Lab 4 – Build a Mapping Data Flow

In this lab…

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

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

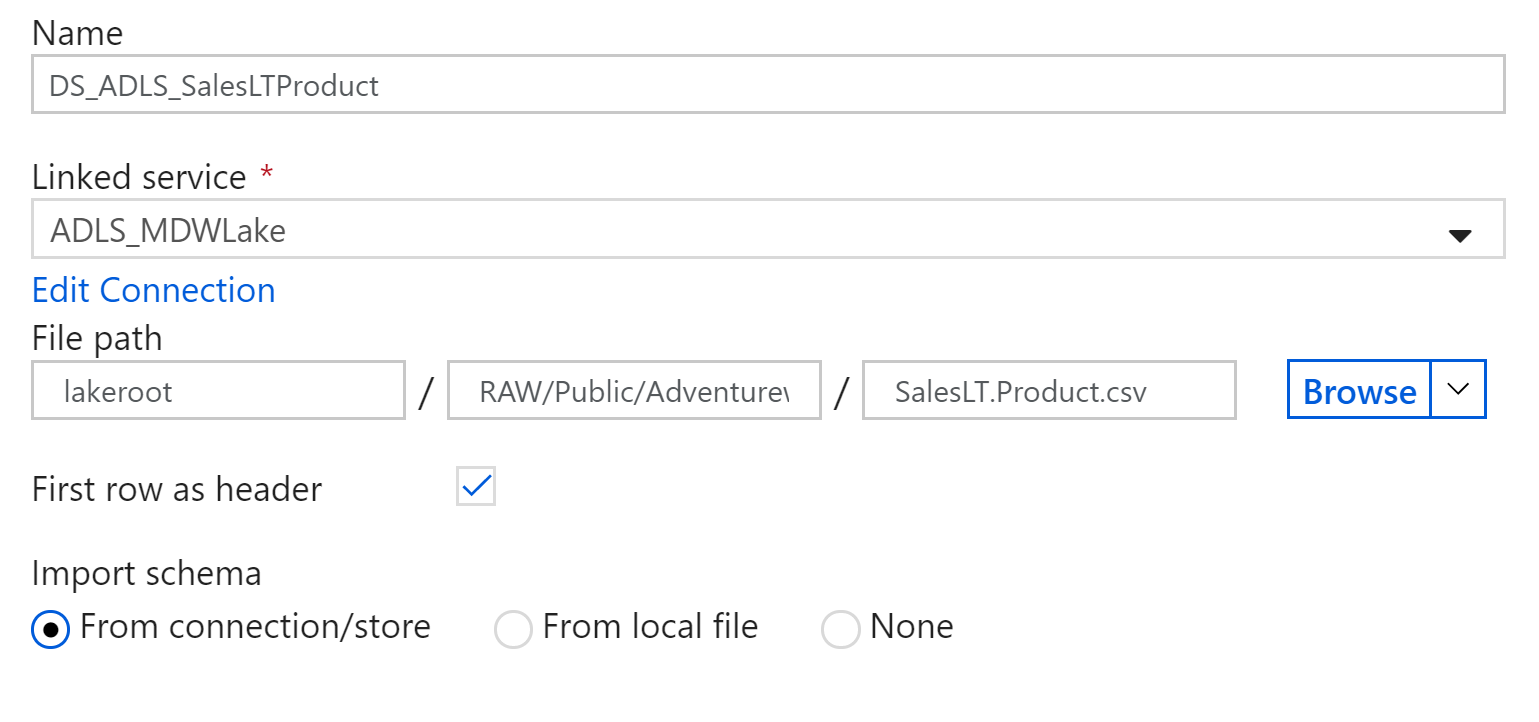
from SalesLT.Product P

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

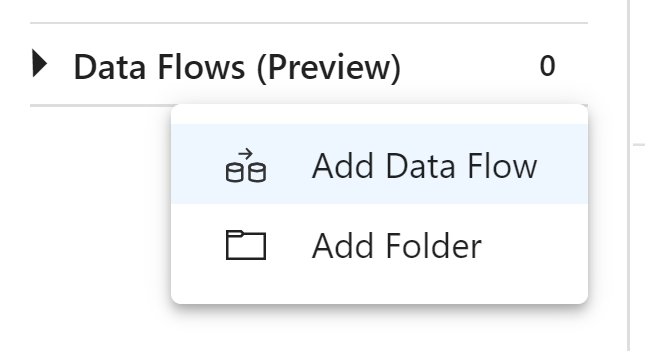
1. Just as we have before, we need to create three datasets pointing to files within our late. Create a Data Lake Store Gen 2 dataset, for a CSV file and select the Product tables we imported earlier:



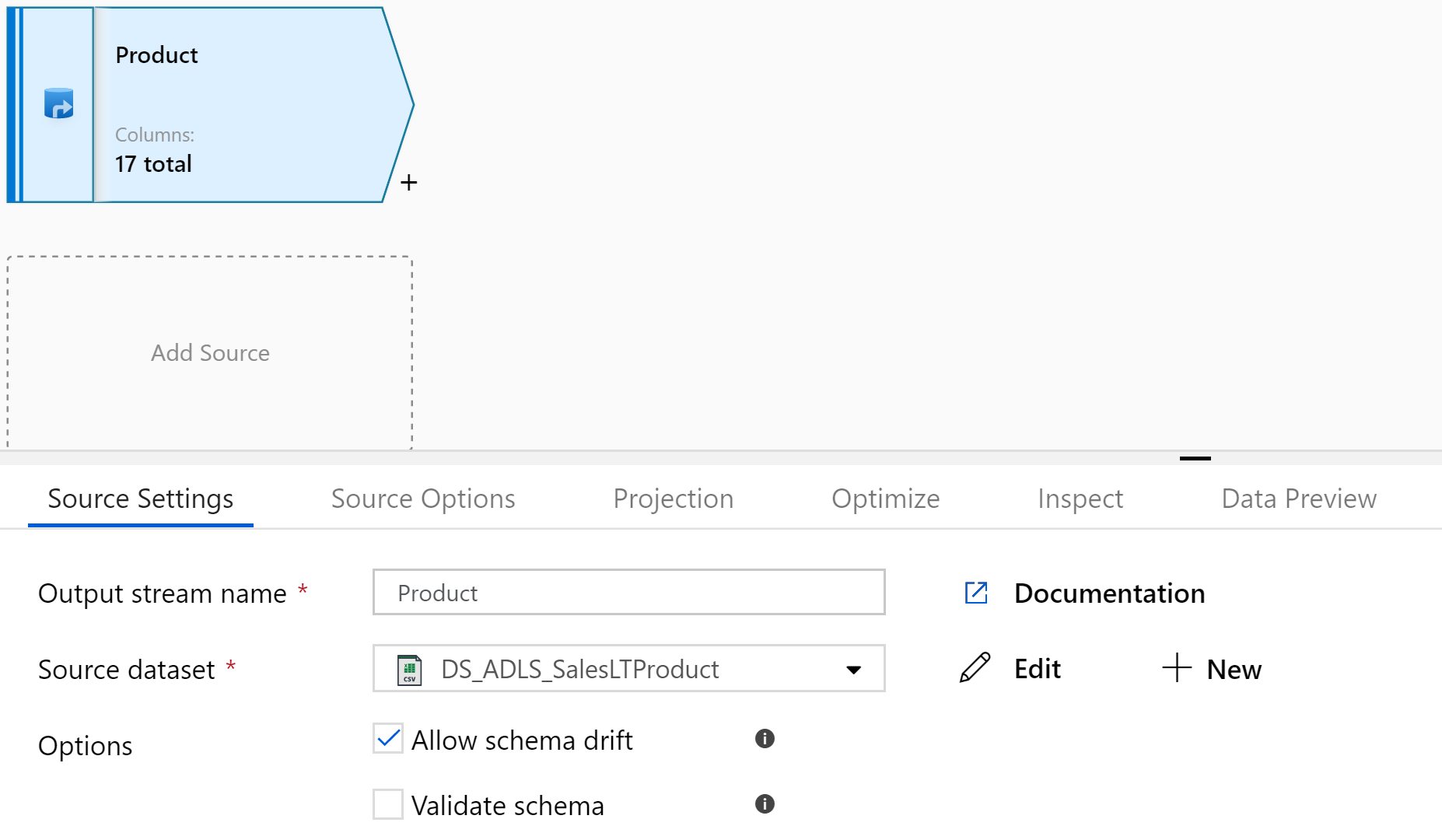
1. Repeat the above steps for ProductCategory and ProductModel

## LAB 04.B – Create a Data Flow

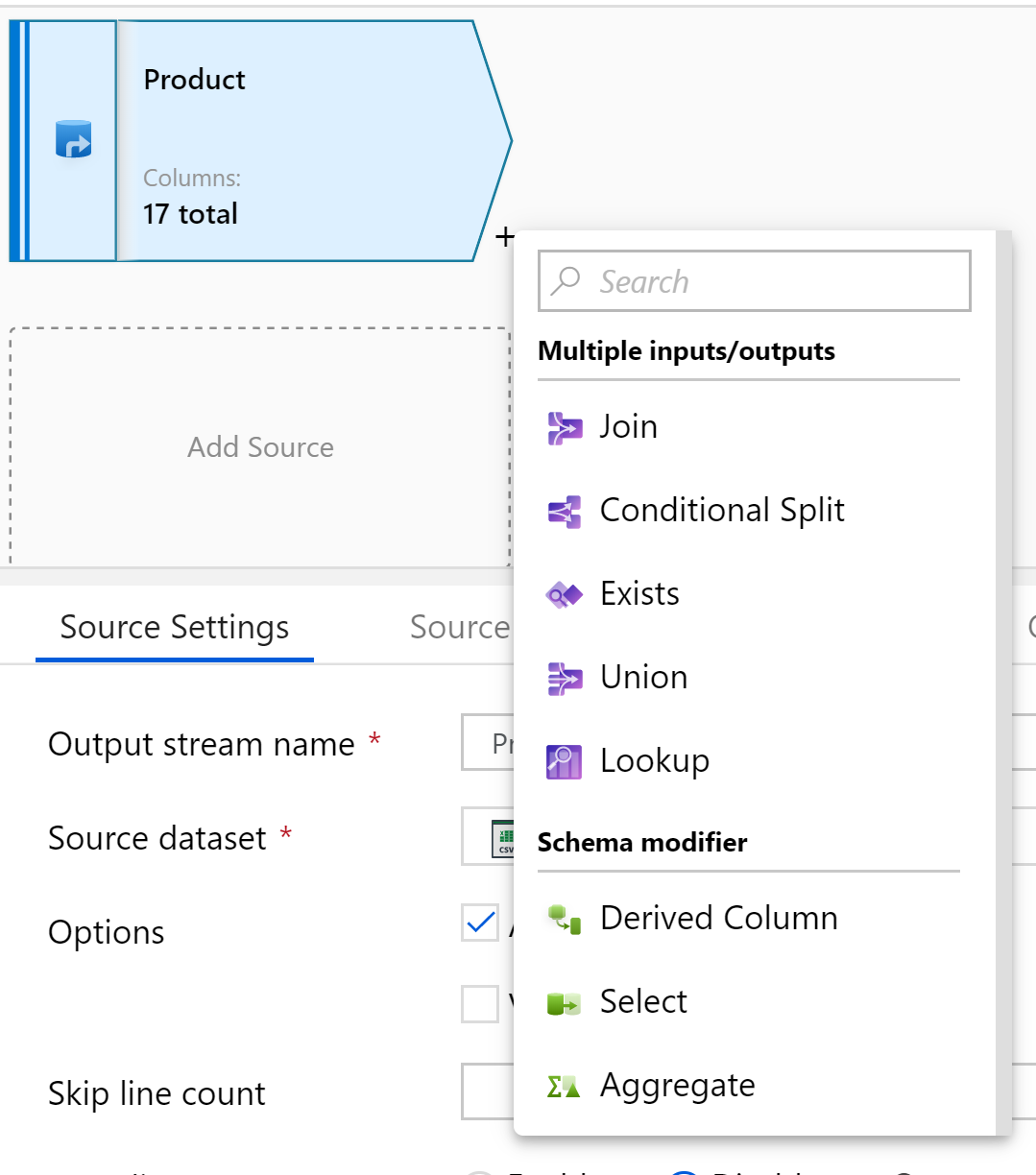
1. Create a new Data Flow



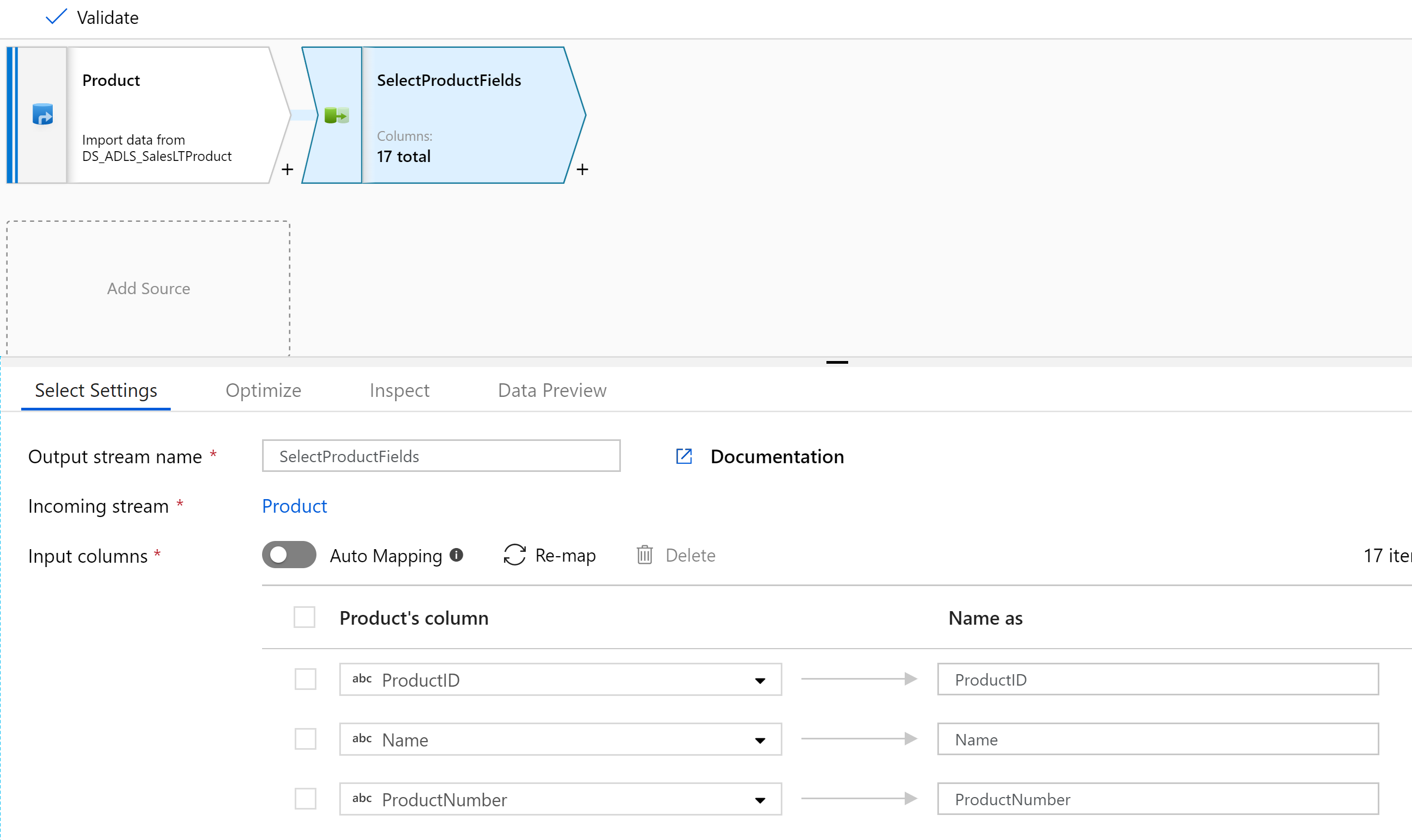
1. You’ll be guided to add a source – click on the empty square and it will create a “source” stream component. Name this “Product” and select the DS\_ADLS\_SalesLTProduct dataset



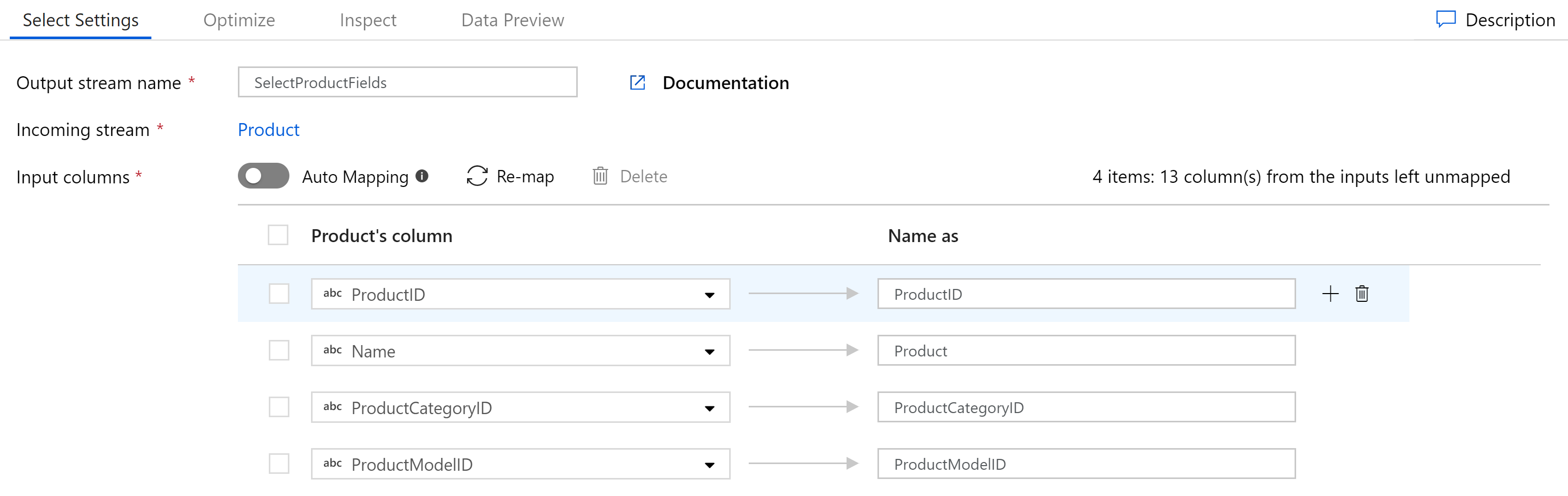
1. Click the small + icon next to the source stream to add a transformation:



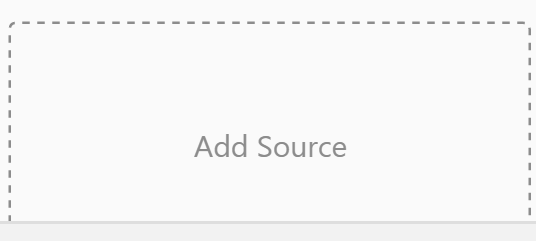
1. We want to get rid of excess columns, so let’s use a “Select” transformation. Give the transformation step a name:



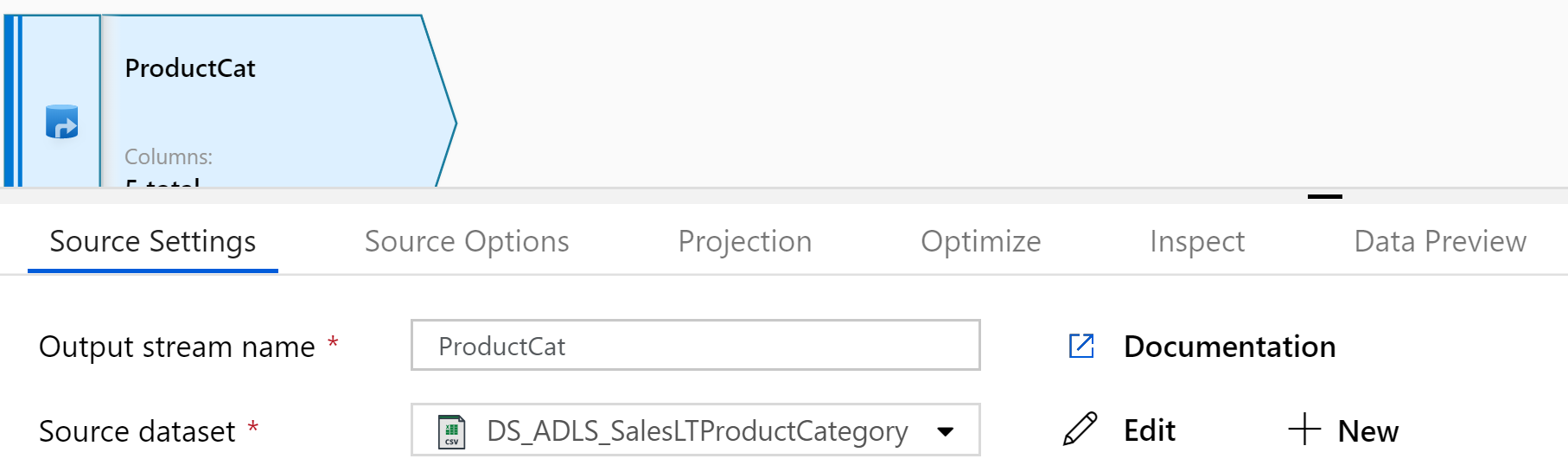
1. We want to rename “Name” to “Product” and remove all columns except for ProductID, ProductCategoryID, ProductModelID and our renamed “Product” attribute.



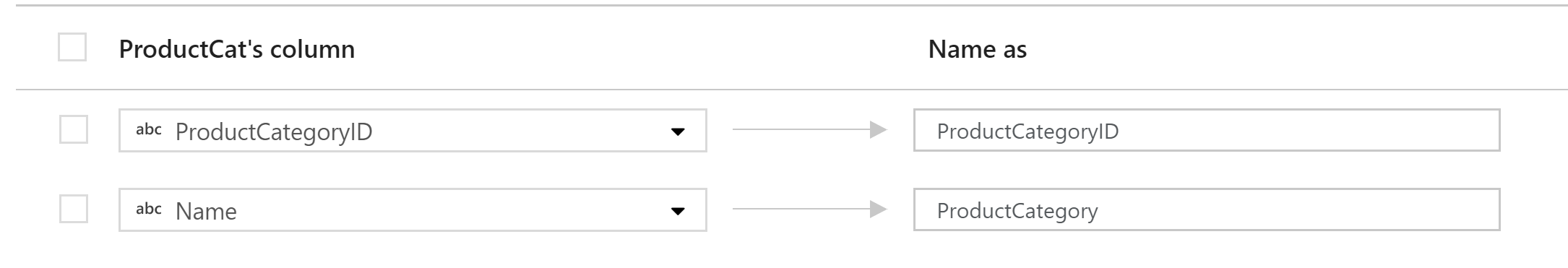
1. We now want to click on the “Add Source” button to add a source for our next table:



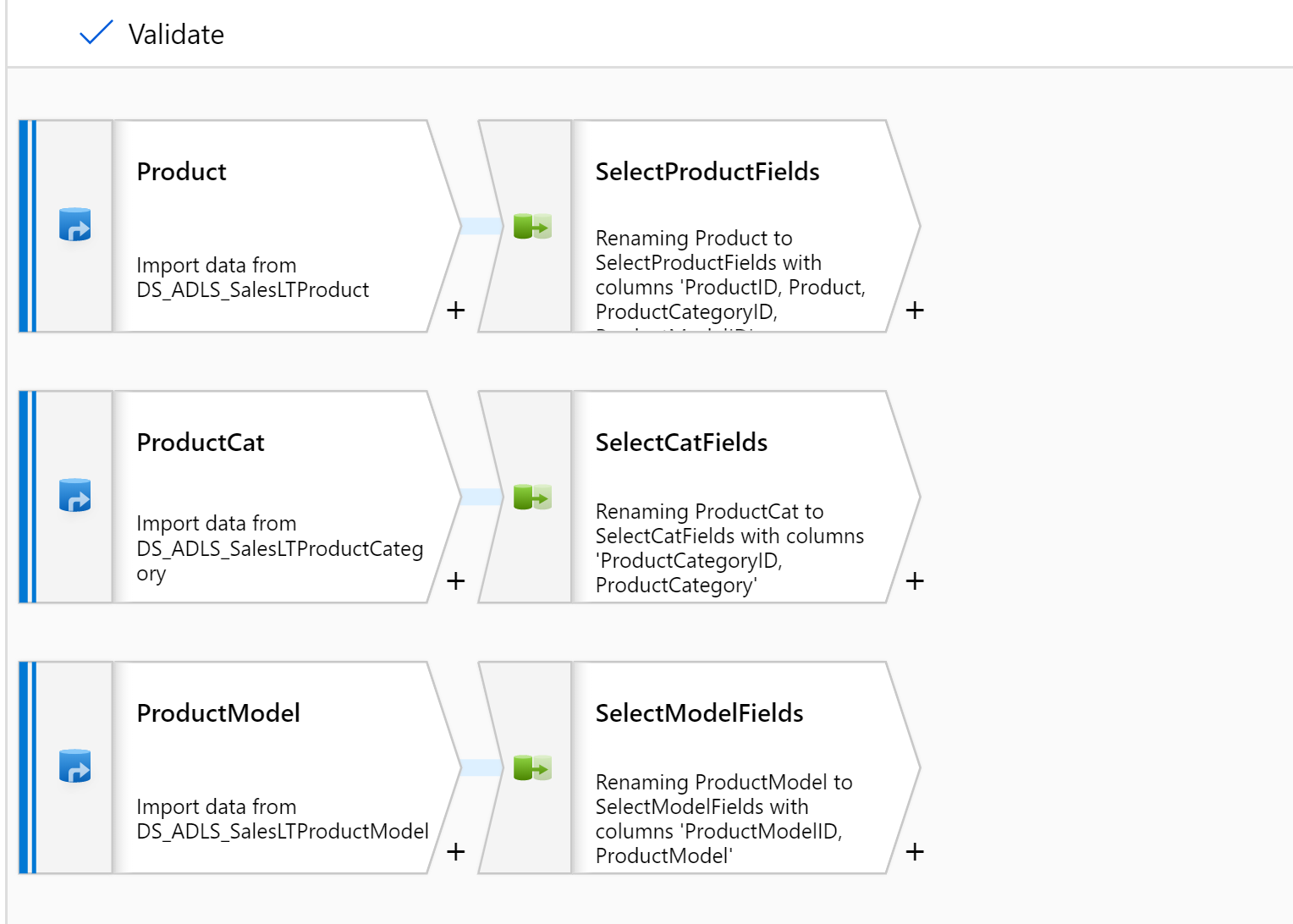
1. Configure this new source to use our ProductCategory dataset:



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

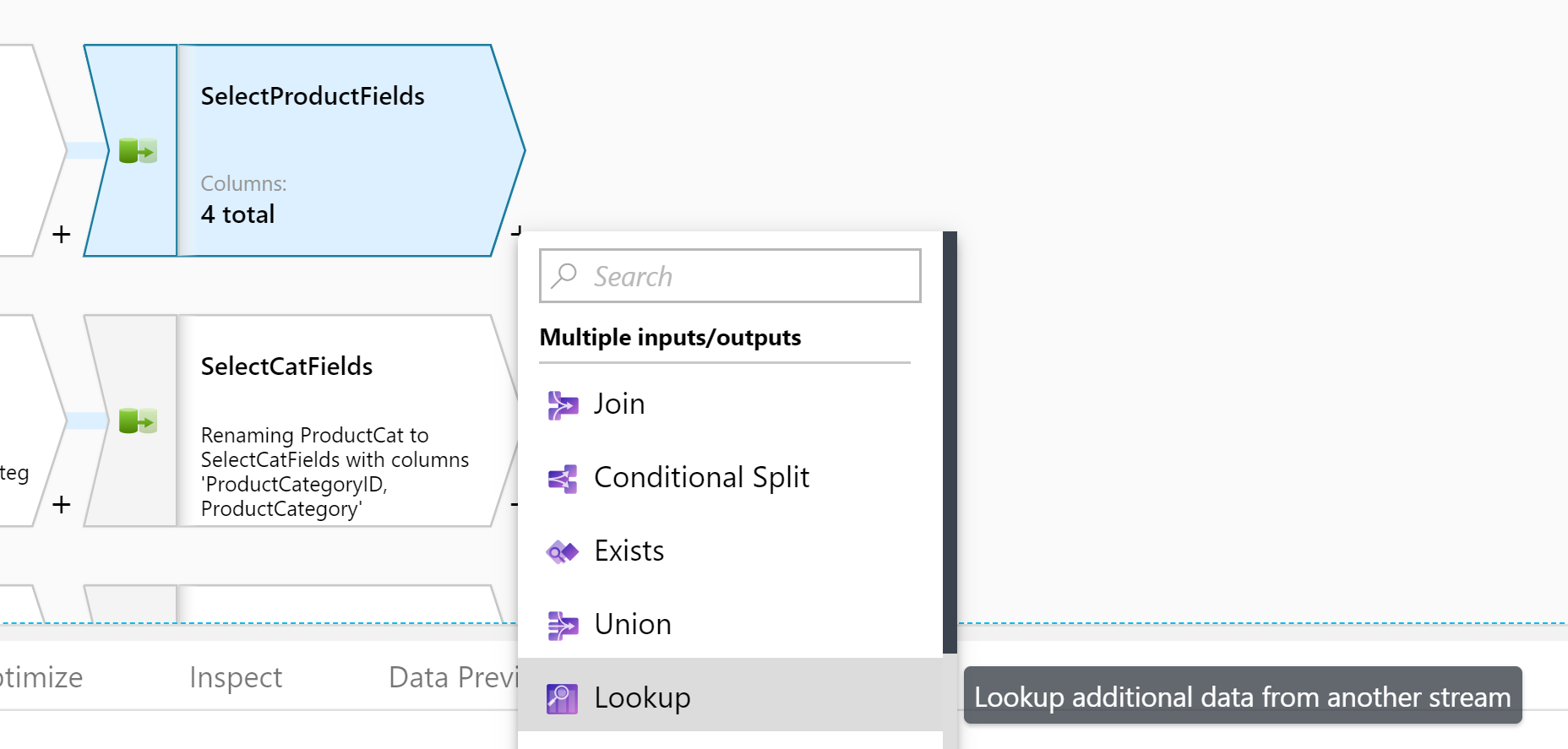
****

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

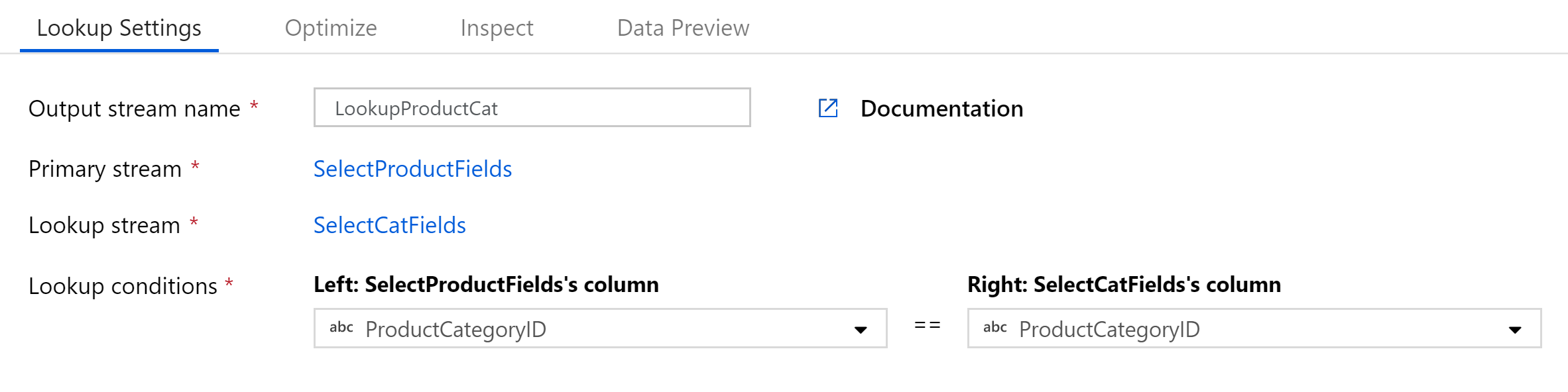


We now have our source data, nicely trimmed of excess fields that we can combine into a single dataset.

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

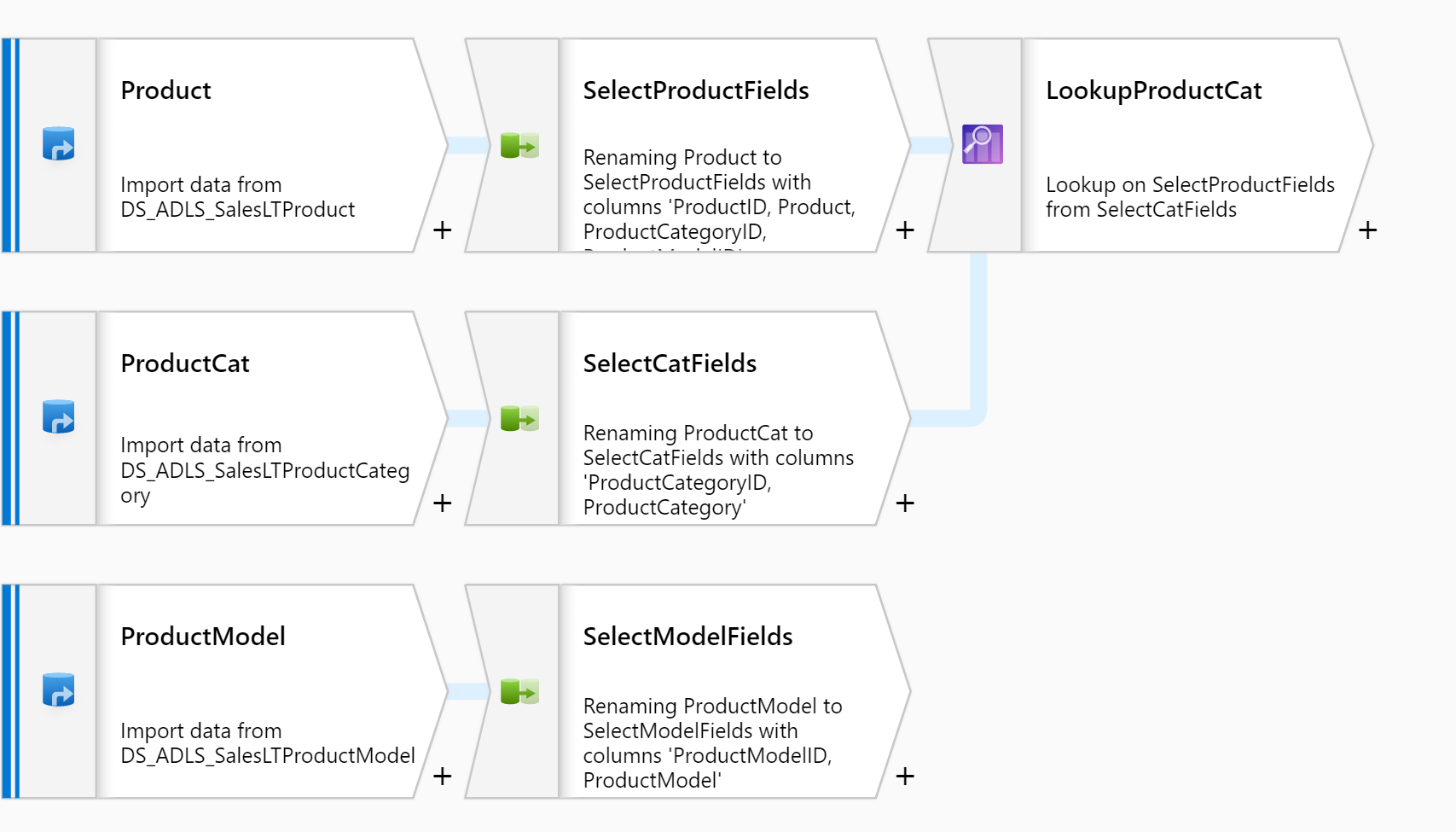


1. Select the “SelectCatFields” stream as the reference stream and configure it to use the ProductCategoryID to perform the reference join, like so:

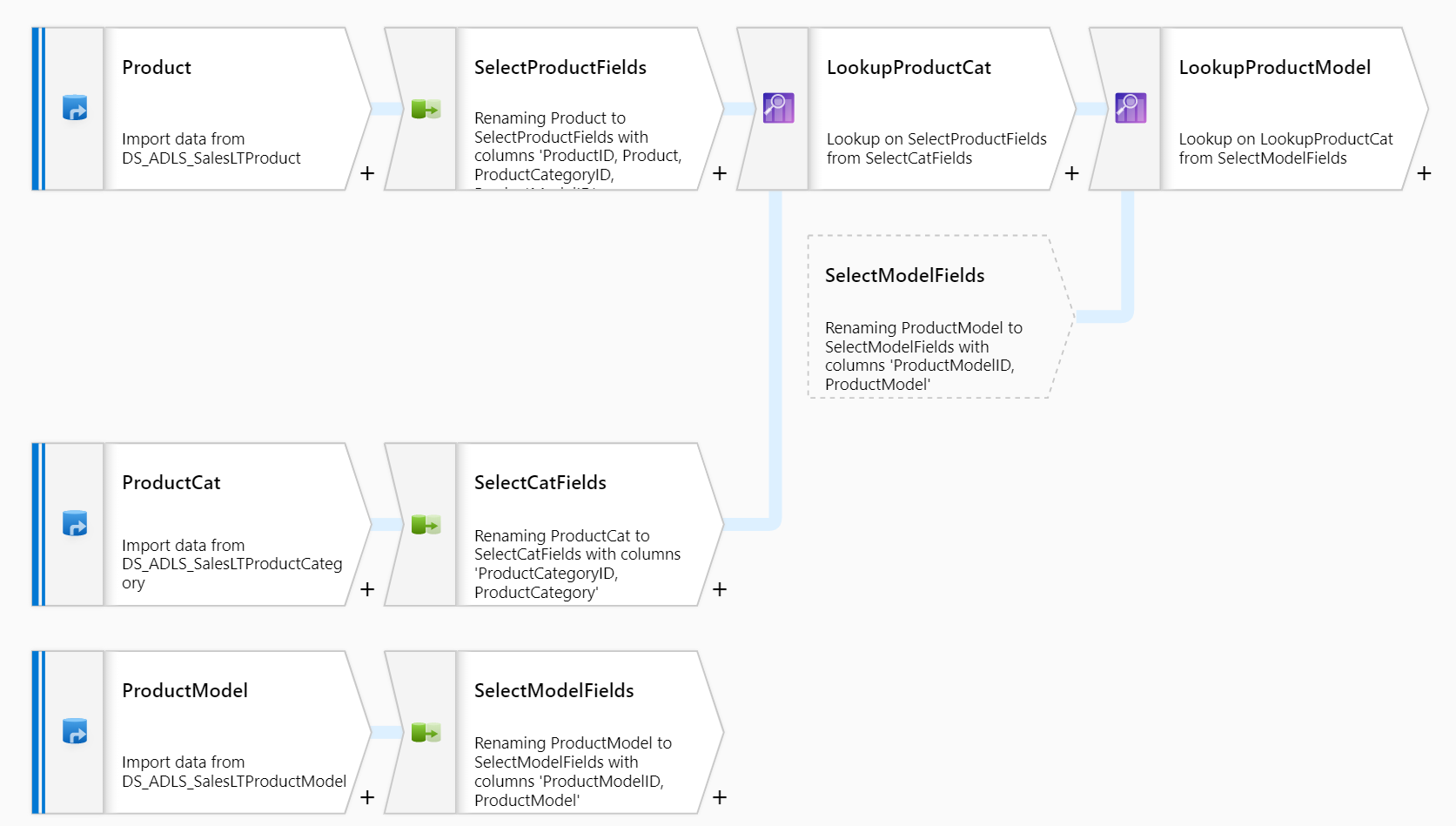


Be careful here – you can actually select to use the output of any of the transformations, including the original source ones, before we had stripped columns and renamed them.

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

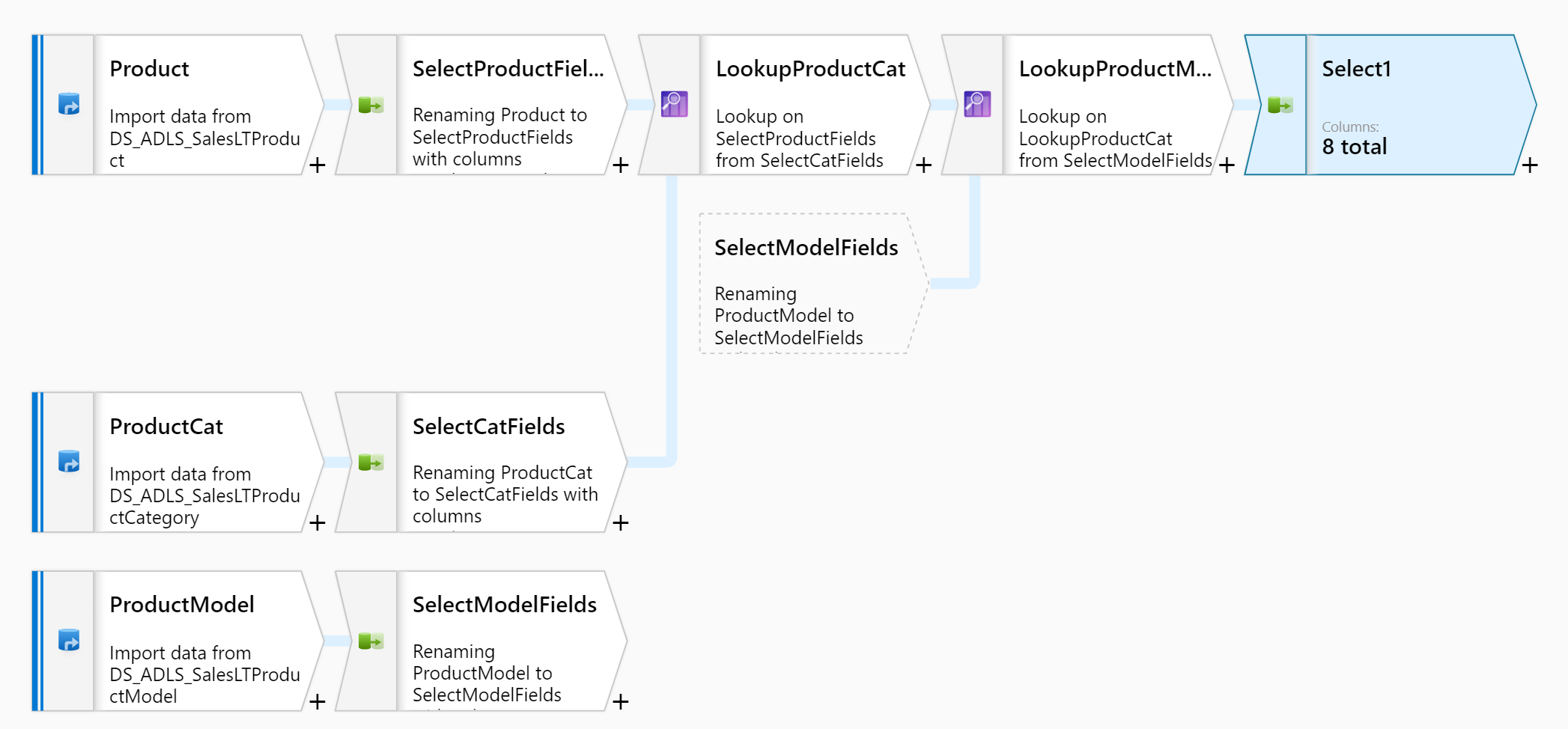


Do the same for the Product Model output stream:

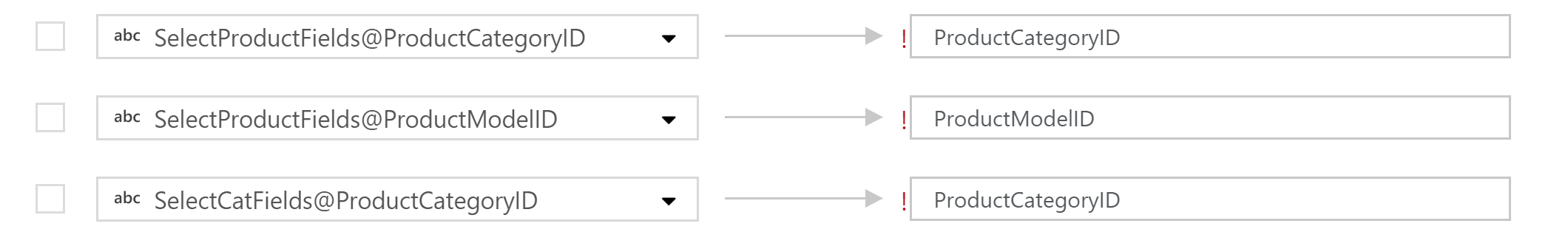


You may notice, we didn’t select the fields to be added at any point, just those that are the join constraints. By default, it will bring the whole table into the aggregation, including duplicates of the keys.

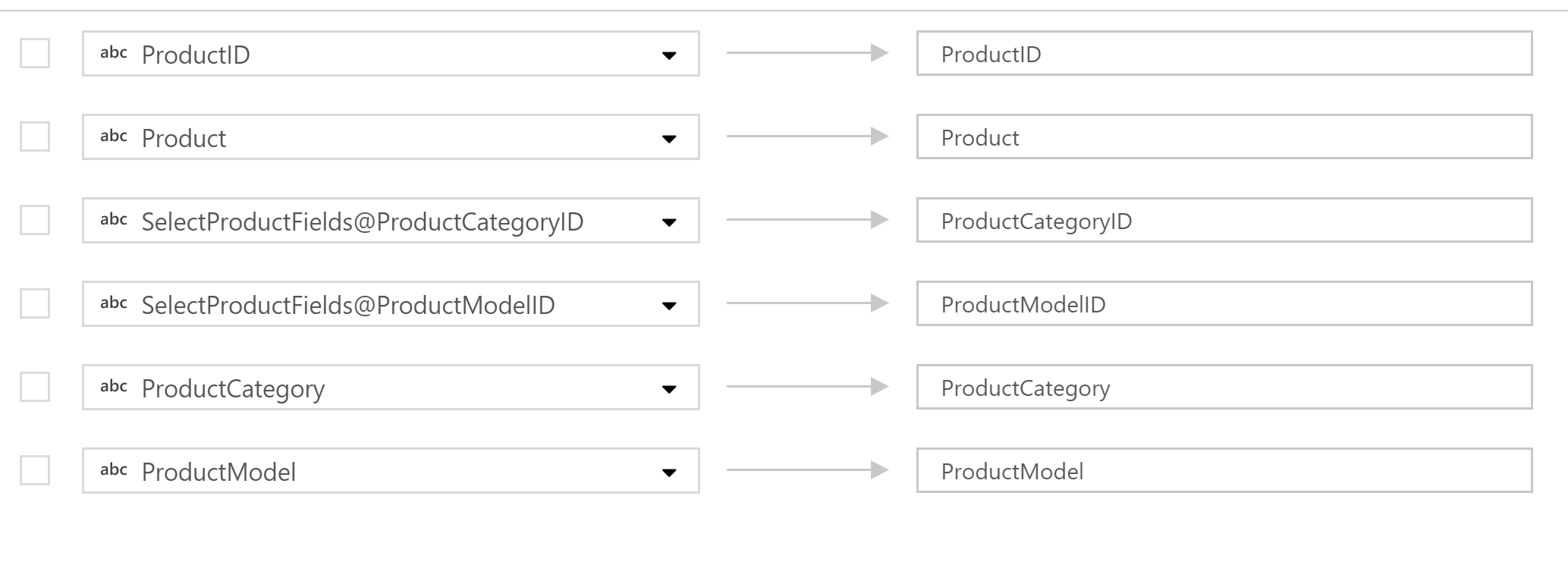
1. Let’s add a final select transformation to get rid of those duplicate keys:



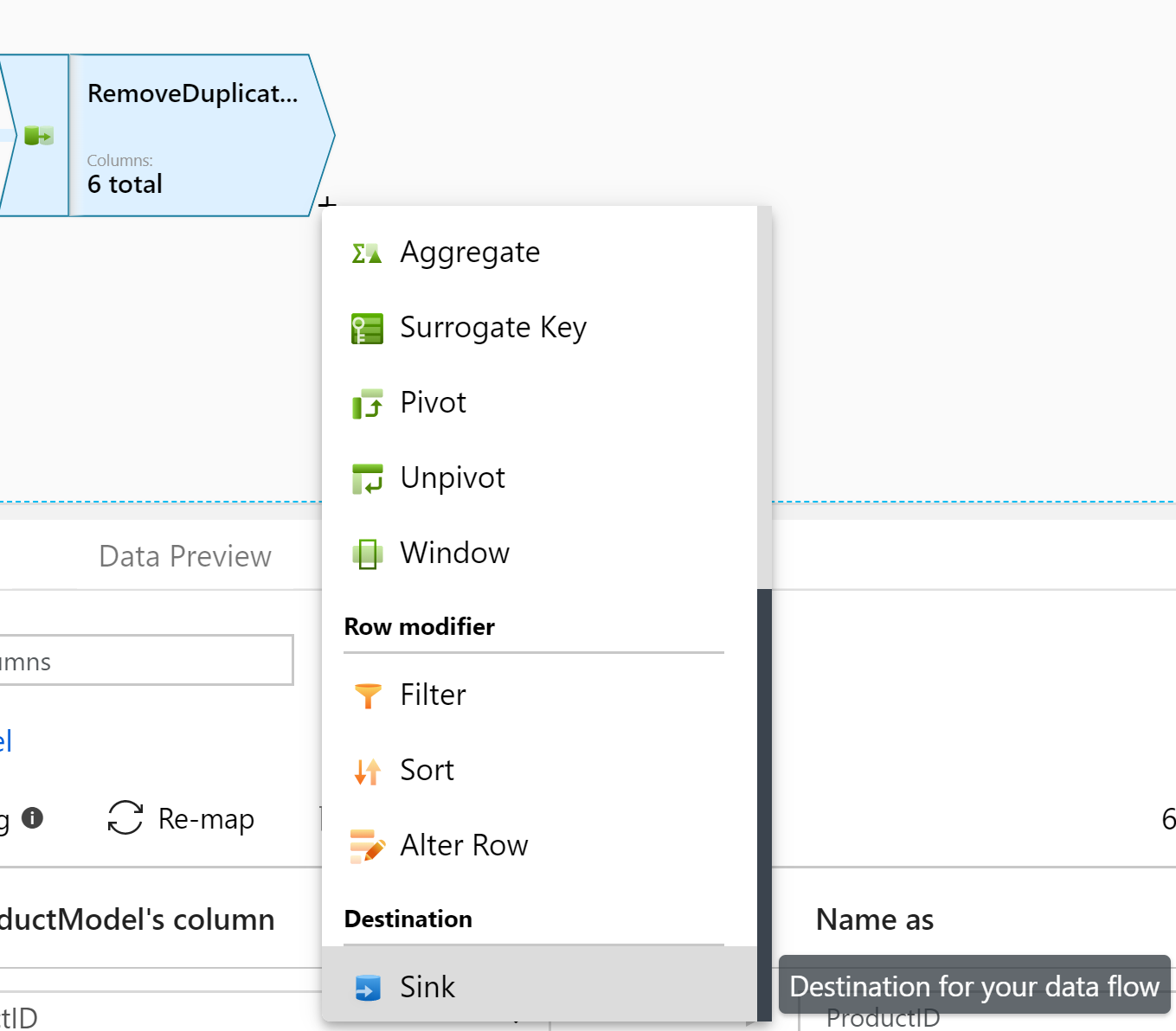
The duplicate columns will be highlighted and use a SELECT syntax to denote which stream they orginally came from:



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

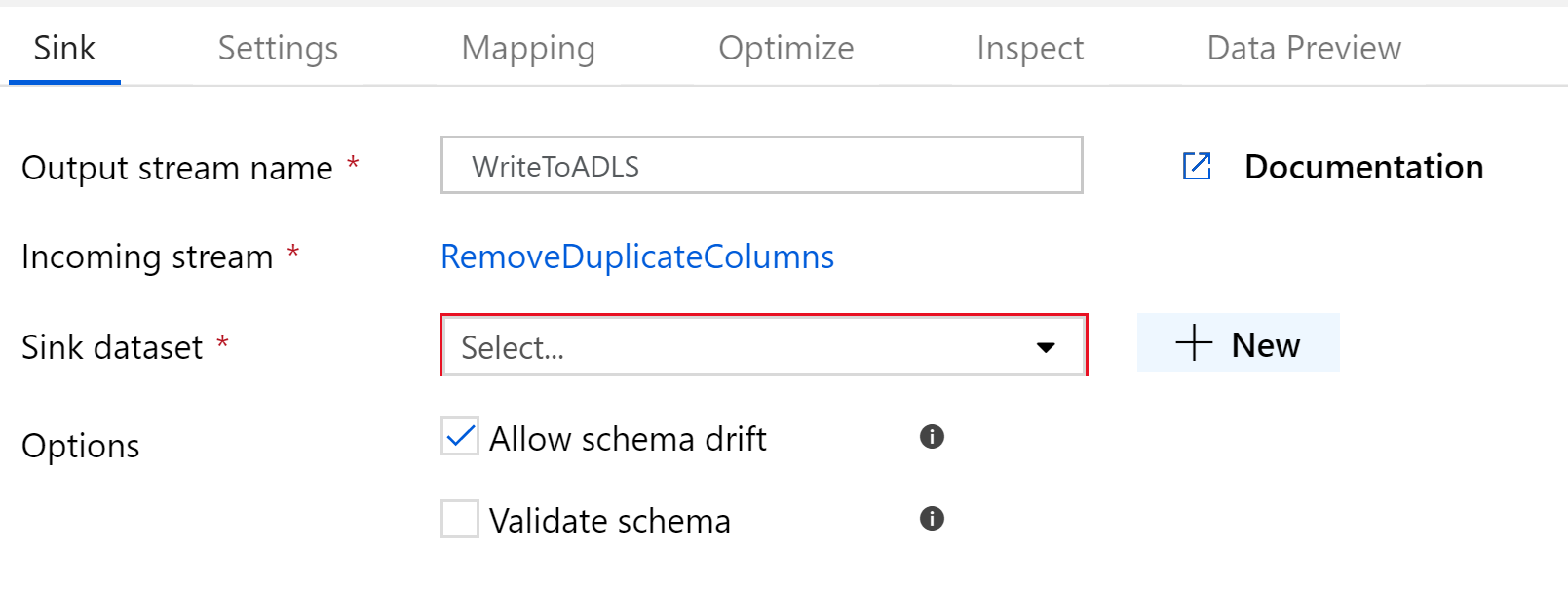


1. Finally, we want to write our data back to our lake in it’s new form – we do this by adding a “Sink” transformation at the point where the data is in the right state:



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

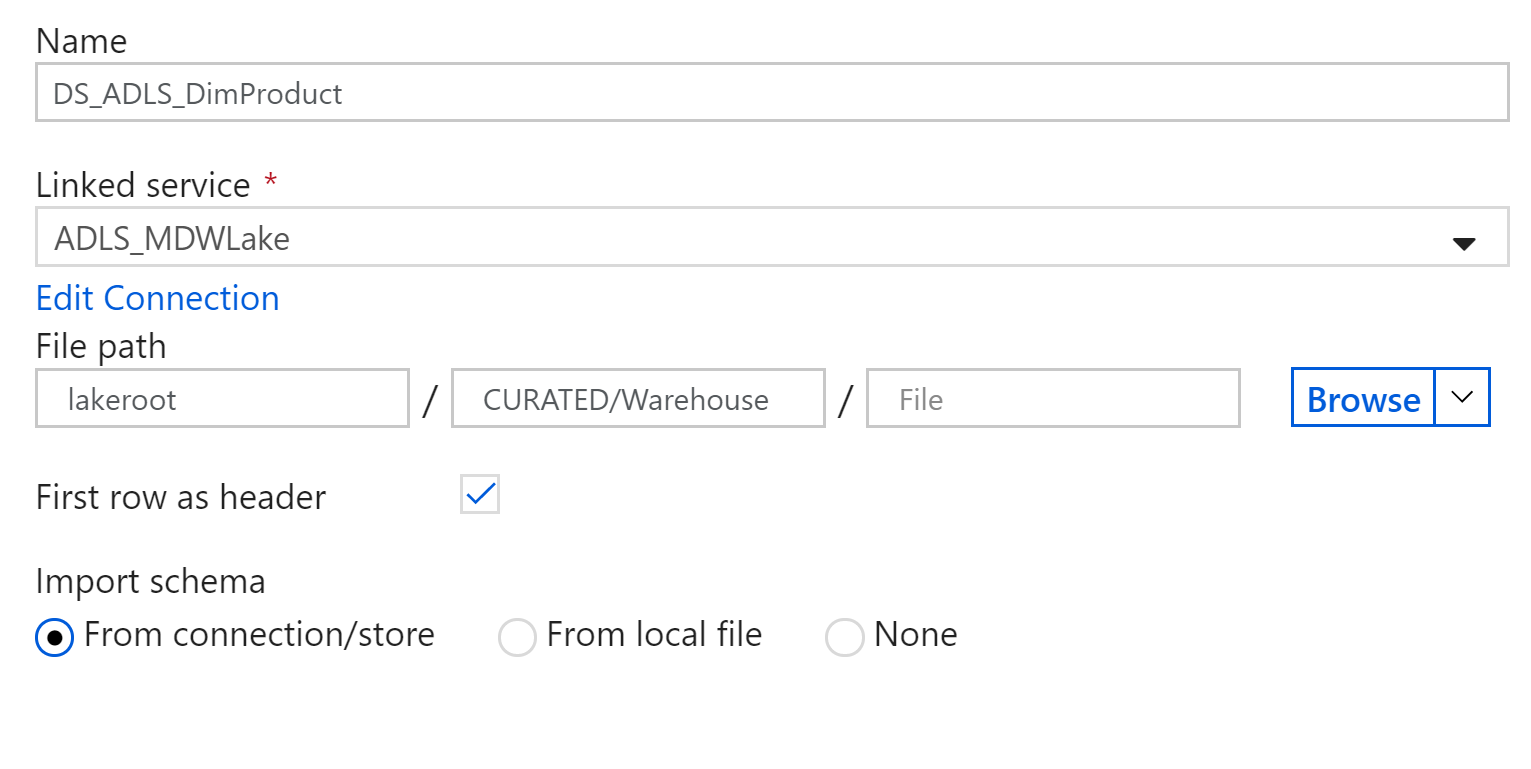


1. Select Data Lake Store Gen 2 as the destination type, then select the file format.

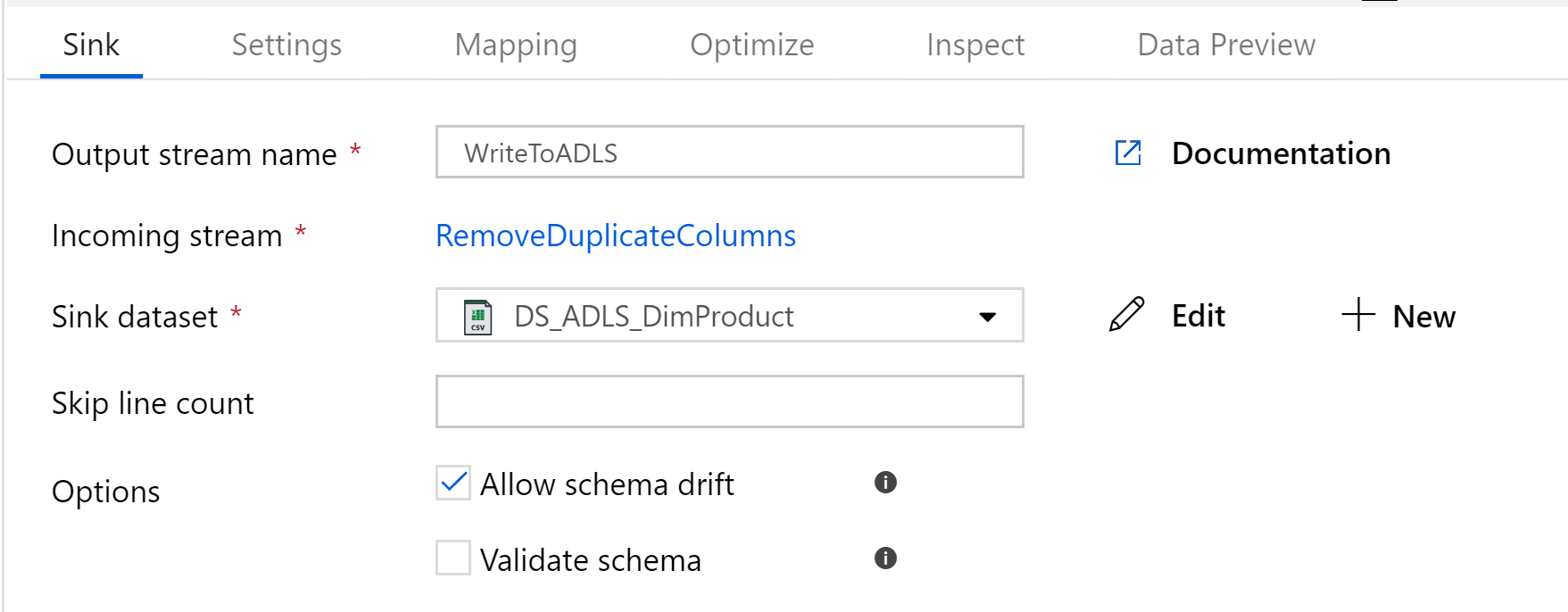
For most scenarios, we would write this as a parquet file for performance – but for ease of testing our transformation, let’s leave it as a CSV

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:

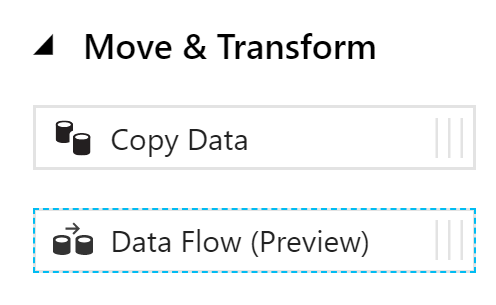


And that’s it! That’s our working Data Flow, ready to go!

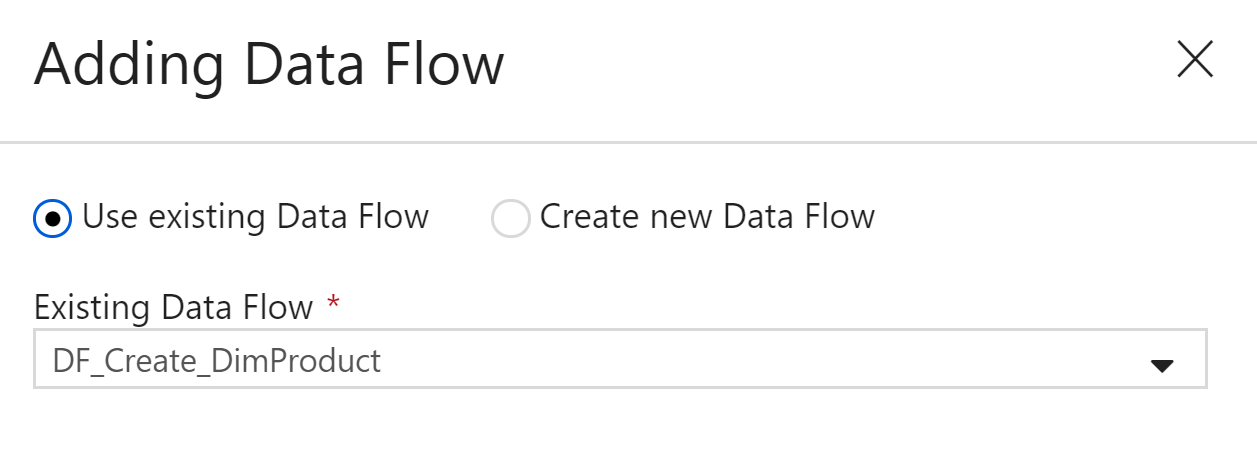
## Lab 04.C – Create a pipeline for your Data Flow

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.

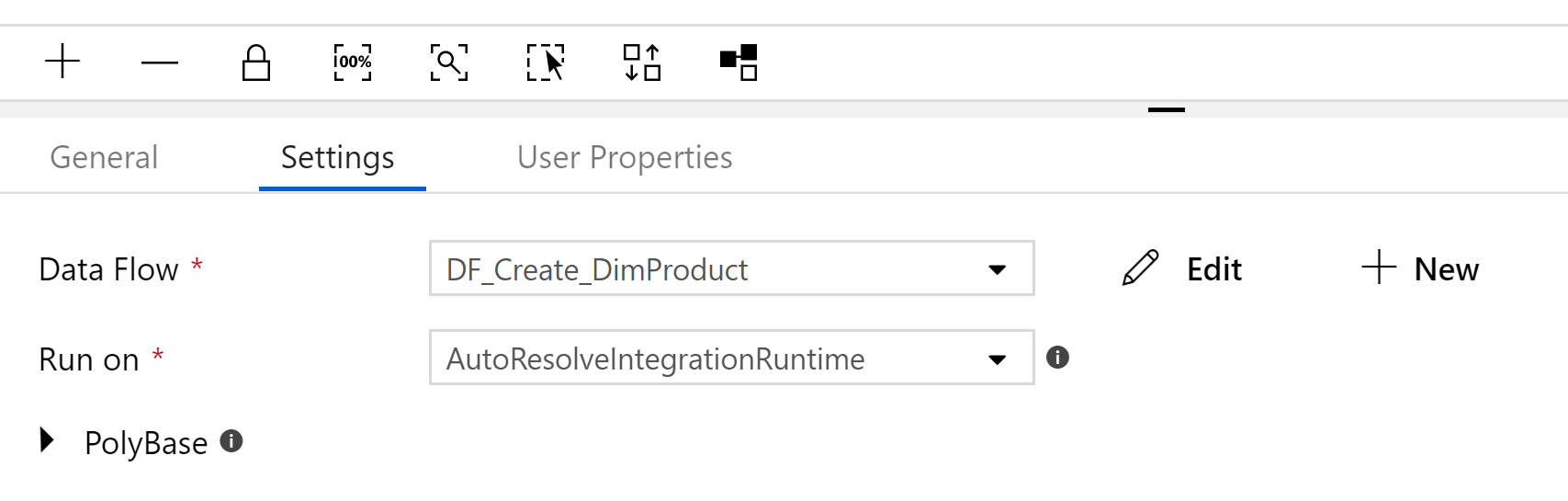
1. Create a new pipeline and drag on a “Data Flow” activity, found in the “Move & Transform” menu:



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:

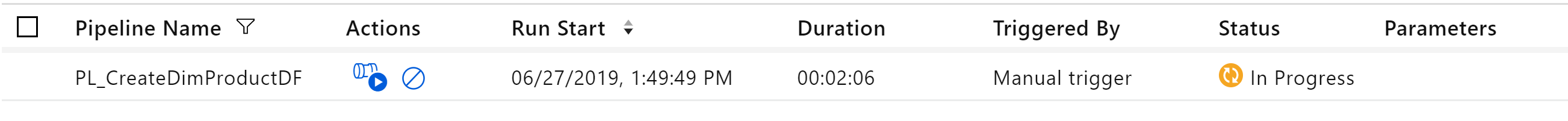


If you look at the settings for your new activity, there isn’t much to do. By default (and as the only option in preview), the data flow will work on an internal ADF Databricks cluster and perform it’s own sizing



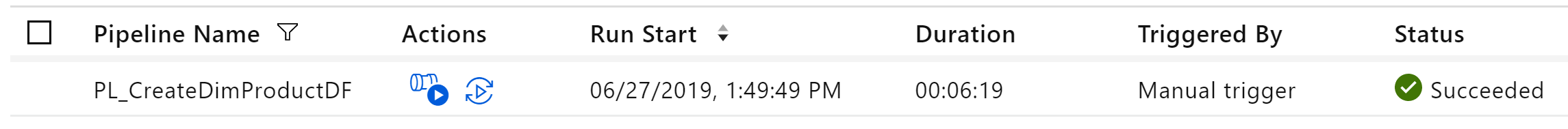
In future, it is expected that we’ll be able to tweak the performance by changing the size/scale of the spark cluster our data flow is running on.

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

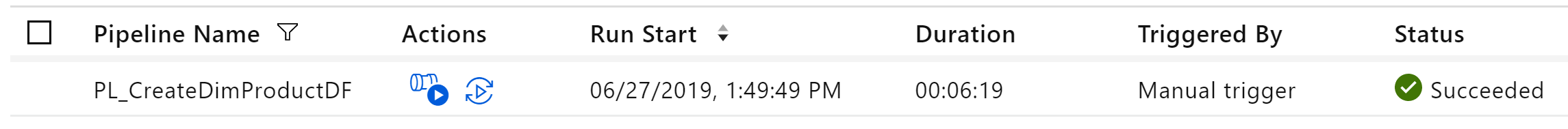


It’ll probably take a couple of minutes before it does anything – that’s because it’s provisioning the spark cluster, which has an overhead. They’re looking to reduce this, but for now bear in mind that these flows are generally meant for fairly large data processing tasks.

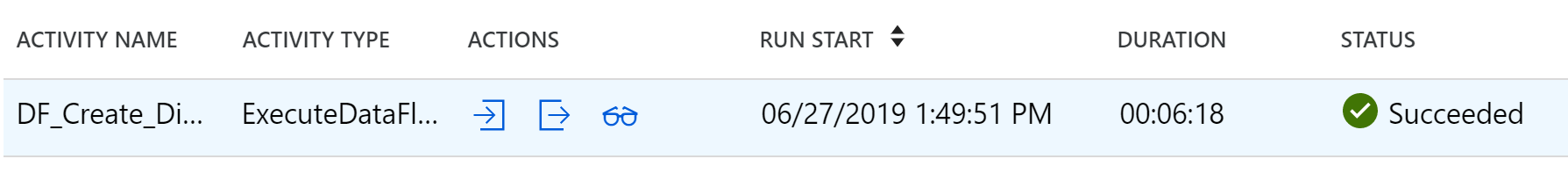
Eventually, we’ll see this turn green as the transformation succeeds:

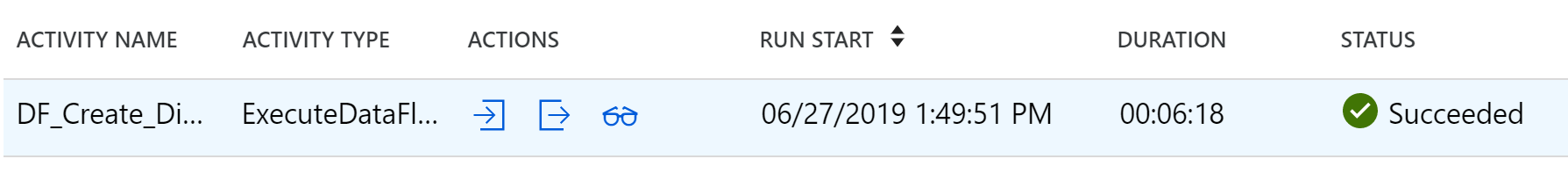


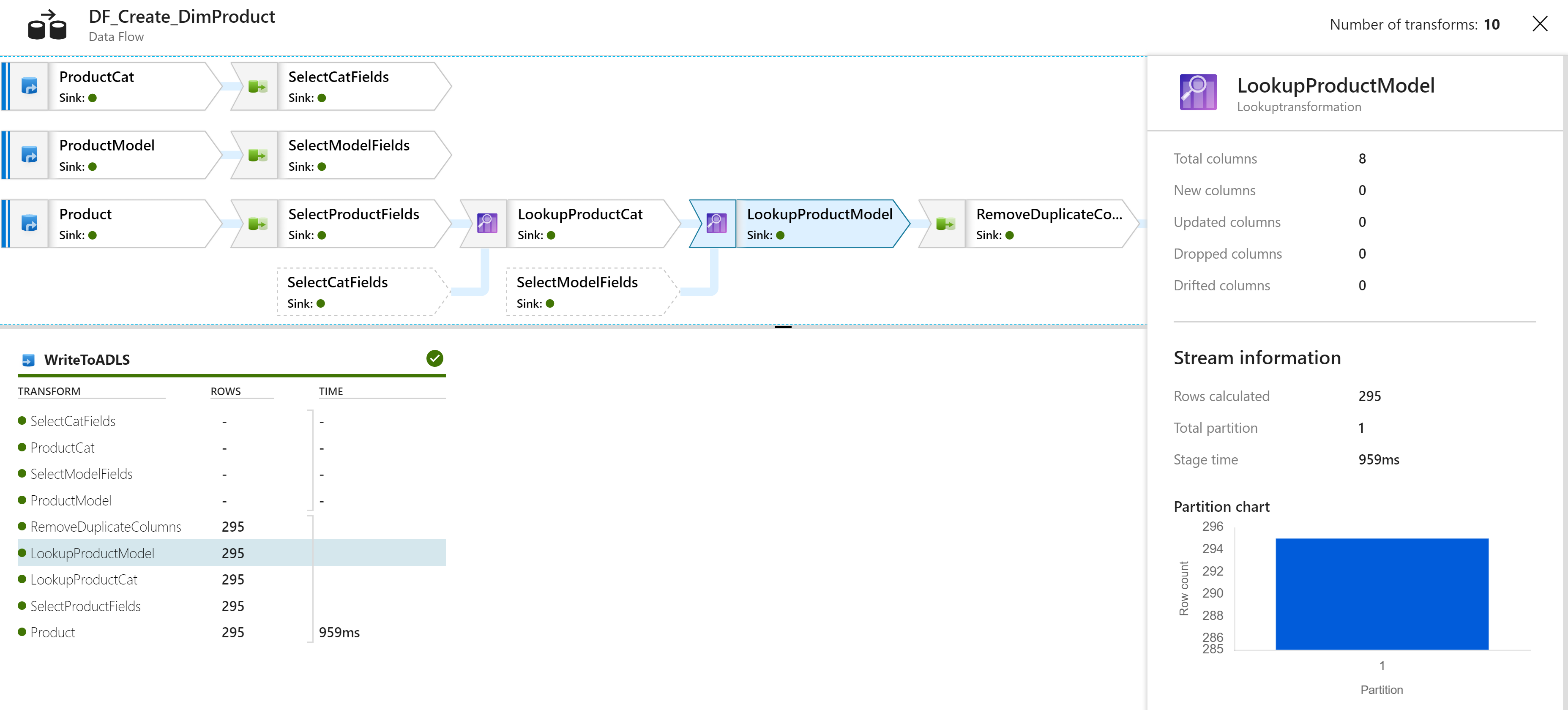
What’s good to learn is the debug/audit information available

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!