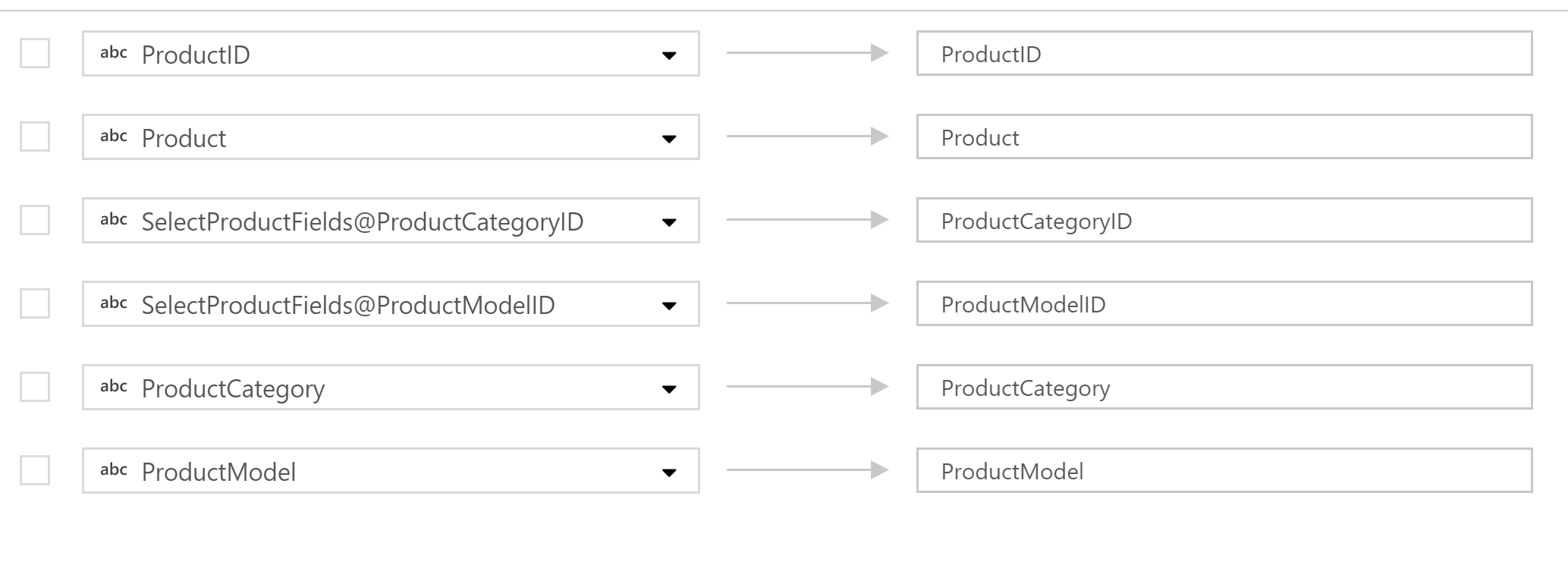
1. Let’s add a final select transformation to get rid of those duplicate keys:



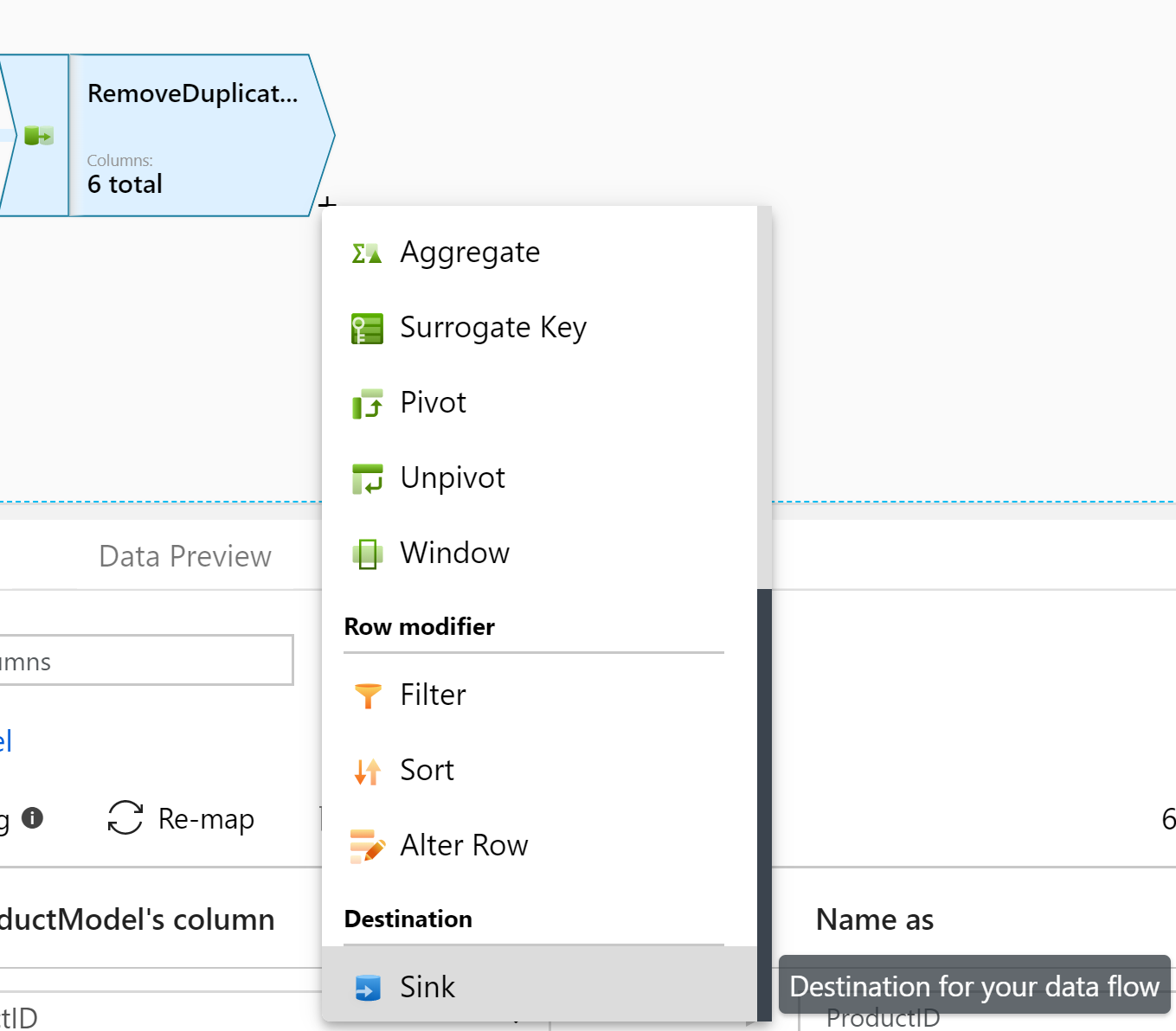
The duplicate columns will be highlighted and use a SELECT syntax to denote which stream they orginally came from:



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

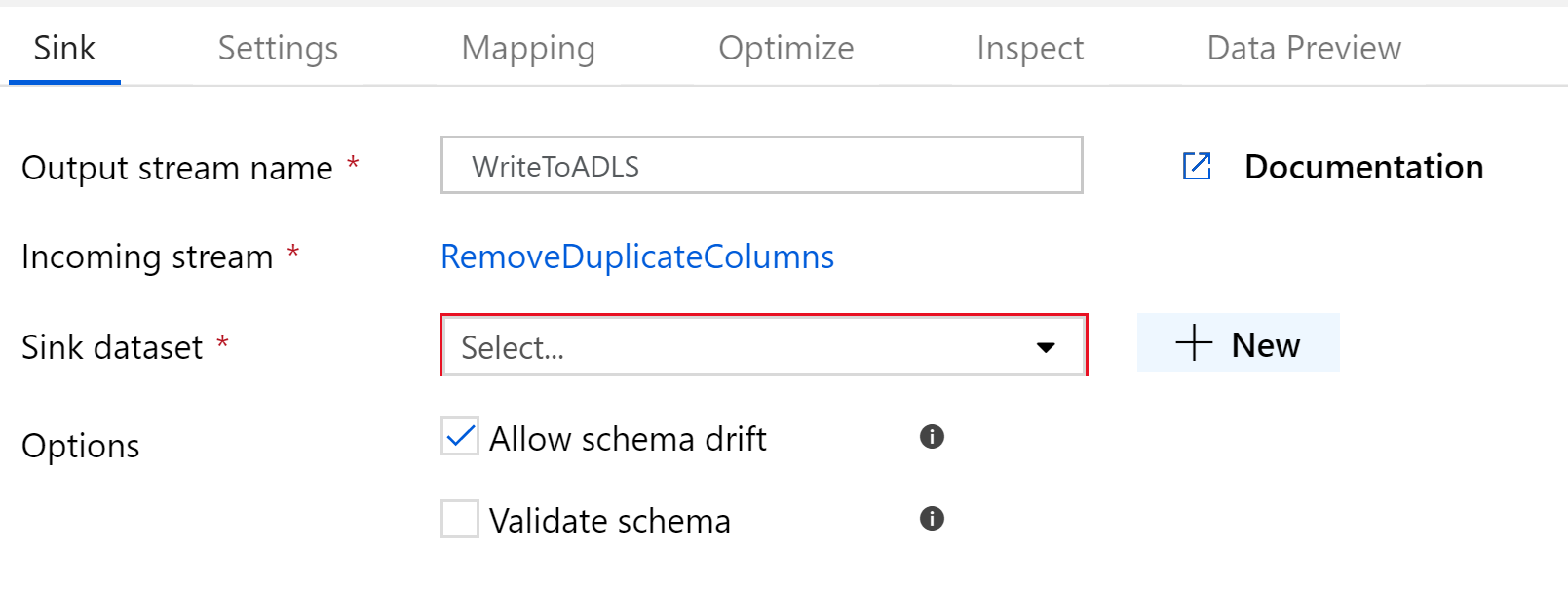


1. Finally, we want to write our data back to our lake in it’s new form – we do this by adding a “Sink” transformation at the point where the data is in the right state:



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

1. Name your sink transformation, then click the “New” button to create a new dataset to hold our data

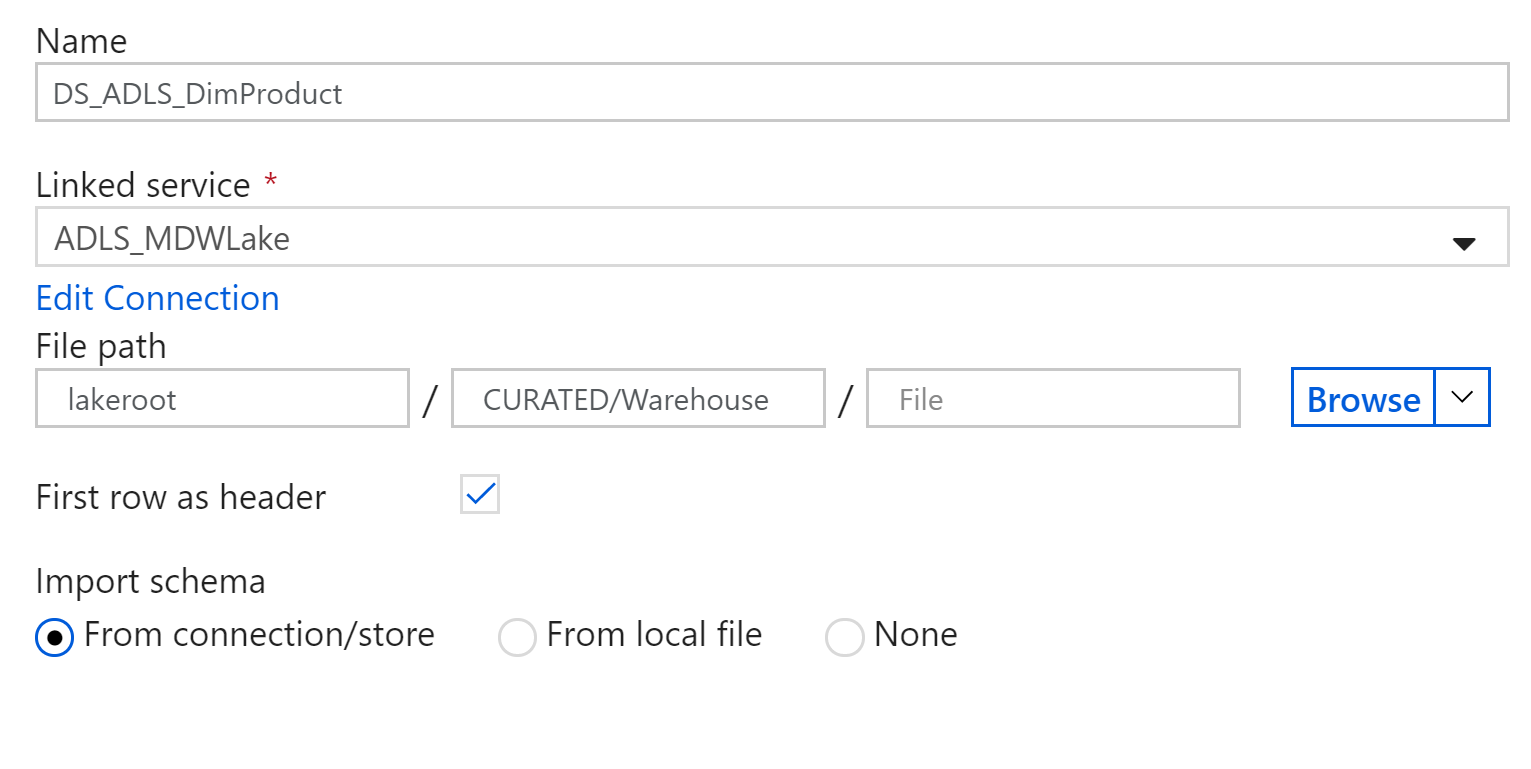


1. Select Data Lake Store Gen 2 as the destination type, then select the file format.

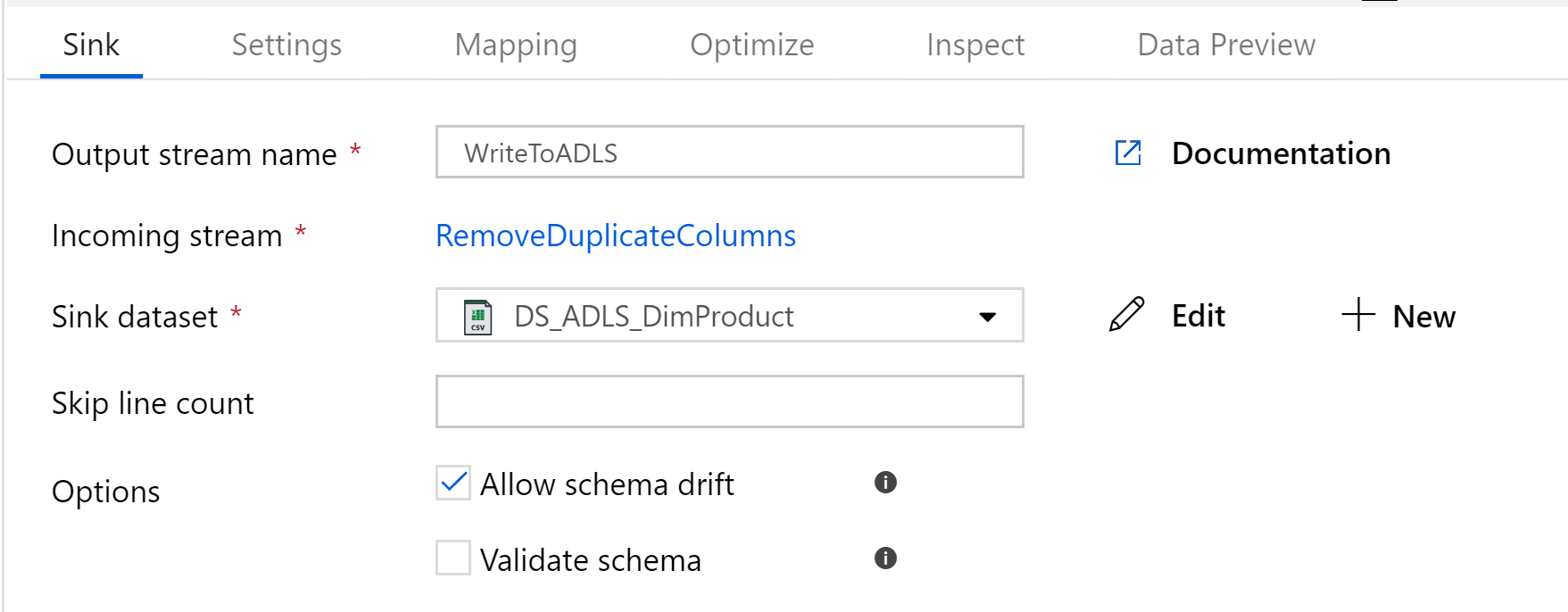
For most scenarios, we would write this as a parquet file for performance – but for ease of testing our transformation, let’s leave it as a CSV

You’ll notice that several other big data formats are shown (ORC, Avro etc) but not all of them are available yet.

1. 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:



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

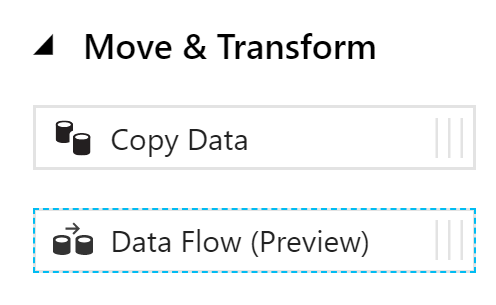


And that’s it! That’s our working Data Flow, ready to go!

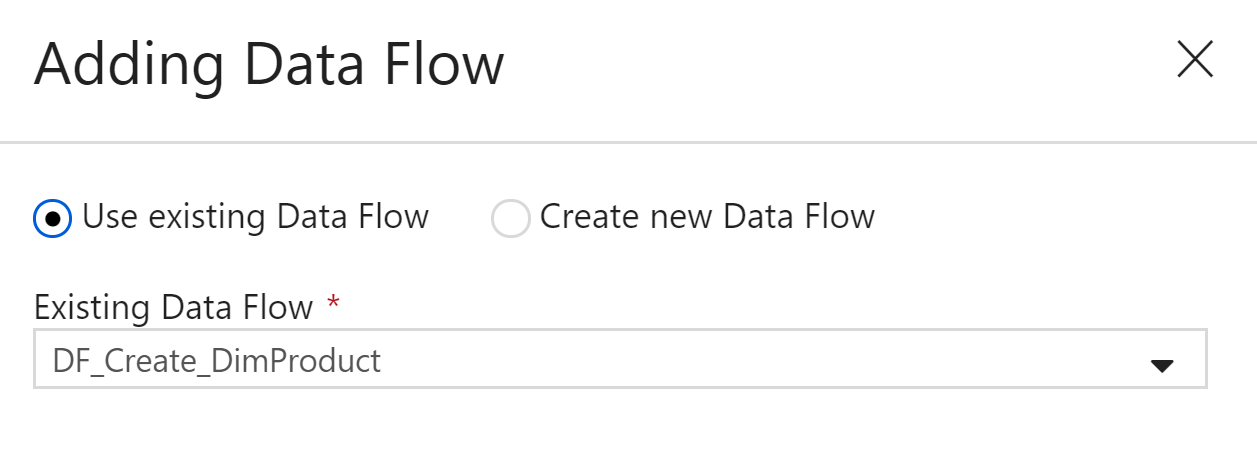
## Lab 04.C – Create a pipeline for your Data Flow

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.

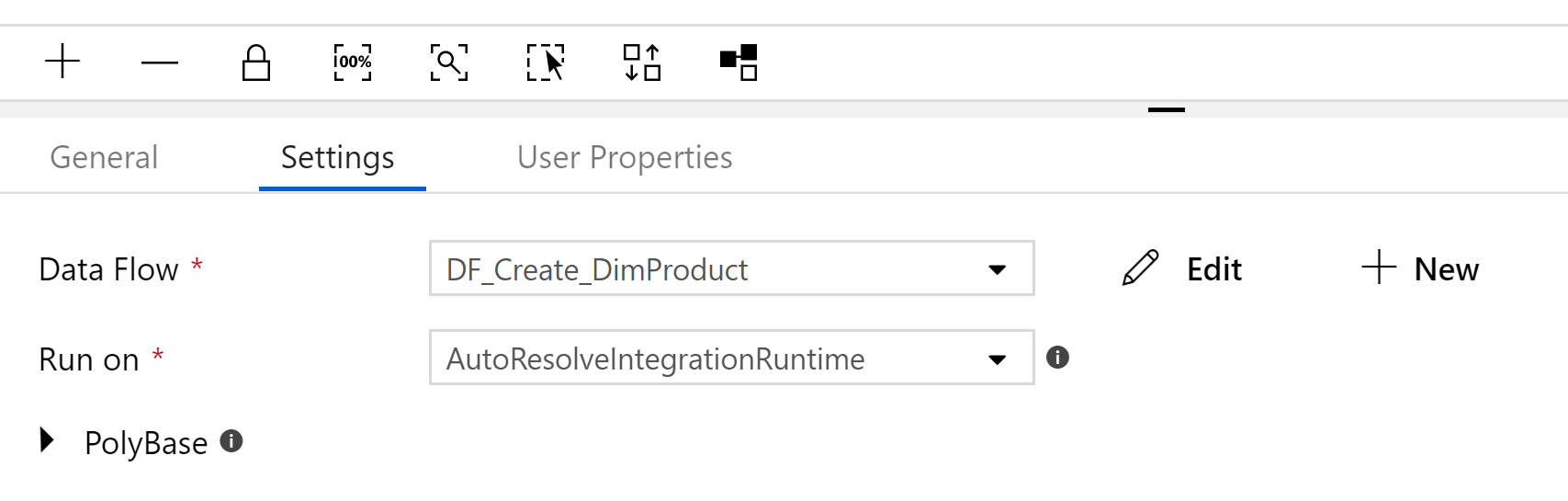
1. Create a new pipeline and drag on a “Data Flow” activity, found in the “Move & Transform” menu:



1. 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:

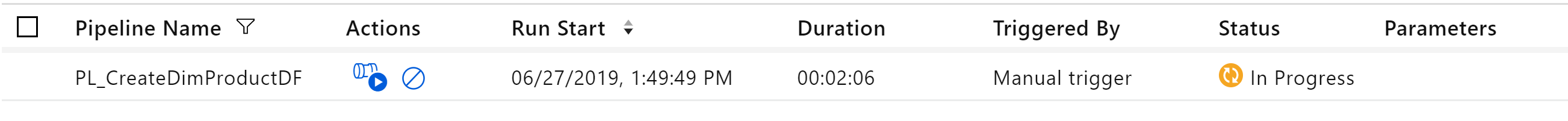


If you look at the settings for your new activity, there isn’t much to do. By default (and as the only option in preview), the data flow will work on an internal ADF Databricks cluster and perform it’s own sizing



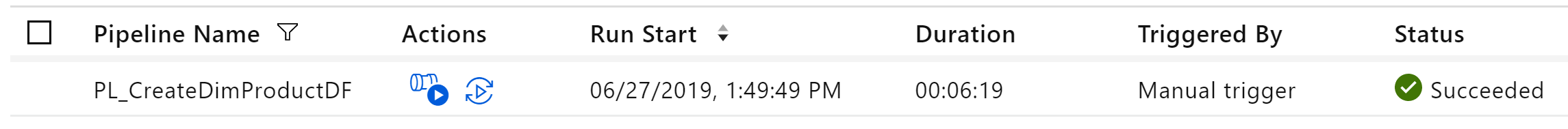
In future, it is expected that we’ll be able to tweak the performance by changing the size/scale of the spark cluster our data flow is running on.

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

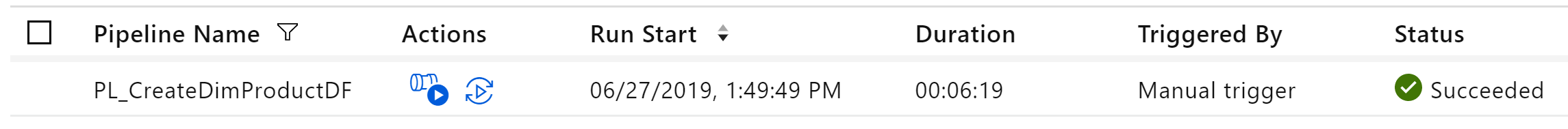


It’ll probably take a couple of minutes before it does anything – that’s because it’s provisioning the spark cluster, which has an overhead. They’re looking to reduce this, but for now bear in mind that these flows are generally meant for fairly large data processing tasks.

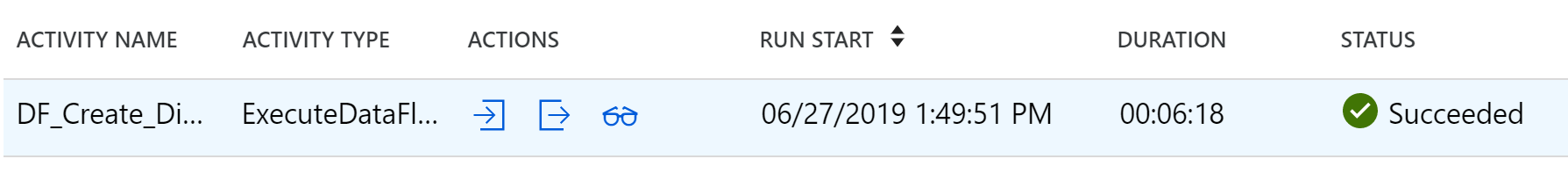
Eventually, we’ll see this turn green as the transformation succeeds:

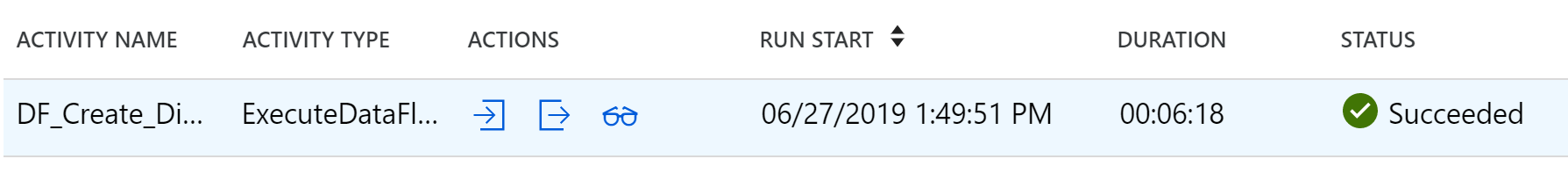


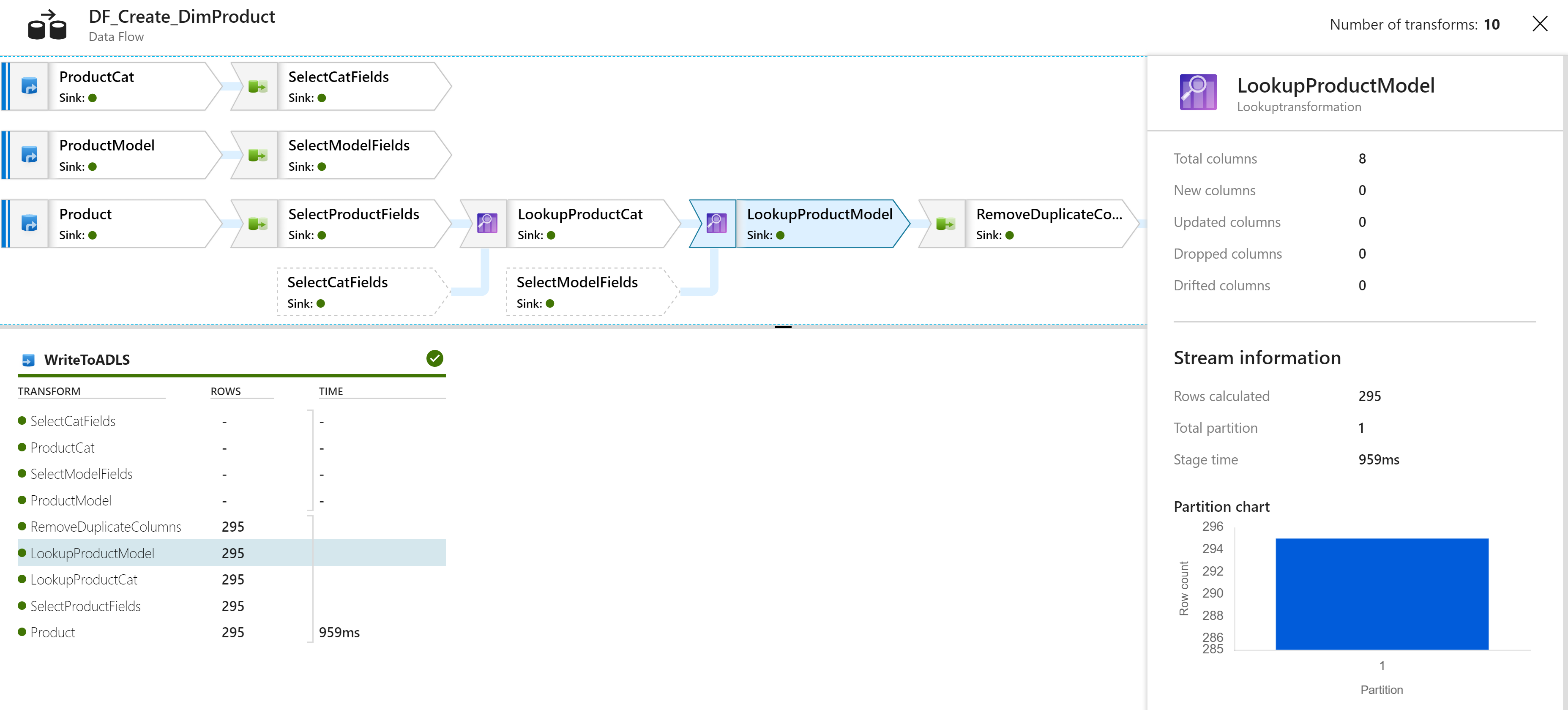
What’s good to learn is the debug/audit information available

1. 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:



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



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!