Shared Jobs in Matillion ETL: How to Create Variables (Part 1 of 2)

This two-part blog will walk you through setting up and creating a Shared Job in Matillion ETL using variables. In Part 1, we will walk you through the setup process for Job and Grid Variables to use later in a Shared Job. ***Part 2*** *will look at creating and packaging the Shared Job.*

https://www.matillion.com/blog/shared-jobs-in-matillion-etl-how-to-create-variables-part-1-of-2

What is a Shared Job?

There are times when orchestrating and transforming data can be tedious, due to the large amounts of data and tables to process. Users sometimes need to create many similar but distinct jobs to perform repeated functions on tables. These jobs may only have minor differences. To address this job fatigue, Matillion came up with the concept of Shared Jobs.

Shared Jobs bundle entire workflows – Orchestration jobs (and the Transformation jobs they link to) – into a custom component. Within **Matillion ETL**, users can create their own components using the Shared Jobs feature. Shared jobs help developers in two ways: they can help save development time when it comes to common workflows. And, you create templated jobs that you will use again and again across your projects.

Let’s look at the Matillion ETL functions and features you will need to understand in order to build your own Shared Jobs.

Job Variables

In Matillion ETL, you have the ability to create Job Variables, which, if you are familiar with programming, are essentially scalar variables. Job Variables can be used to define a reference to a value instead of hard coding a value. In the context of Shared Jobs, Job Variables can be set up within the Shared Job but with the value supplied later, since the value may be unknown to the creator of the Shared Job. This means that it is possible to create jobs that can be driven by configuration instead of fixed values. For example, if we are writing one job to process multiple tables with different names, all we need to do is set the Table Name within the job to be a variable.

How to Create a Job Variable

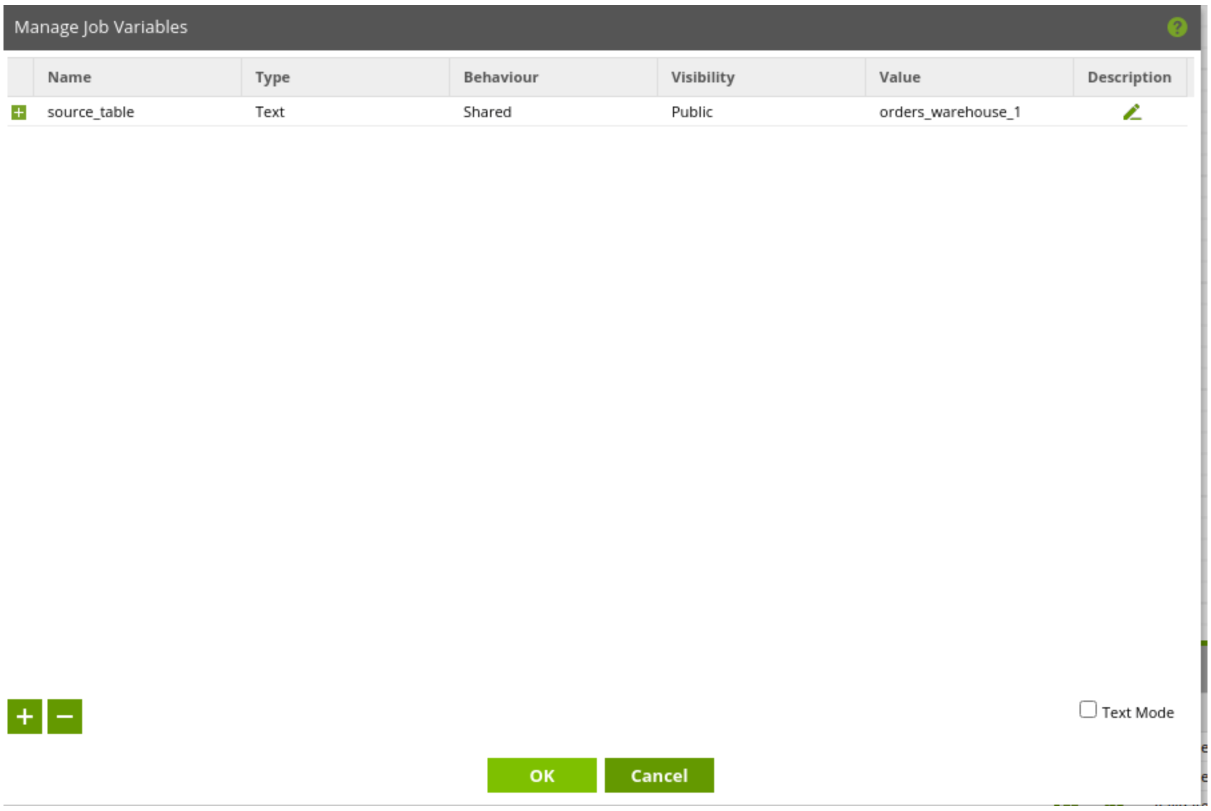
In this example, we have three tables of data from different warehouse orders that we want to manage. In creating a Job Variable, we only need to configure the variable once and then we can reuse it. This is helpful because it makes development easier and prevents the need to duplicate logic. As we add more warehouses, we can easily and quickly combine this data within an existing ETL flow by reusing our existing Transformation Job and just running it again with the new table name passed into the Job Variable.

First, create a Job Variable by clicking ‘+’ in the bottom left-hand corner of your Matillion ETL screen. When configuring your Job Variable, you can set:

- Name: Specify the name of the variable which you will later reference.
- Type: Specify the Matillion ETL Data Type as text, numeric, or datetime.
- Behavior: Set the branch behavior inside a job. This determines how the variable is updated when more than a single job branch is making use of it.
- Visibility: Select either Public or Private. When a variable is private, it cannot be discovered or overwritten when this job is called from a Run Orchestration or Run Transformation component. For this reason,

visibility should be set to Public for use in a Shared Job, if you wish for the value to be set by the end user of your Shared Job.
- Default Value: Set the default value for this variable in this job. The default value will be used to validate the shared job during development – and is the value used if a value is not passed in by a calling job.

In this example we will be creating a variable called *source_table* which will be of type Text, will have a Shared behaviour, will be publicly visible, and will have a default value of orders_warehouse_1.



Name	Type	Behaviour	Visibility	Value	Description
source_table	Text	Shared	Public	orders_warehouse_1	

Note: because we set a Default Value of a table with the structure we require – Matillion ETL is able to read the table structure and thus the job can validate.

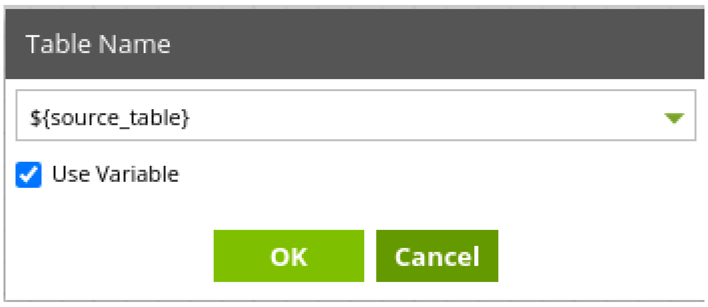


Table Name

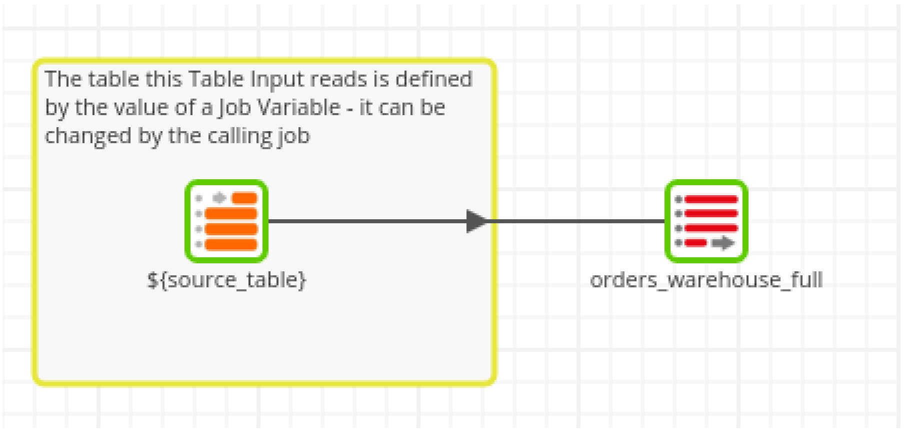
${source_table}

[✓] Use Variable

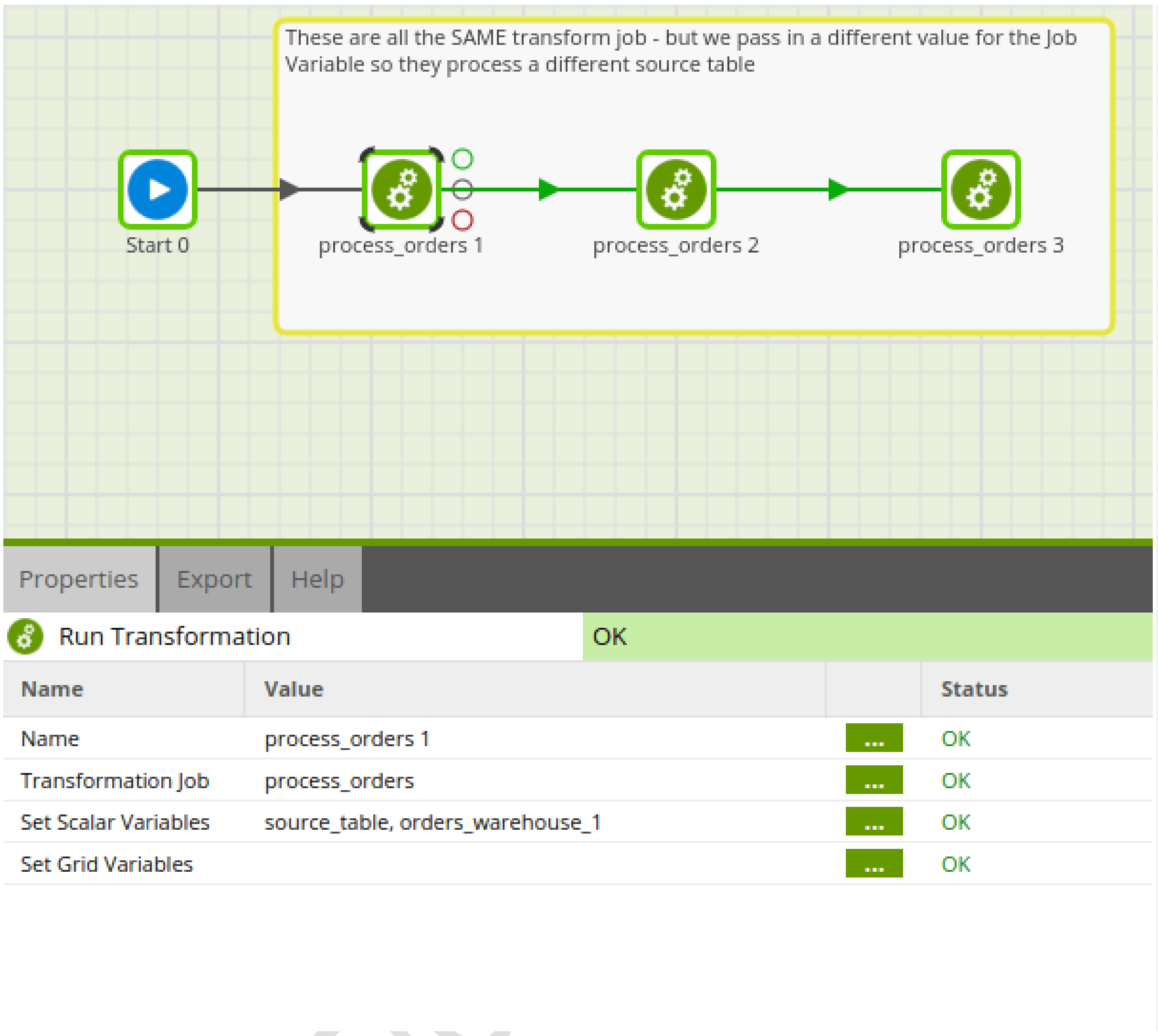
OK Cancel

When you want to reference your new variables in your job, you will be able to use ${variable}, with ‘variable’ being the name of the variable you set. In our example, this will be ${source_table} to reference a warehouse orders table.

This simple example Transformation Job simply appends all of the rows from the source table into the target table.

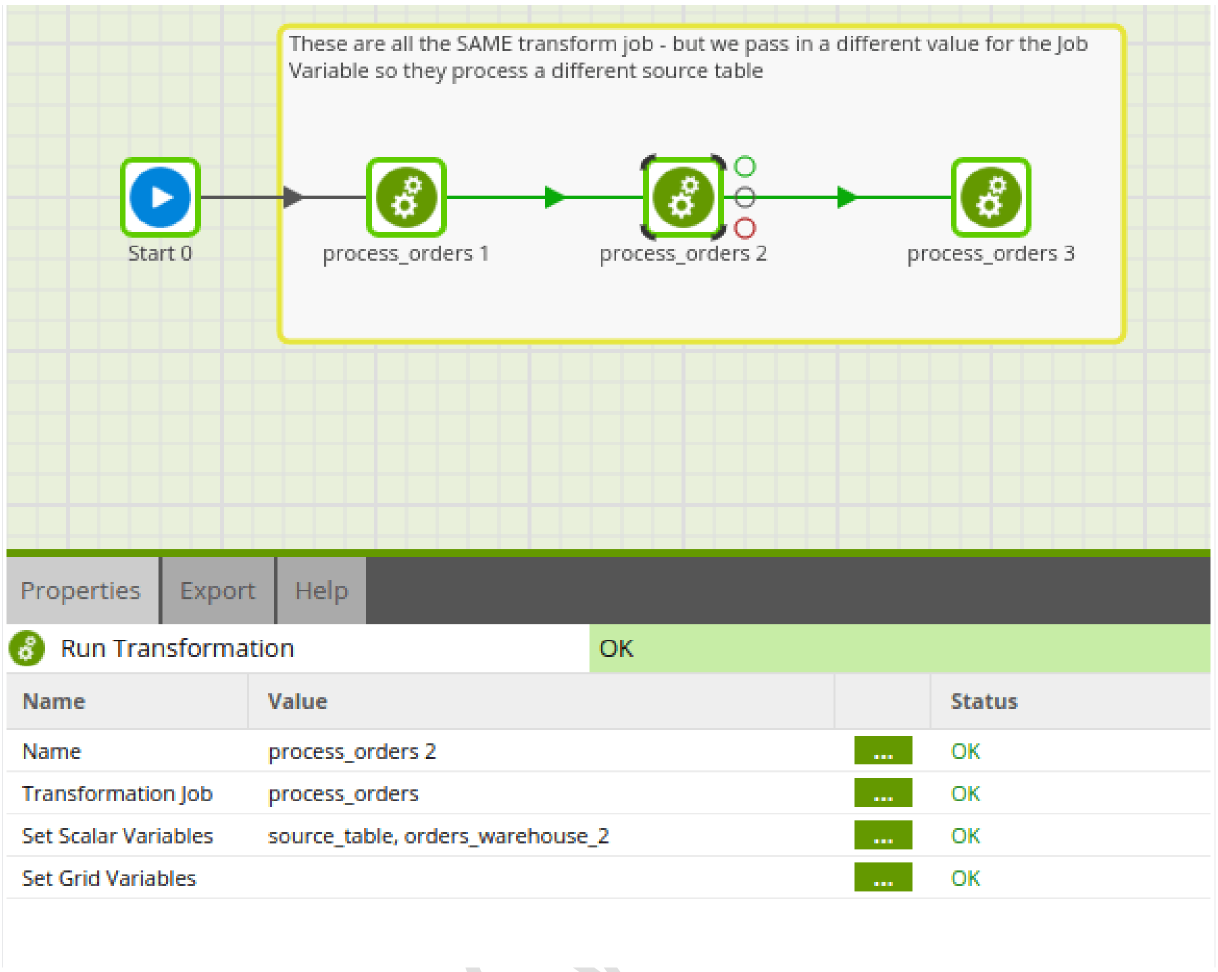

The table this Table Input reads is defined by the value of a Job Variable - it can be changed by the calling job

If we run this job directly, it will use the default value of the variable. However, if calling from a parent Orchestration Job, we can override this default value. Below you can see that we have brought the same Transformation Job into our Orchestration Job three times to account for each table of warehouse order data.


These are all the SAME transform job - but we pass in a different value for the Job Variable so they process a different source table
Start 0
process_orders 1
process_orders 2
process_orders 3
Properties
Export
Help

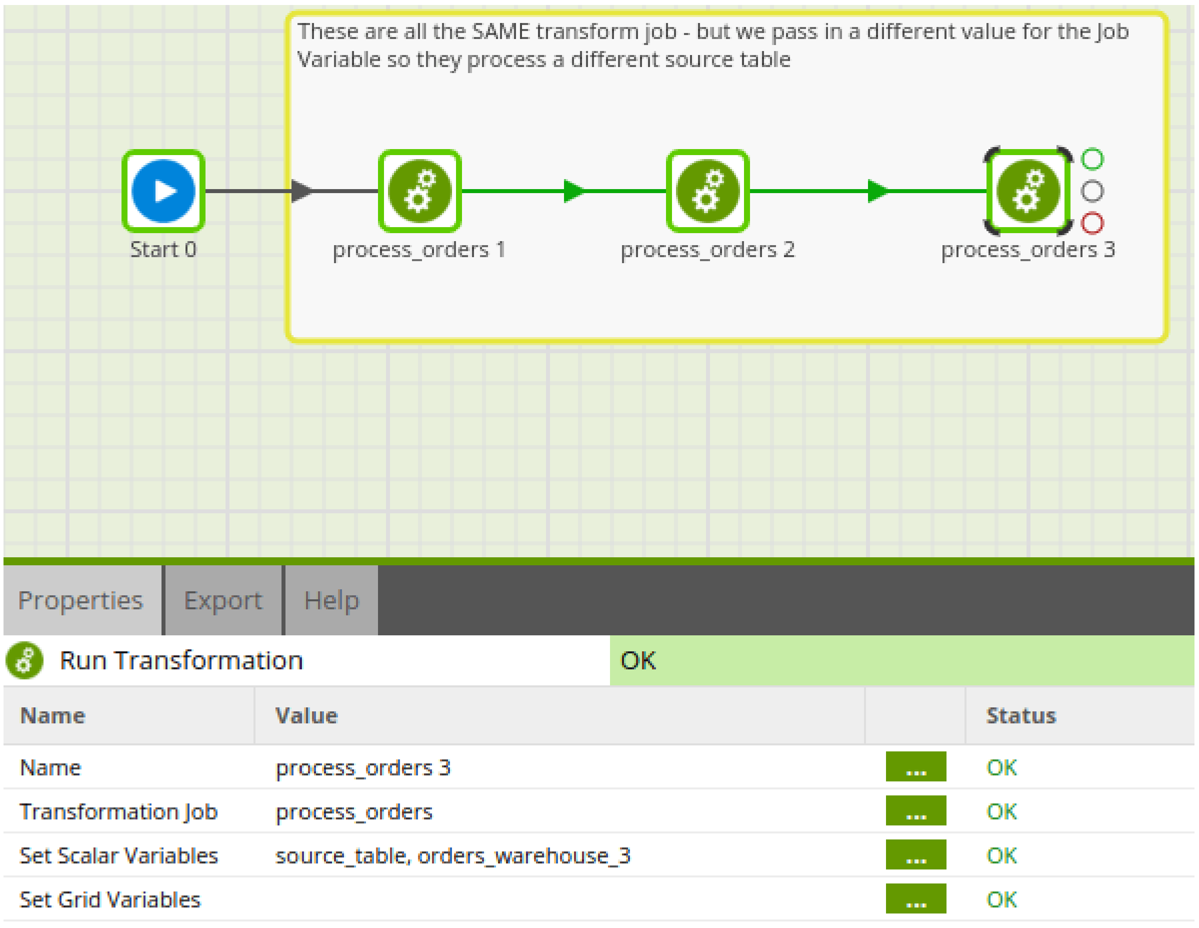
Run Transformation OK

Name	Value		Status
Name	process_orders 1	...	OK
Transformation Job	process_orders	...	OK
Set Scalar Variables	source_table, orders_warehouse_1	...	OK
Set Grid Variables		...	OK


These are all the SAME transform job - but we pass in a different value for the Job Variable so they process a different source table
Start 0
process_orders 1
process_orders 2
process_orders 3
Properties
Export
Help

Run Transformation OK

Name	Value		Status
Name	process_orders 2	...	OK
Transformation Job	process_orders	...	OK
Set Scalar Variables	source_table, orders_warehouse_2	...	OK
Set Grid Variables		...	OK


These are all the SAME transform job - but we pass in a different value for the Job Variable so they process a different source table
Start 0
process_orders 1
process_orders 2
process_orders 3
Properties
Export
Help

Run Transformation OK

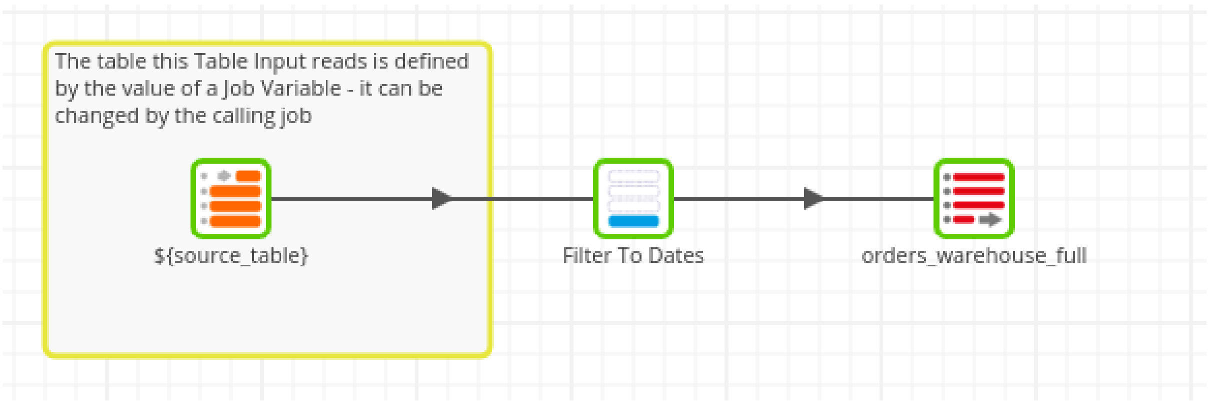
Name	Value		Status
Name	process_orders 3	...	OK
Transformation Job	process_orders	...	OK
Set Scalar Variables	source_table, orders_warehouse_3	...	OK
Set Grid Variables		...	OK

You can see from the Properties of the Run Transformation component we are running the same Transformation Job 3 times, but passing in a different value for the ${source_table} variable

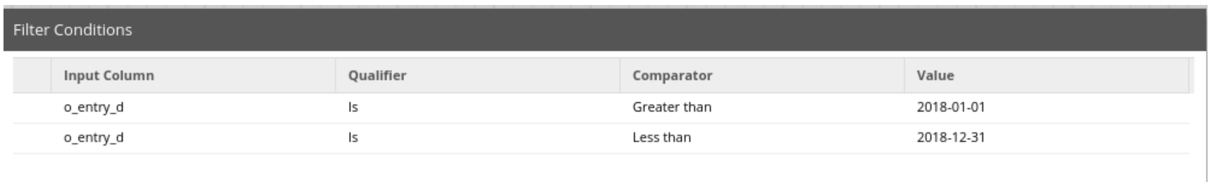
To add another table, we would follow the same quick process and add a fourth run of the Transformation Job, but pass the new table name when calling it. Alternatively, if our source table structure changes, we only need to amend the one Transformation Job and the logic will be applied to the processing of all four warehouse order tables.

Grid Variables

Also common in Shared Jobs is the concept of a Grid Variable. Unlike a Job Variable, which can only hold a single value, a Grid Variable can hold multiple values (similar to a 2D array found in many programming languages). This is helpful if you have a commonly used set of values that you want to replicate in a number of different jobs: For example, filter conditions or a selection of data columns within a Data Source component (i.e. Salesforce). If we take our previous example, we realize we want to filter the order_dates from each warehouse. We can add a Filter component to our existing Transformation Job.


The table this Table Input reads is defined by the value of a Job Variable - it can be changed by the calling job
${source_table}
Filter To Dates
orders_warehouse_full

We only want dates from a specific year, so we set out filter criteria:



Filter Conditions

Input Column	Qualifier	Comparator	Value
o_entry_d	Is	Greater than	2018-01-01
o_entry_d	Is	Less than	2018-12-31

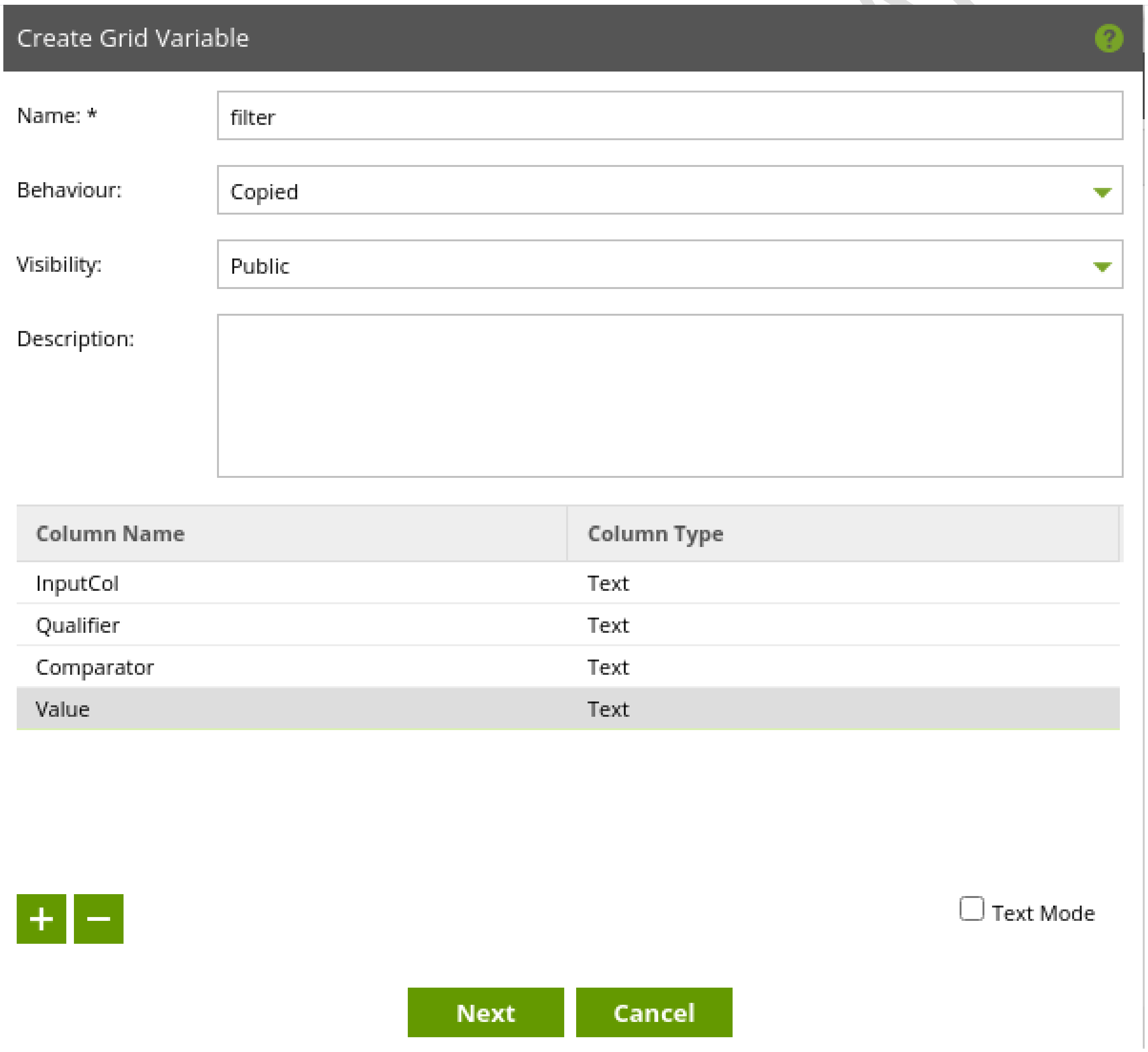
However, by hard coding the above it will apply to every run of the table, even if different warehouses need different filter rules. If each warehouse needs a different set of dates – we can still use a single Transformation Job by adding a Grid Variable.

To add a Grid Variable, right-click on the desired Orchestration or Transformation Job within the explorer panel and select Manage Variables>>Variables>>Manage Grid Variables.

Create a Grid Variable by clicking '+' in the bottom left-hand corner. When configuring your Grid Variable you can set:

- Name: Name of the variable, which you will later reference
- Type: Matillion ETL Data Type as text, numeric, or datetime.

- Behavior: Determines its 'branch behavior' inside of a job. That is, how the variable is updated when more than a single job branch is making use of it.
- Visibility: Select either Public or Private. When a variable is private, it cannot be discovered or overwritten when this job is called from a Run Orchestration or Run Transformation component. For this reason, visibility should be set to Public for use in a Shared Job.
- Description: Write a description for the Grid Variable. This description has no consequence or functionality beyond being present in the dialog for editing the variable and when this job is being called through the Run Transformation and Run Orchestration components.


Create Grid Variable

Name: * filter

Behaviour: Copied

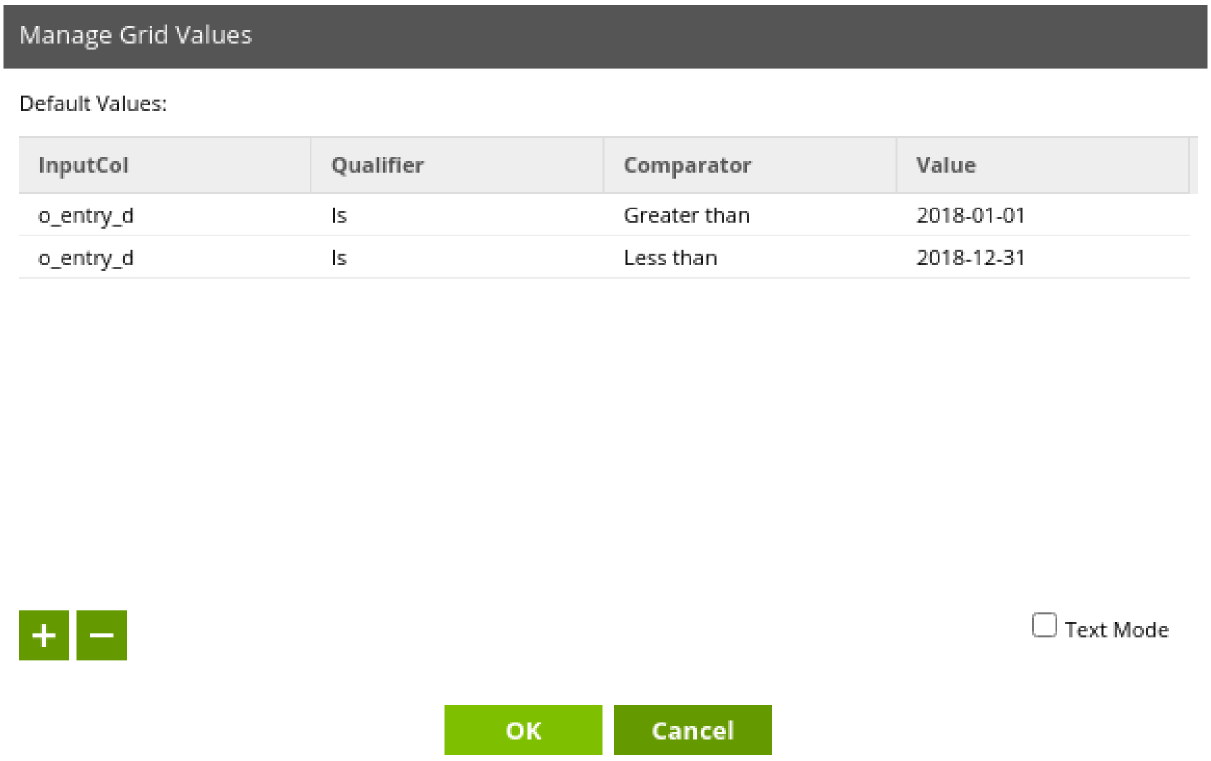
Visibility: Public

Description:

Column Name	Column Type
InputCol	Text
Qualifier	Text
Comparator	Text
Value	Text

The Grid Variable needs the same columns as the Filter Conditions from the Filter component, also set the default values. Like the default value for a Job

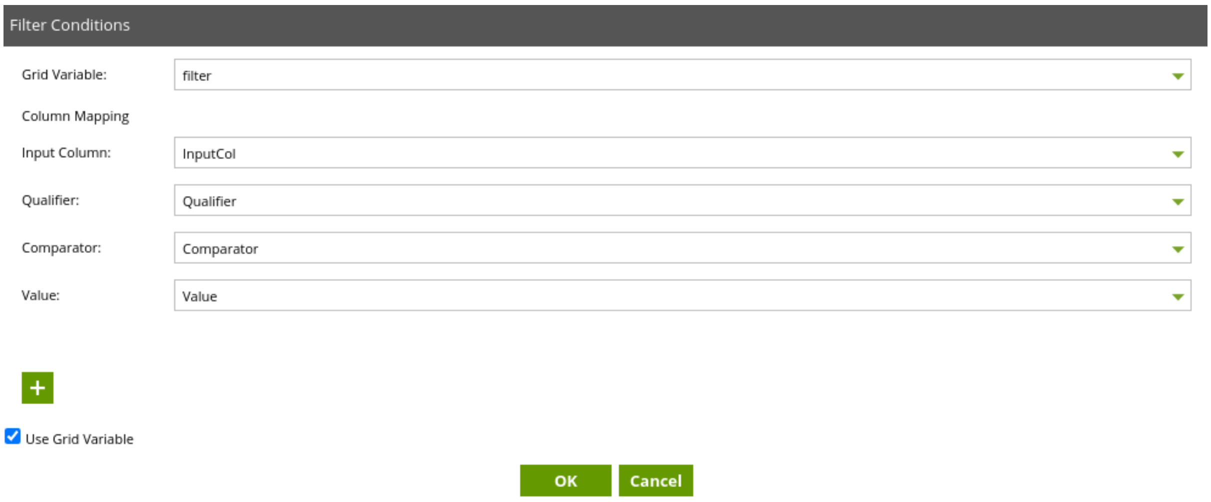
Variable, these values will be used when running the job unless values are passed in from a calling parent job.



Default Values:

InputCol	Qualifier	Comparator	Value
o_entry_d	Is	Greater than	2018-01-01
o_entry_d	Is	Less than	2018-12-31

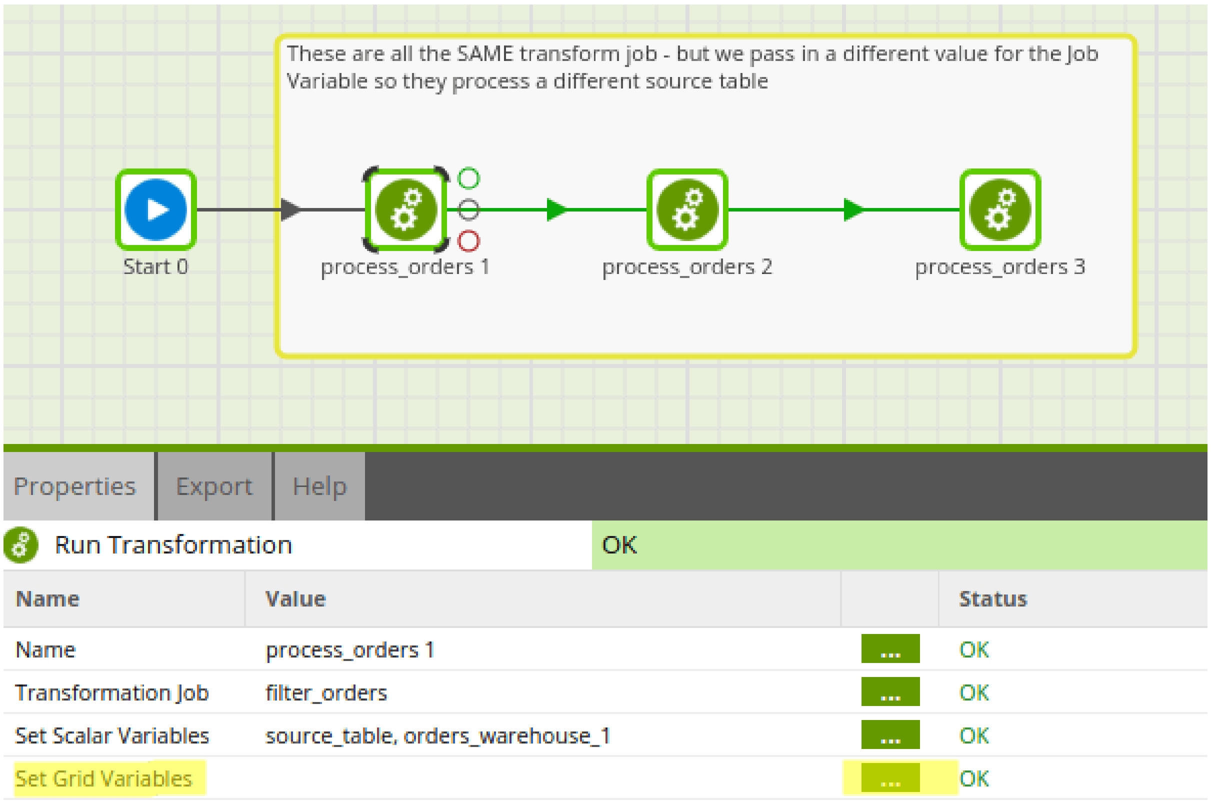
You can use Grid Variables anywhere in Matillion ETL where there is a ‘Use Grid Variable’ checkbox. Going back to the Filter component – click “Use Grid Variable”, and then map the Grid Variable columns to the component columns:



Filter Conditions

Grid Variable:	filter
Column Mapping	
Input Column:	InputCol
Qualifier:	Qualifier
Comparator:	Comparator
Value:	Value

Now that our Grid Variable is set up, we can jump back into our Orchestration Job that runs the same Transformation Job for the three warehouses. We now need to set the values to pass into the Transformation Job to override the default values. Under Properties, select 'Set Grid Variables'.

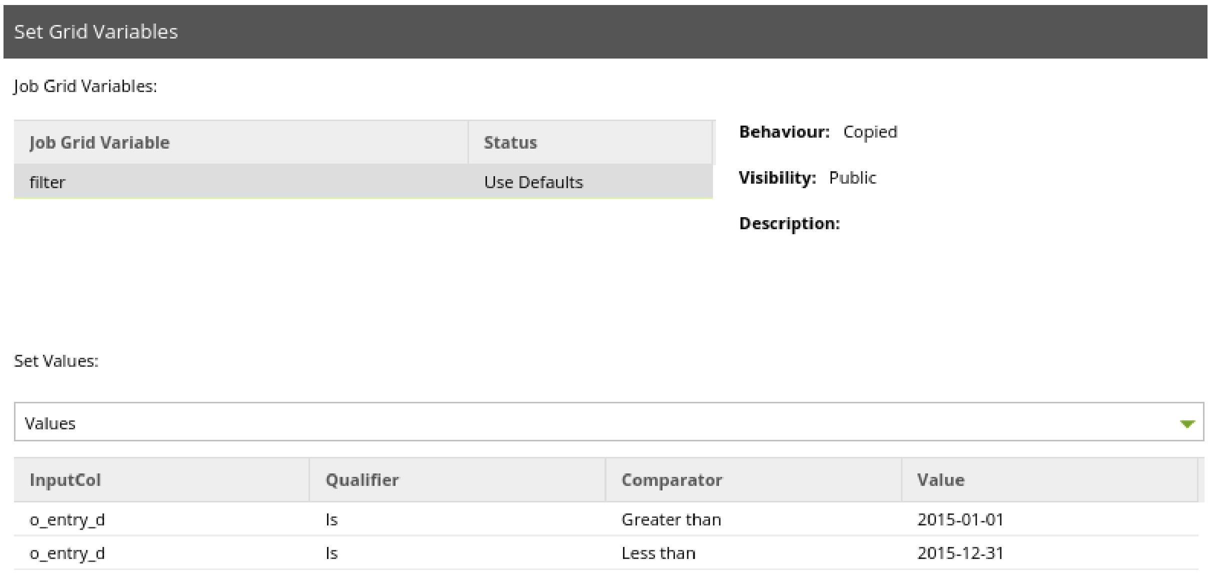


These are all the SAME transform job - but we pass in a different value for the Job Variable so they process a different source table
Start 0
process_orders 1
process_orders 2
process_orders 3
Properties
Export
Help

Run Transformation OK

Name	Value		Status
Name	process_orders 1	...	OK
Transformation Job	filter_orders	...	OK
Set Scalar Variables	source_table, orders_warehouse_1	...	OK
Set Grid Variables		...	OK

Now we can apply our Grid Variable for the filter for each table of warehouse order data. Pictured below is the filter for process_orders 1. We will follow the same steps for process_orders 2 for 2016, and process_orders 3 for 2017.


Set Grid Variables

Job Grid Variables:

Job Grid Variable	Status
filter	Use Defaults

Behaviour: Copied

Visibility: Public

Description:

Set Values:

Values

InputCol	Qualifier	Comparator	Value
o_entry_d	Is	Greater than	2015-01-01
o_entry_d	Is	Less than	2015-12-31

Now using a combination of the Job Variable and the Grid Variable, the same Transformation Job is running for different source tables, and filtering different dates. While this may seem like a lot of sets for a relatively simple job, the time savings when working with complex ETL jobs can be significant for your development team. You can expect faster development times, and lower overhead of maintenance of jobs since you can control changes from a single variable.

Next in the series: Creating a Shared Job

Now that you have a better understanding of Job Variables and Grid Variables you are ready to get started building Shared Jobs, which we will explore in **Part 2** of this series.

https://www.matillion.com/blog/shared-jobs-in-matillion-etl-using-variables-to-speed-up-development-part-2-of-2

Shared Jobs in Matillion ETL: Using Variables to Speed Up Development (Part 2 of 2)

*This is a two-part blog to walk you through setting up and creating a Shared Job in Matillion ETL using variables. **Part 1** walked through the set up process for Job and Grid Variables for use later in a Shared Job. In Part 2, we will look at creating and packaging the Shared Job.*

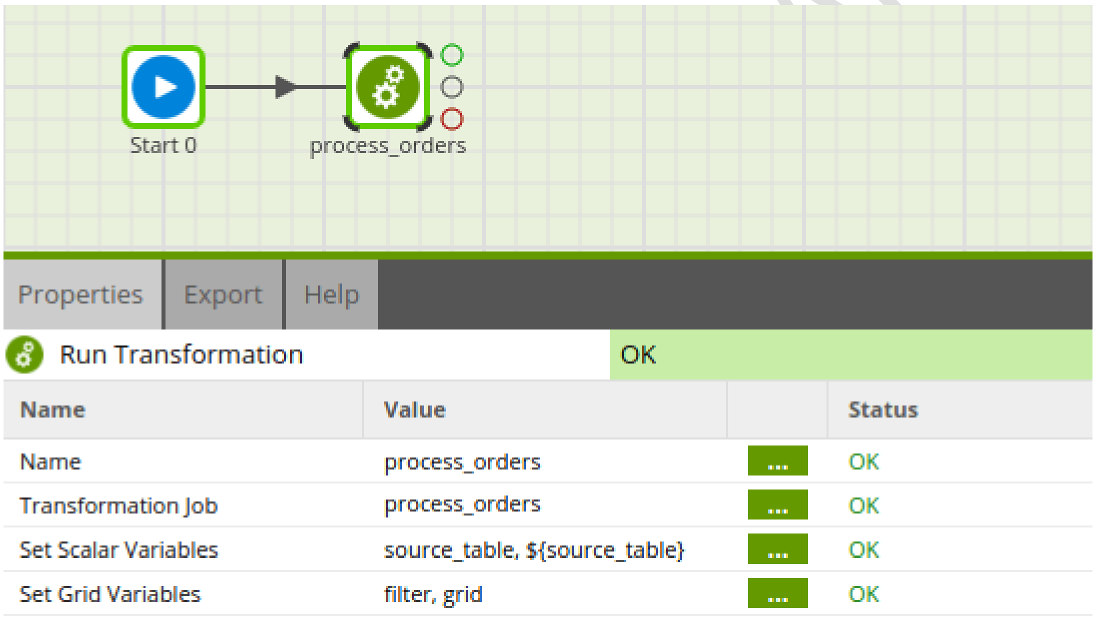
Using Shared Jobs, you can create your own components using the Shared Jobs feature. Shared Jobs bundle entire workflows – Orchestration Jobs (and the Transformation Jobs they link to) – into a custom component. In this way, Matillion ETL can help you save development time when it comes to common workflows and jobs that you will use again and again as it ultimately is a templated job to be reused across your projects.

In this blog, we build on the example in **Part 1**, to show you how you can create your own Shared Job.

Set up your Matillion job

In Part 1, we created a Transformation Job to filter our warehouse order data from three different tables. Now we want to make our Transformation Job that processes a warehouse table specified in a scalar and filters to values specified in a Grid Variable into a Shared Job. To do that, there are just two more steps we need to do:

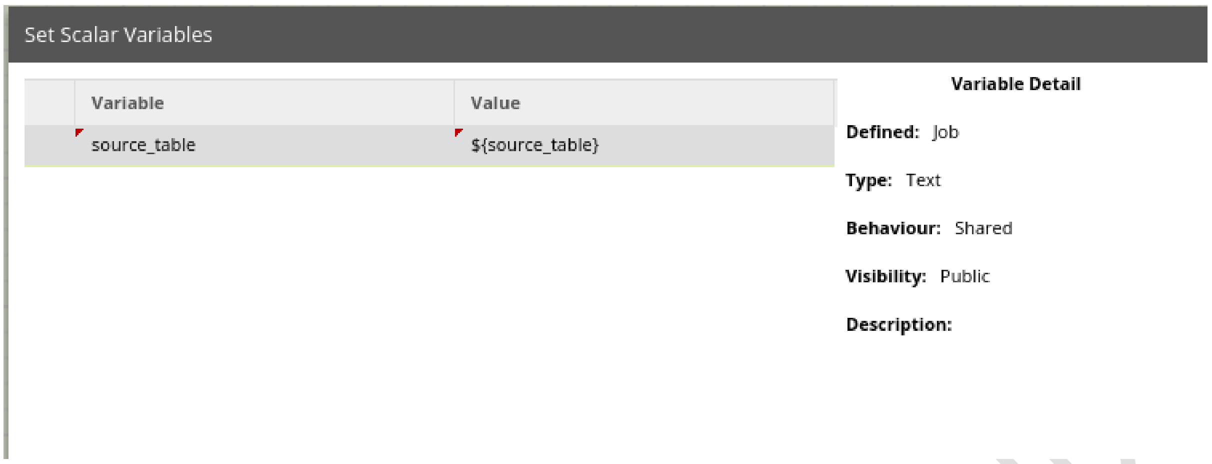
1. Since it is only possible to package up an Orchestration Job as a Shared Job, we need to make an Orchestration Job that calls our Transformation Job
2. Then we can package up our new Orchestration Job into a Matillion Shared Job (which will also bring the Transformation Job into the package)



Run Transformation — OK

Name	Value		Status
Name	process_orders	...	OK
Transformation Job	process_orders	...	OK
Set Scalar Variables	source_table, ${source_table}	...	OK
Set Grid Variables	filter, grid	...	OK

You will need to create the same Job Variable and Grid Variable that exist in the Transformation Job in Orchestration Job. This way, the Shared Job we create accepts the values as parameters, and can pass them to the Transformation Job. We also need to explicitly pass the Job and Grid Variables from this job into the process_orders sub-job. First, we can set the scalar Job Variable to reference our table variable:


Set Scalar Variables

Variable	Value
source_table	${source_table}

Variable Detail

Defined: Job

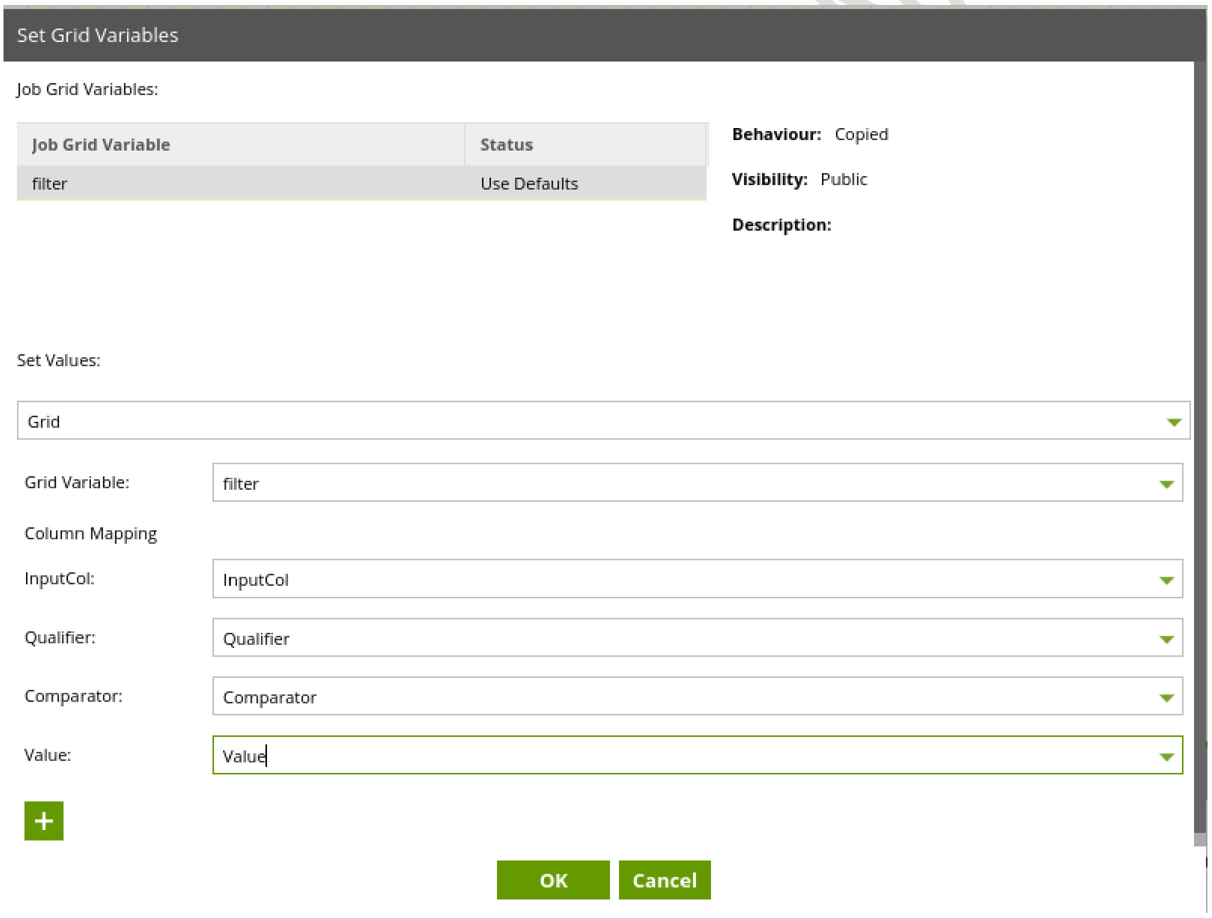
Type: Text

Behaviour: Shared

Visibility: Public

Description:

Then, we want to set the Grid Variable to reference our pre-configured Grid Variable that filters on year.


Set Grid Variables

Job Grid Variables:

Job Grid Variable	Status
filter	Use Defaults

Behaviour: Copied

Visibility: Public

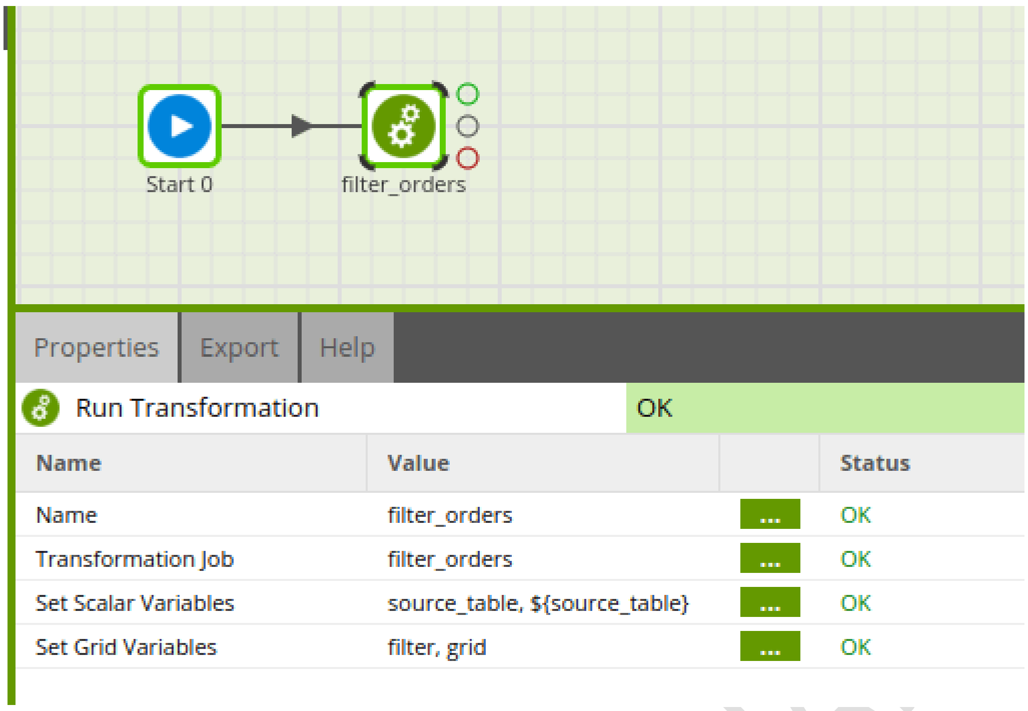
Description:

Set Values:

Grid

Grid Variable:	filter
Column Mapping	
InputCol:	InputCol
Qualifier:	Qualifier
Comparator:	Comparator
Value:	Value

Now our Orchestration Job looks like this:

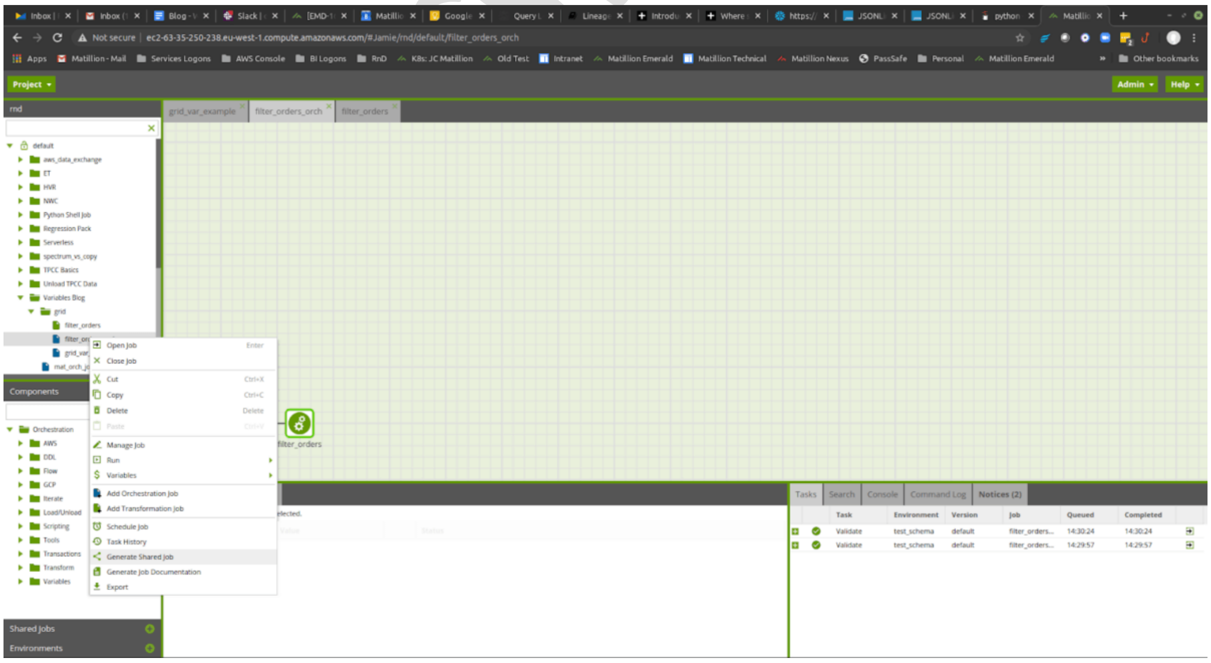


Run Transformation | OK

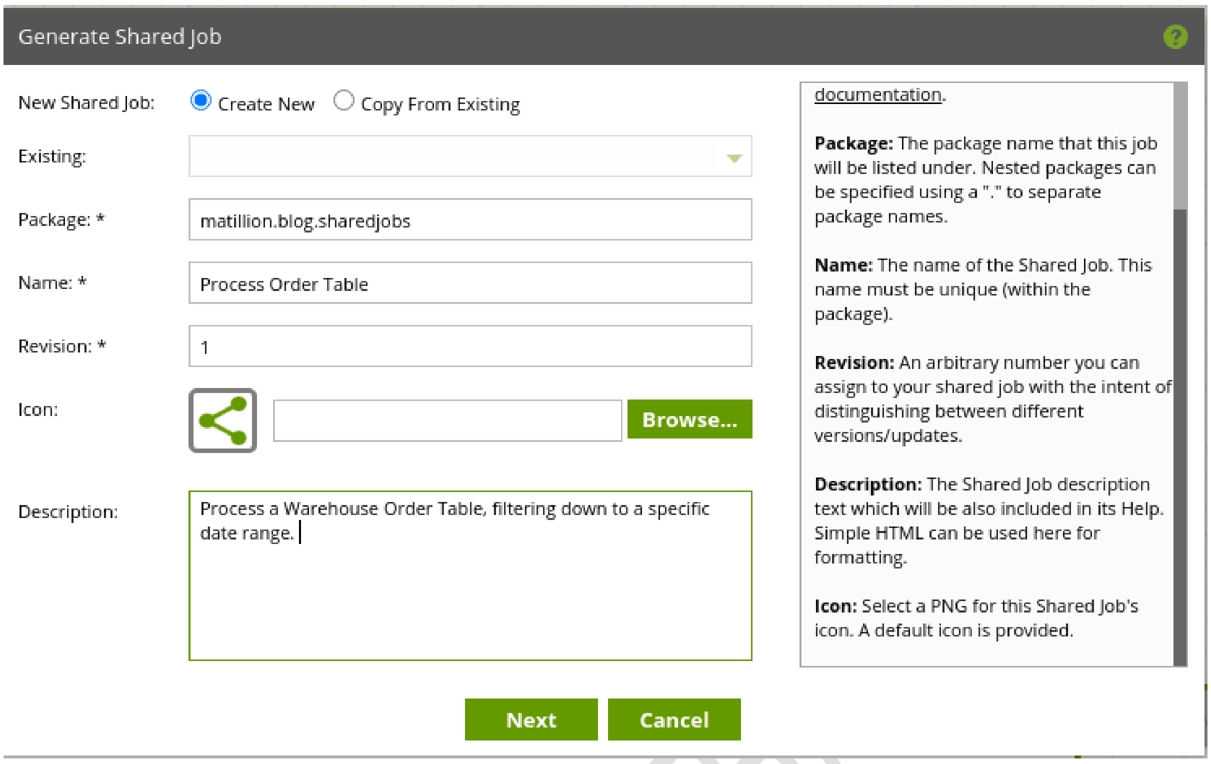
Name	Value		Status
Name	filter_orders	...	OK
Transformation Job	filter_orders	...	OK
Set Scalar Variables	source_table, ${source_table}	...	OK
Set Grid Variables	filter, grid	...	OK

Generate Your Shared Job

Now we can package the Orchestration Job as a Shared Job, find it in the Object panel, and right click and select “Generate Shared Job”



Make sure you are creating a new Shared Job, and add a package name, a Name for the shared job, and a description. You can also add your own icon, but for this example we’re going to use the default


Generate Shared Job

New Shared Job: [✓ Create New] Copy From Existing

Existing:	
Package: *	matillion.blog.sharedjobs
Name: *	Process Order Table
Revision: *	1
Icon:	
Description:	Process a Warehouse Order Table, filtering down to a specific date range.

documentation.

Package: The package name that this job will be listed under. Nested packages can be specified using a "." to separate package names.

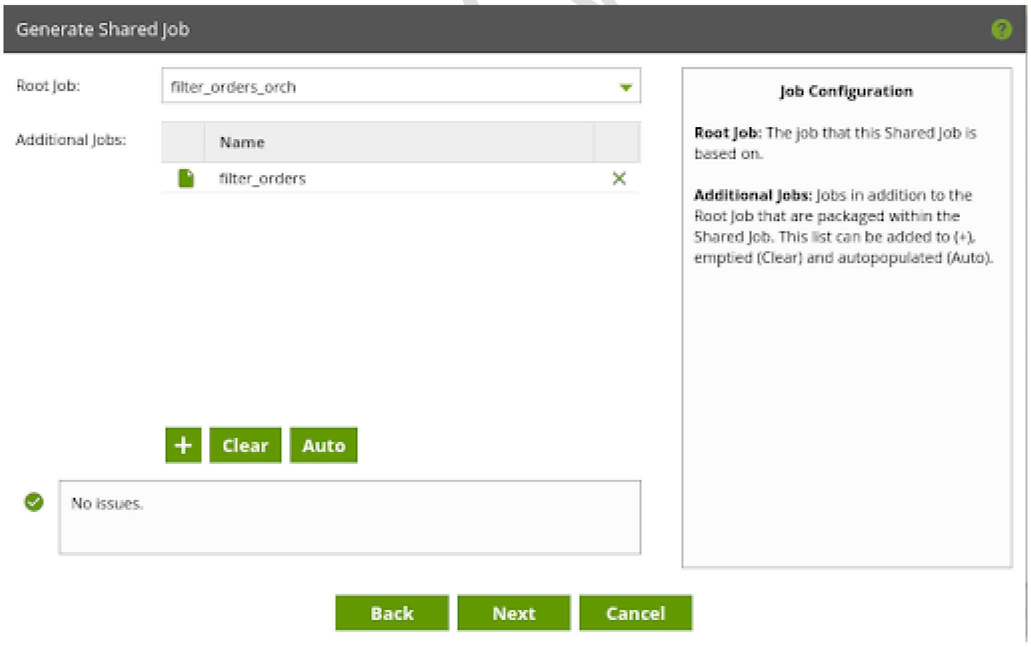
Name: The name of the Shared Job. This name must be unique (within the package).

Revision: An arbitrary number you can assign to your shared job with the intent of distinguishing between different versions/updates.

Description: The Shared Job description text which will be also included in its Help. Simple HTML can be used here for formatting.

Icon: Select a PNG for this Shared Job's icon. A default icon is provided.

Matillion ETL will automatically determine which other jobs need packaging along with the Orchestration Jobs. You can amend this list if need be, but here Matillion ETL has correctly detected our sub-job Transformation.


Generate Shared Job

Root Job: filter_orders_orch

Additional Jobs:

Name
filter_orders

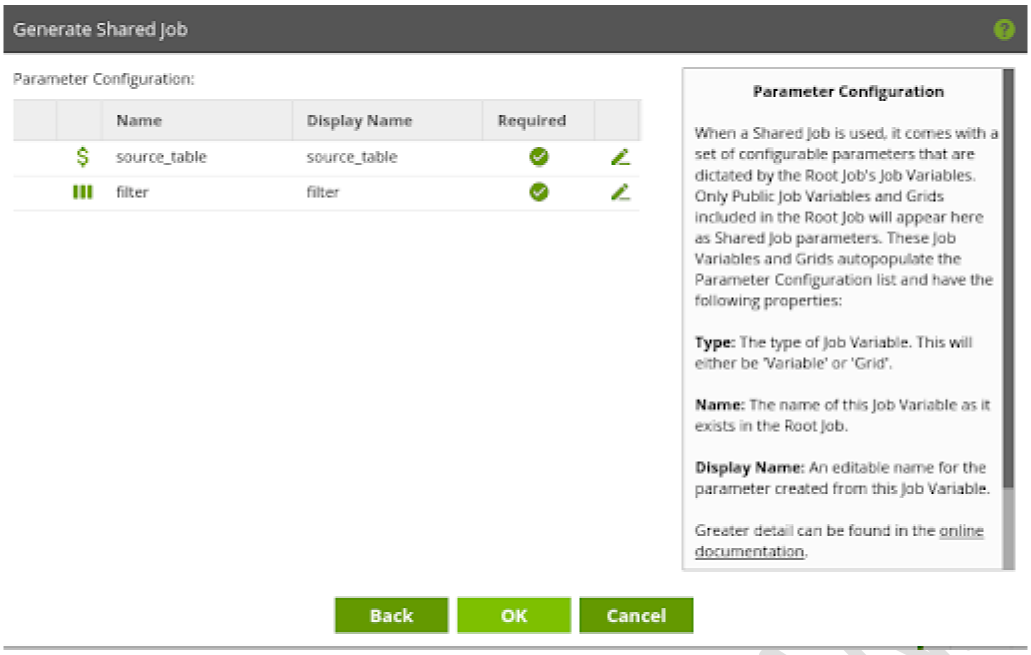
No issues.

Job Configuration

Root Job: The job that this Shared Job is based on.

Additional Jobs: Jobs in addition to the Root Job that are packaged within the Shared Job. This list can be added to (+), emptied (Clear) and autopopulated (Auto).

Here we are presented with a list of variables from the Orchestration Job we are packaging – now called Parameters. Parameters are the values that users of the Shared Job will be able to enter, and they directly correspond to the variables in the job.


Generate Shared Job

Parameter Configuration:

		Name	Display Name	Required	
	$	source_table	source_table	✓	
	▮▮▮	filter	filter	✓	

Parameter Configuration

When a Shared Job is used, it comes with a set of configurable parameters that are dictated by the Root Job's Job Variables. Only Public Job Variables and Grids included in the Root Job will appear here as Shared Job parameters. These Job Variables and Grids autopopulate the Parameter Configuration list and have the following properties:

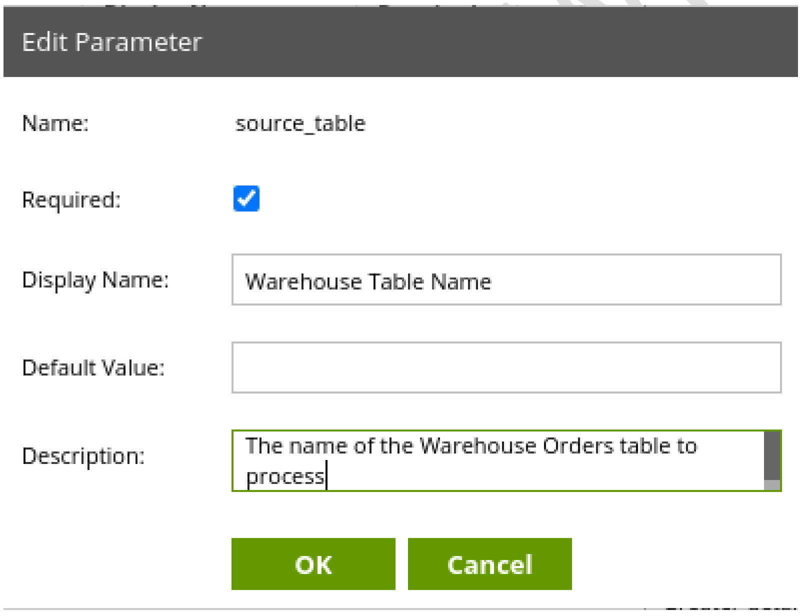
Type: The type of Job Variable. This will either be 'Variable' or 'Grid'.

Name: The name of this Job Variable as it exists in the Root Job.

Display Name: An editable name for the parameter created from this Job Variable.

Greater detail can be found in the online documentation.

By clicking on the green pencil icon, we can edit our Parameters. Here I have added a more user-friendly name and a description of the Parameter, as well as marking it as 'Required' (the Shared Job will fail to validate if a user does not supply Required parameters).

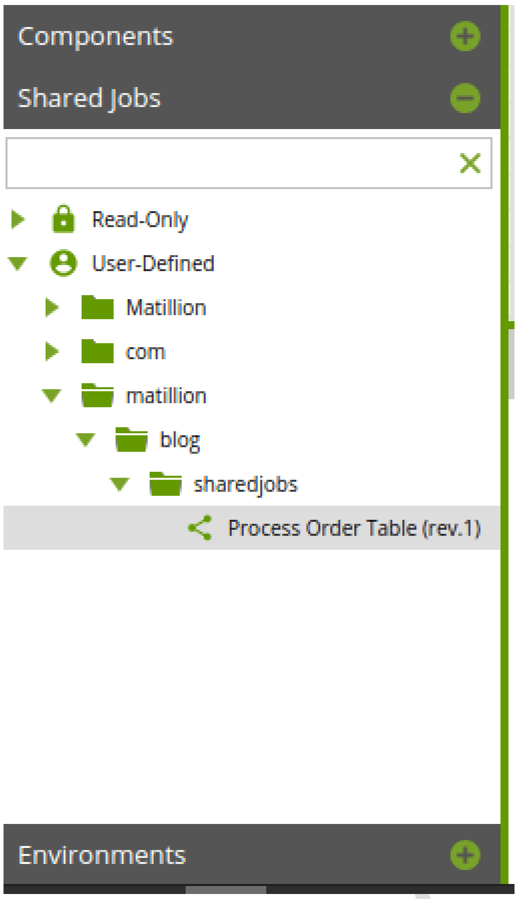

Edit Parameter

Name:	source_table
Required:	✓
Display Name:	Warehouse Table Name
Default Value:	
Description:	The name of the Warehouse Orders table to process

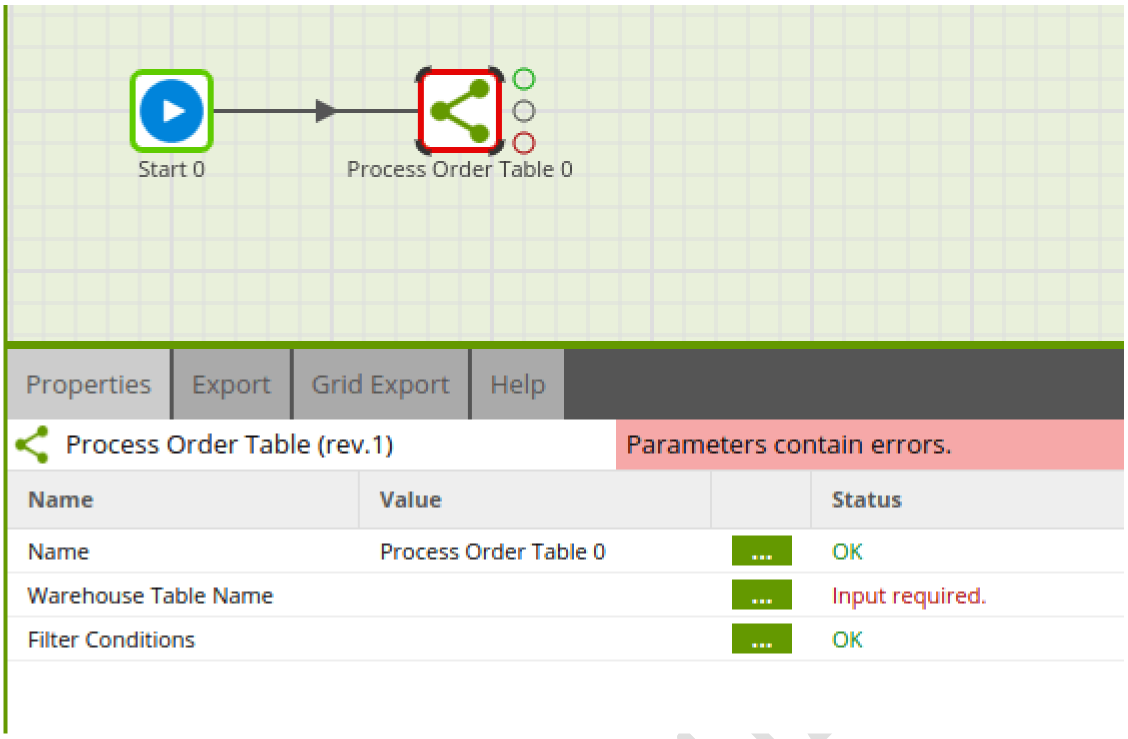
Click Ok to save changes. Then click Ok in the Generate Shared Job box to generate the job.

Using your Shared Job

To access your newly created Shared Job, find the Shared Jobs panel on the left side of the Matillion ETL user interface.



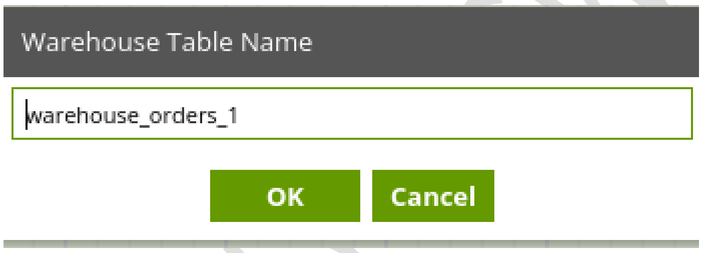
You can then drag your Shared Job onto an Orchestration Job like any other Matillion ETL Orchestration component:



Process Order Table (rev.1) — Parameters contain errors.

Name	Value		Status
Name	Process Order Table 0	...	OK
Warehouse Table Name		...	Input required.
Filter Conditions		...	OK

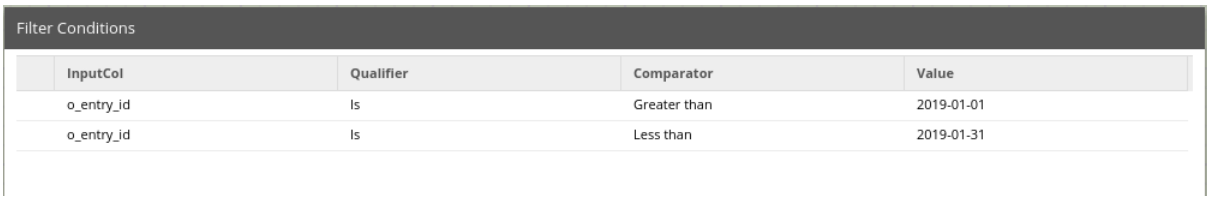
To run the job, we first need to set the properties. We can set the Warehouse Table Name to reference 'warehouse_orders_1,' which will validate the component.



Warehouse Table Name

warehouse_orders_1

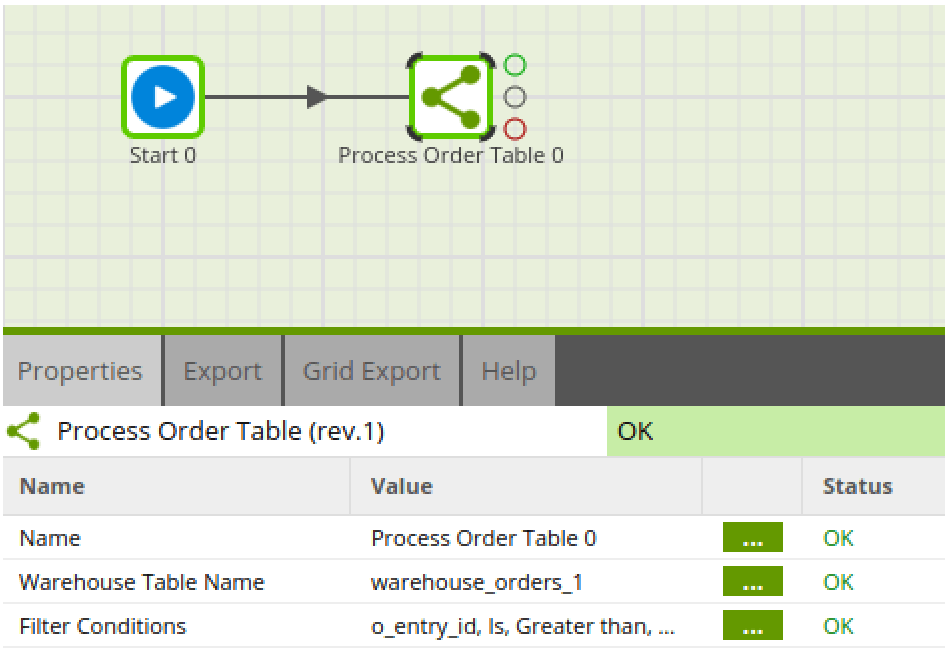
Finally, we check our Filter Conditions:



Filter Conditions

InputCol	Qualifier	Comparator	Value
o_entry_id	Is	Greater than	2019-01-01
o_entry_id	Is	Less than	2019-01-31

You have made your own Matillion ETL component!



Process Order Table (rev.1) OK

Name	Value		Status
Name	Process Order Table 0	...	OK
Warehouse Table Name	warehouse_orders_1	...	OK
Filter Conditions	o_entry_id, Is, Greater than, ...	...	OK