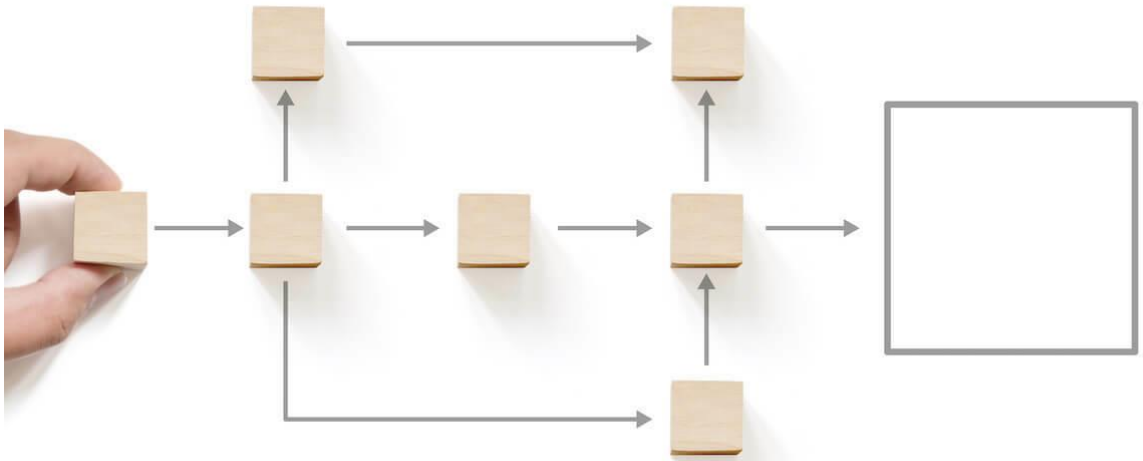
Developing Workflows in Matillion ETL



This guide describes how to manage your **Matillion ETL** workflows – known as "Jobs" – by using Orchestration and Transformation components in the most appropriate way.

Prerequisites

The prerequisites for managing workflows in Matillion ETL are:

- Access to **Matillion ETL**
- Permission to edit and run **Matillion ETL Jobs**

Orchestration and Transformation jobs in Matillion ETL

Matillion ETL is Matillion's data integration platform. Behind the scenes it uses "ELT" technology, meaning that the work of data integration gets split into two parts.

1. First, ingest data into your target **cloud data warehouse** or **lakehouse**. This is known as e**xtracting and loading** – or sometimes "staging" – the data

2. Second, do the **data transformation and integration**, **inside** the cloud data warehouse or lakehouse, using the **power and scalability of the cloud**.

This split defines the difference in purpose between Matillion ETL's **Orchestration Jobs** and **Transformation Jobs**.

WORKFLOW MANAGEMENT AND CONTROL			
EXTRACT AND LOAD DATA			
TRANSFORM AND INTEGRATE DATA			

A good way to think about this is you should use Transformation Jobs for data transformation and integration tasks, and Orchestration Jobs for everything else.

Use Transformation Jobs for data transformation and integration

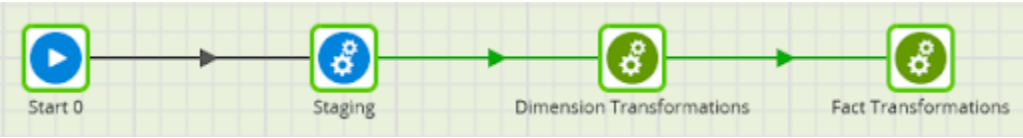
Only Orchestration Jobs can be scheduled, or launched, using **Matillion's REST API** or via one of the supported Queue services (**AWS SQS**, **Azure Queue** and **GCP Pub-Sub**).

When setting up a data integration workflow for the first time, the main tasks will be broadly like this:

- Create an Orchestration Job to manage the overall flow of control
- Add some e**xtract and load components** (also known as Data Staging)
- Create Transformation Job(s) to transform and integrate the newly loaded data
- Add the Transformation Job(s) to the Orchestration Job

To add a Transformation Job to an Orchestration Job, use a **Run Transformation component**. You can also have an Orchestration Job launch another Orchestration Job, using a **Run Orchestration component**.

For a simple **two-tier data architecture**, where you create **star schema facts and dimensions** directly from staging tables, your main Orchestration Job would end up looking something like below. The screenshot is from the "Late Arriving Dimensions" area in the **Developer Relations Example Jobs**, if you want to try it out yourself.



The last step with any Matillion ETL workflow is to make sure the components are valid

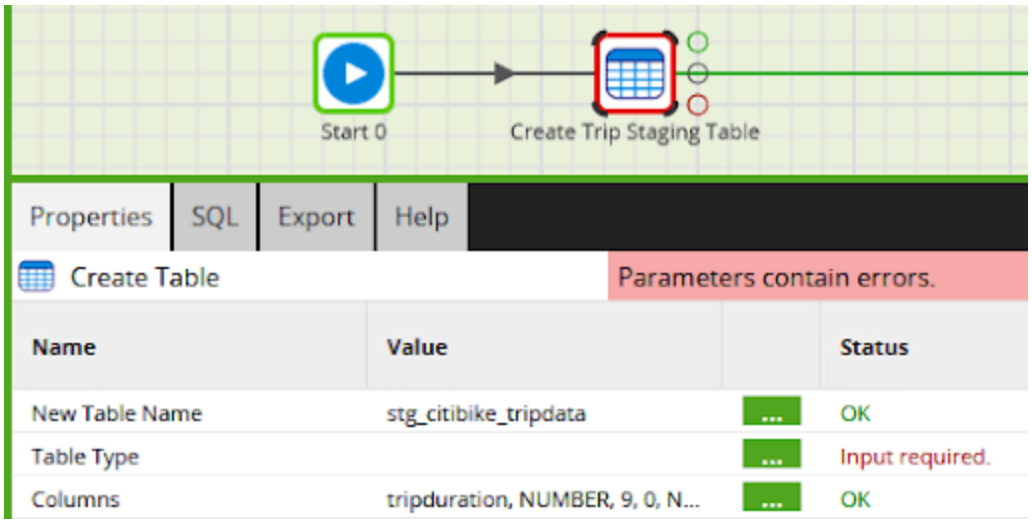
Component Validation

When they are valid, components are displayed in the job editor with a green border. When they are **not** valid, they have a gray or a red border.

- A **gray** border means validation has not been attempted yet. This can happen when you first open a job in the design editor. It also happens when **switching environments**, and when using undo or redo
- A **red** border means there is definitely a validation problem
- A **green** border means the component is valid
- You may briefly see an **orange** border, which means that component is in the process of being validated

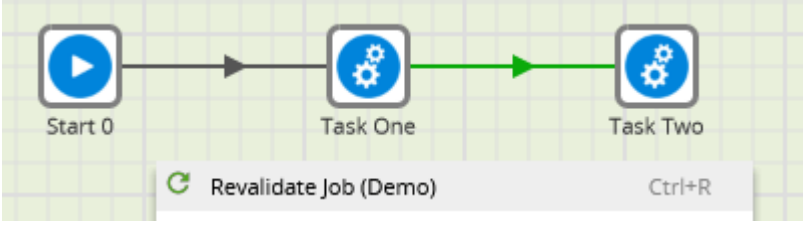
Some functionality – such as data sampling – will not work unless the component is in a valid state, with a green border.

While editing a job, select any component to view and edit its properties. In general, if there is a validation problem, it will be highlighted in red among the properties.

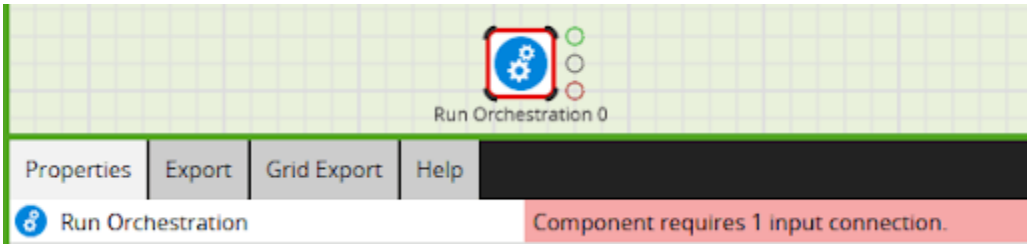


Changing any of the properties will make that single component revalidate.

To revalidate **all** the components in the job, choose “revalidate” from the context menu of the job canvas:



It is good practice to revalidate at design time. But validation also occurs at runtime. This handles situations where one component makes a database change – such as creating a table – that a later component requires. At design time, the job probably will not validate cleanly, but the runtime validation ensures that everything will execute correctly. This **statefulness is an aspect of DataOps**.

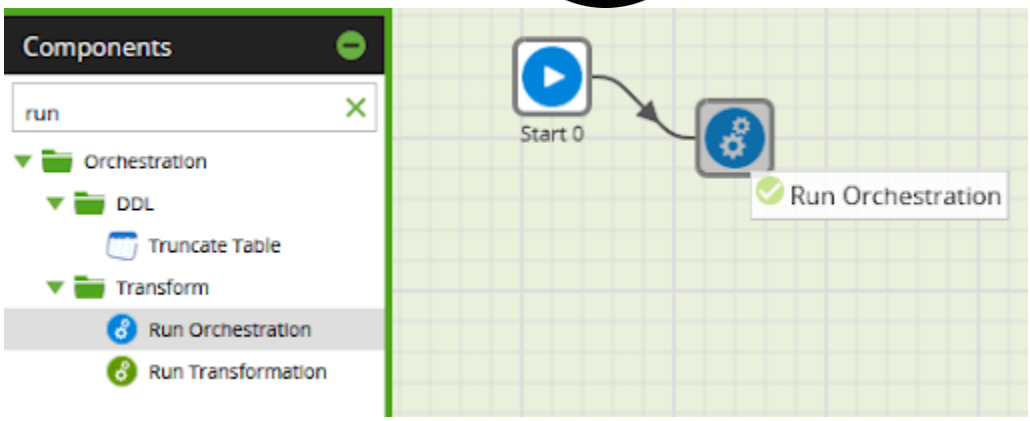


The links between Matillion ETL components define the flow of control in an Orchestration Job.

How to link Matillion ETL components

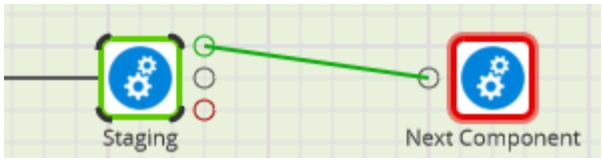
While you are editing a job, you can drag a new component from the panel on the left and drop it on the canvas. If you drop it near another component, and to the right, they will link automatically, like this:

Components will **not** validate without an input connection that links them to a predecessor.



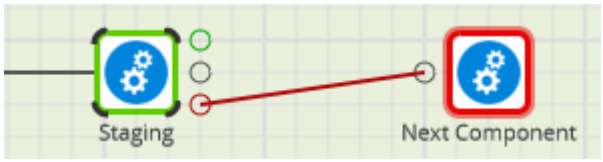
A default option is chosen if you link components by just dropping them near each other. If instead you select a component, you can drag out a link from one of three available connectors. Release the link on top of another (currently un-linked) component.

If you choose the green success connector at the top, the next component will run **if this component runs successfully**.

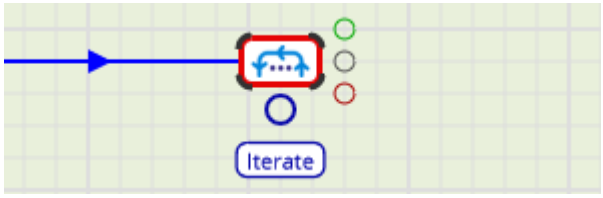


If you choose the black unconditional connector in the middle, the next component will **always run, regardless if this component succeeds or fails**.

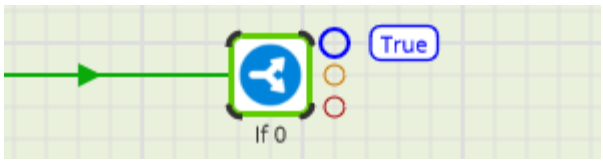
If you choose the red failure connector at the bottom, the next component will run **if this component fails**. This is a good way to add error handling.



All Matillion's **iterator components** have one extra connector at the bottom. Connect this to another component, and it will be run multiple times – once per iteration.



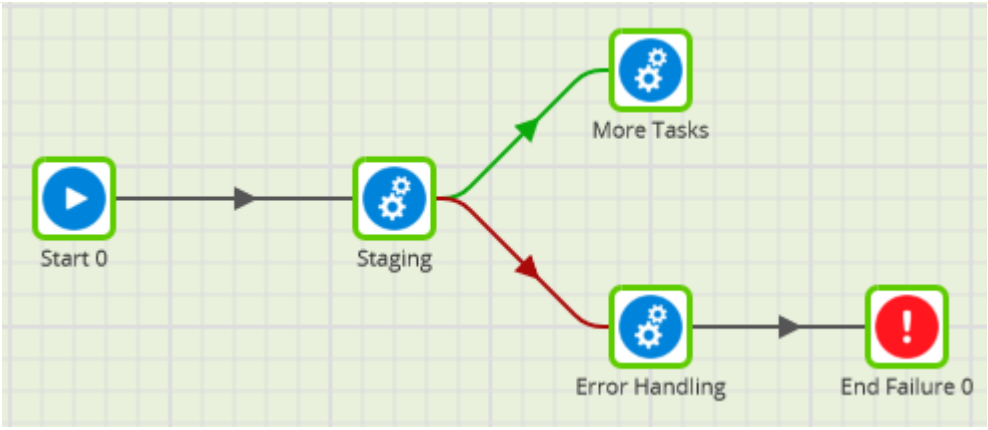
The **If Component** is slightly different, having true, false and failure connectors. It is used to branch conditionally within an Orchestration Job.



You can use more than one of the output connectors if you want. A common pattern is shown below:

- A component (Staging) does something that must happen before the More Tasks component
- The More Tasks component is linked using the green, success connector. It will only run if Staging finishes successfully
- If Staging hits an error, the job will branch along the red failure connector instead, and will run the Error Handling component

- Assuming the Error Handling component succeeds, the job would be marked as a success. Often that is undesirable, so a final End Failure component is linked using a black unconditional connector



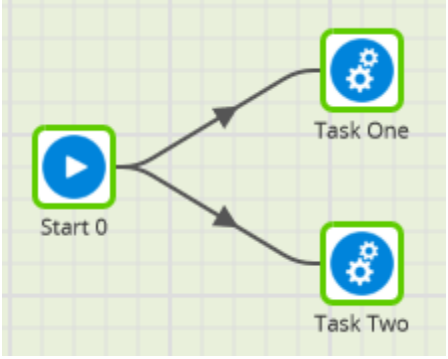
Using more than one green or black connection is the way to make a Matillion ETL Orchestration Job branch into parallel threads.

Parallelism and Threading in Matillion ETL Orchestration Jobs

In a Matillion ETL **Orchestration** Job, all component invocations are synchronous. The flow of control passes to the next component(s) after the component has finished running.

In the screenshot below, Task One and Task Two will run **in series** – one after the other:

In the screenshot below, Task One and Task Two will run **in parallel**, in two different threads:



Depending on your Matillion configuration, there may be limits on how many parallel threads can be running at any one time. If you launch more threads in parallel than are available, some threads will pause until others have finished.

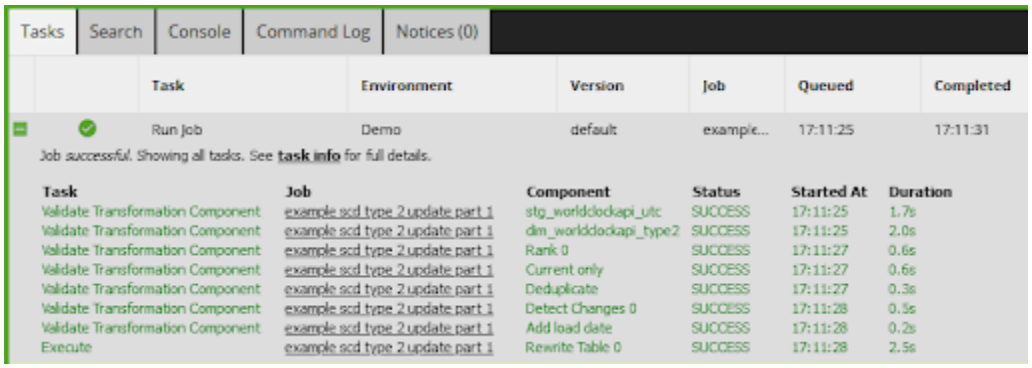
The **"And" and "Or" components** can be used to re-synchronize multiple parallel threads back together.

Parallelism and Threading in Matillion ETL Transformation Jobs

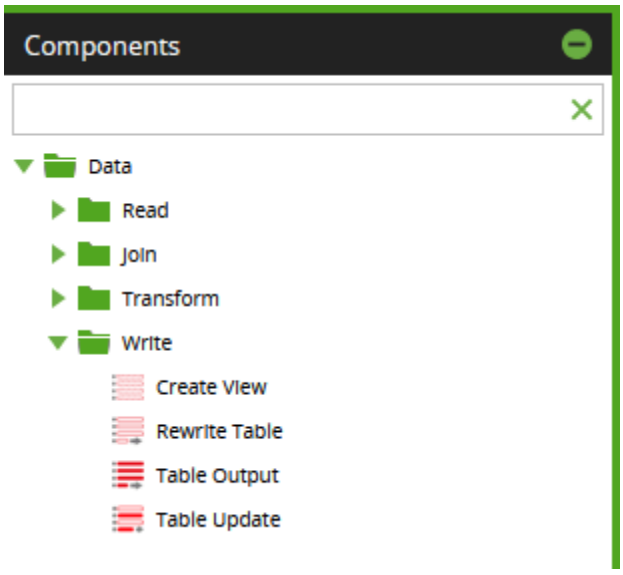
In a Matillion ETL **Transformation** Job, parallelism and threading work in a very different way. Because it is "ELT" technology, transformation job components are combined to **generate code** that is pushed down onto the target cloud data warehouse or lakehouse for execution. All the components in a transformation job run **at the same time**.

For this reason, there are no thread control components among Matillion ETL's transformation components. All the components are already running in parallel.

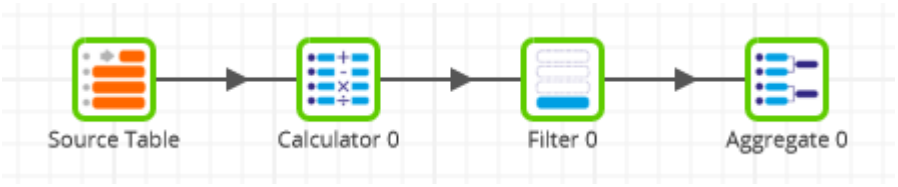
It can sometimes look like transformation job components are running one after the other. Below is a screenshot from the task history of a successful Transformation Job run. The first seven tasks are **validation** steps. The only time the components actually run is in the final Rewrite Table step, in which **all eight components run at the same time**.



There is one small clarification to the above statement. Only transformation components taken from the “Write” area generate code that executes. Components selected from elsewhere are combined together and accumulate into a single Write operation.



As an example, the below Transformation Job contains **no** components from the Write area, so it will do absolutely **nothing** when it is run – except take a finite time to validate. This kind of job is nevertheless useful during design and development, because you can use the Sample tabs to view data.



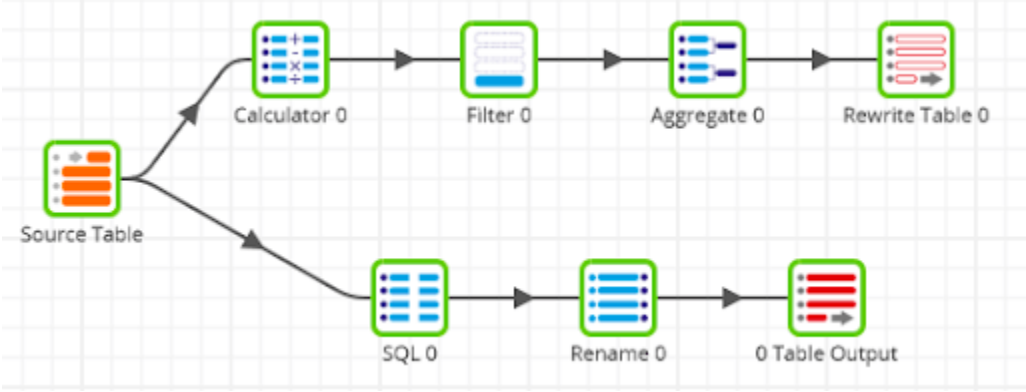
The next Transformation Job, below, contains **one** component from the Write area – a Rewrite Table – so it will do **one** thing at runtime. All five components will execute **at the same time**.

The last example Transformation Job, below again, contains **two** components from the Write area in total – a Rewrite Table and a Table Output – so it will do **two** things at runtime:

1. The Source Table query, Calculator, Filter and Aggregate will all run **at the same time** as a Rewrite Table operation (which is a CREATE TABLE AS SELECT in SQL)

2. The same Source Table query again, freehand SQL and Rename will all run **at the same time** as a Table Output operation (which is an INSERT in SQL)

You can not influence the order in which the two SQL statements run: it is not deterministic. Consequently if you are using multiple Write components, you must not write to any of the tables that are being used as inputs.



It is usually simplest to understand and maintain if you stick to using one Write component per Transformation Job.

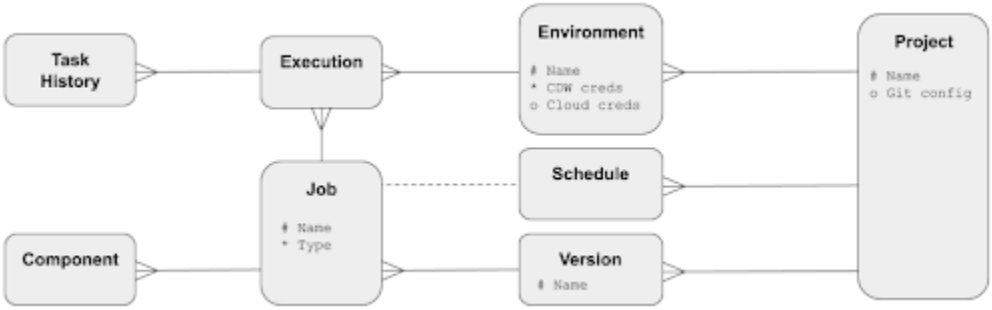
Use only one Write component per Transformation Job

Workflow metamodel for Matillion ETL

Orchestration and Transformation Jobs are versioned within Matillion ETL projects. Versioning is used by the optional **git integration**.

Jobs always execute within the context of exactly one **Environment** object. That is one target **cloud data warehouse** or **lakehouse** connection, and one set of cloud provider credentials.

When a job finishes executing, its history is saved both at job level and at component level.



Next Steps

It is easiest to develop a workflow using hardcoded values. Then switch to using **variables** once it is known to be working.

Use full data volumes if possible while testing your workflows. Afterward, you can add Matillion DataOps features including:

- **Promote between environments**
- **Schedule your jobs**
- **Add error handling and logging**
- Use the **Task History Profiler** for performance analysis and tuning
- **Choose the data tiers to deploy**
- **Decide which Data Model to use**
- Think about **history tracking and time variance**

Reference : https://www.matillion.com/blog/developing-workflows-in-matillion-etl