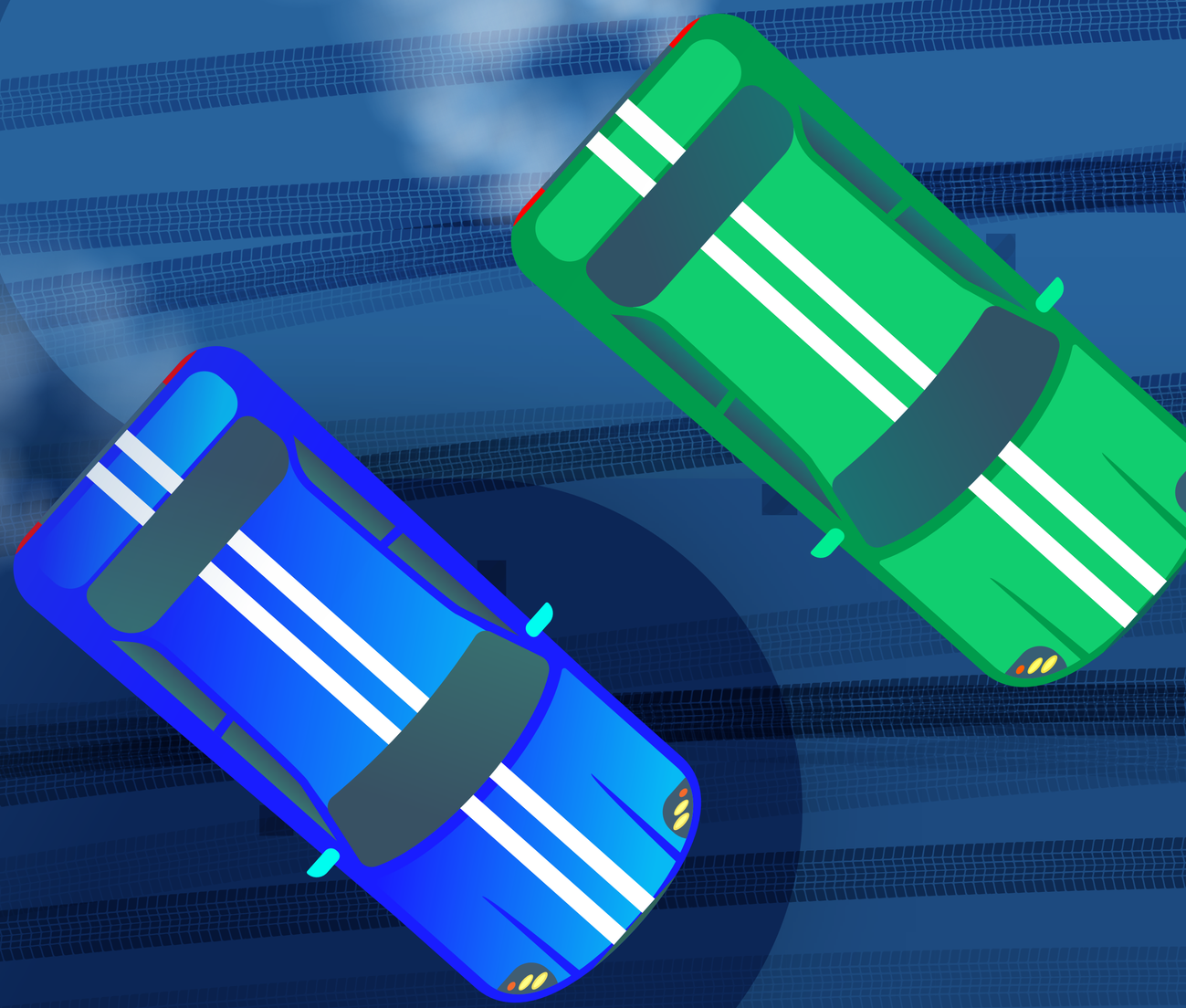
EBOOK

# Solving Schema Drift in Matillion

## 3 practical approaches to ensure data integrity



matillion

# Introduction

Data integrity often faces threats from unexpected changes in data structures, known as schema drift. This eBook, **“Solving Schema Drift in Matillion: 3 practical approaches to ensure data integrity,”** delves into essential strategies to tackle this persistent issue. Offering a detailed exploration of practical solutions and methodologies, this guide helps you design robust data pipelines that seamlessly adapt to schema changes. Whether dealing with Excel files, S3 files, or database tables, this eBook  provides the tools and knowledge to maintain accuracy and consistency in your data processes. By the end of this book, you’ll be equipped to handle schema drift with confidence, ensuring your data remains reliable and your operations uninterrupted.

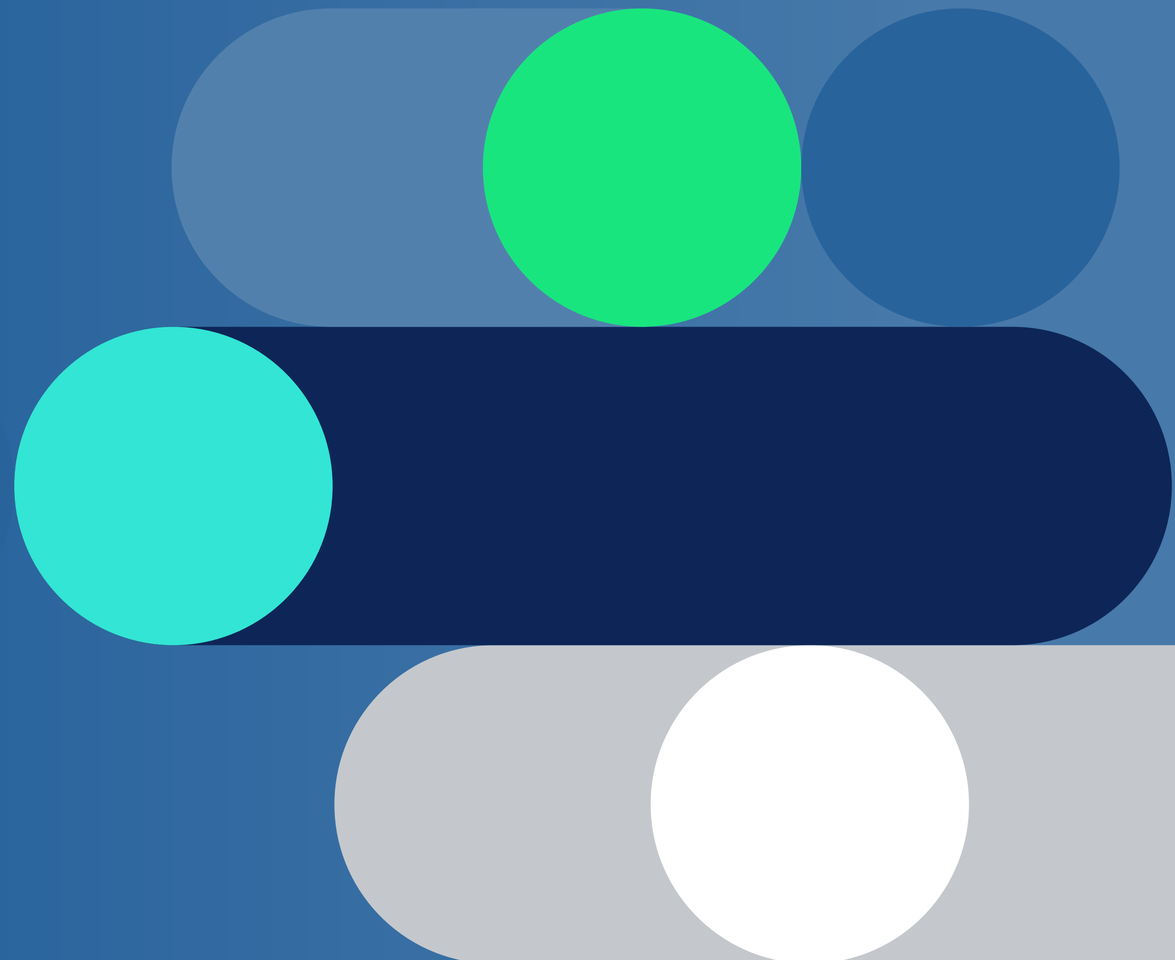
# Contents


01 Schema Drift for Excel files with Matillion's Data Productivity Cloud 4

02 Schema Drift and Dynamic S3 Files 11

03 Schema Drift on database tables 23

04 Conclusion 29

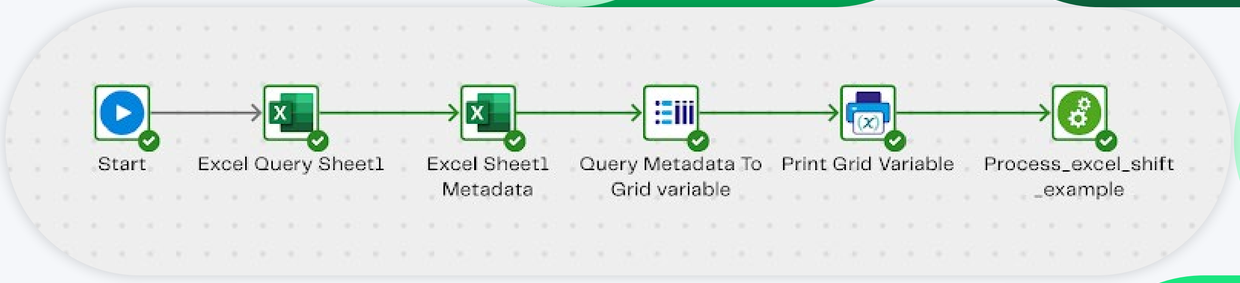


01

# Schema Drift for Excel Files

# This chapter discusses the challenges and solutions for managing schema drift in Excel files using Matillion Data Productivity Cloud.

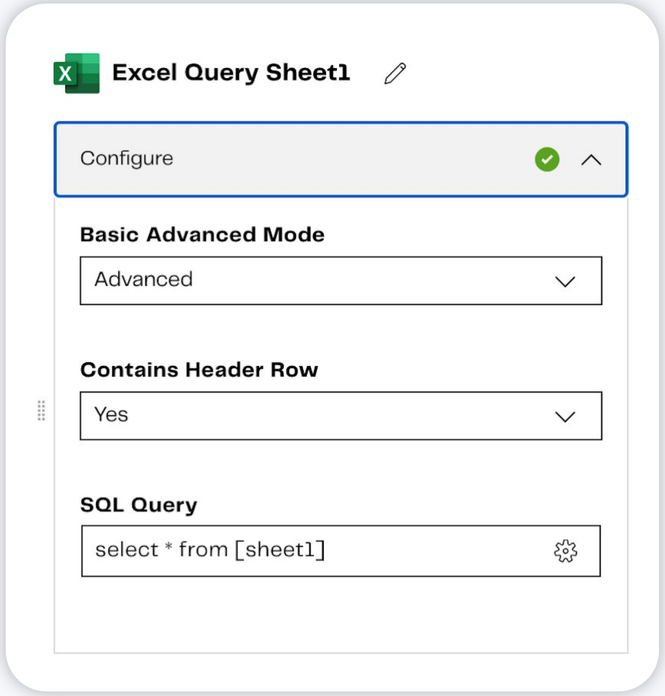
One of the biggest challenges data teams encounter today is the constant evolution of source data. Schema drift occurs when you have defined data sources and data pipelines built out and then the source changes. Either new columns are added, an existing column is renamed, or the data type changes. These changes can often break the pipeline and send the data engineering team into a scramble to find the problem. One common data source is files, specifically Excel files. This pattern will show how to design a pipeline specifically for Excel files that may change over time. This pipeline shows the pattern for capturing those file changes, also called schema drift.



01

## Excel Query Component - initial query

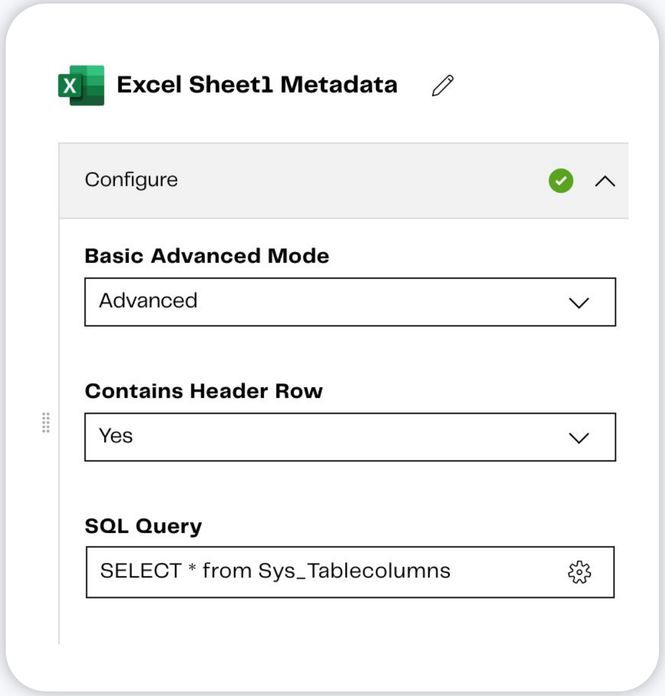
The Excel query component has an advanced query mode that enables an SQL query similar to tables. In this example, you can create an SQL query to return all of the columns within a sheet. SQL Query: Select * from [Sheet1]



02

## Excel Query Component - 2nd query accesses the sheet metadata.

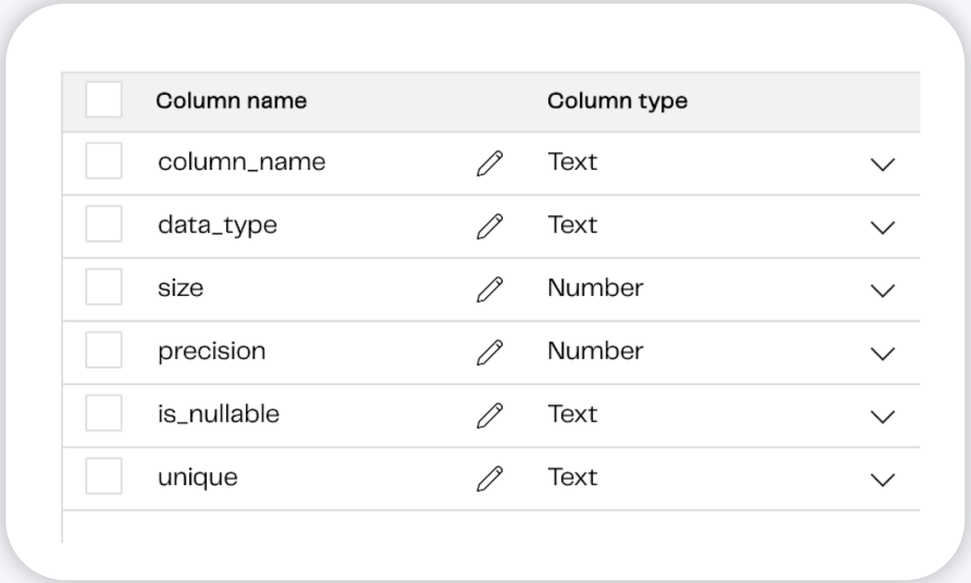
This query accesses the metadata available for the Excel sheet. This component has some system tables which contain this metadata. Query the sys_tablecolumns table to get the sheet metadata and write the metadata to a table in Snowflake.



03

## Create a grid variable to store the Excel sheet metadata.

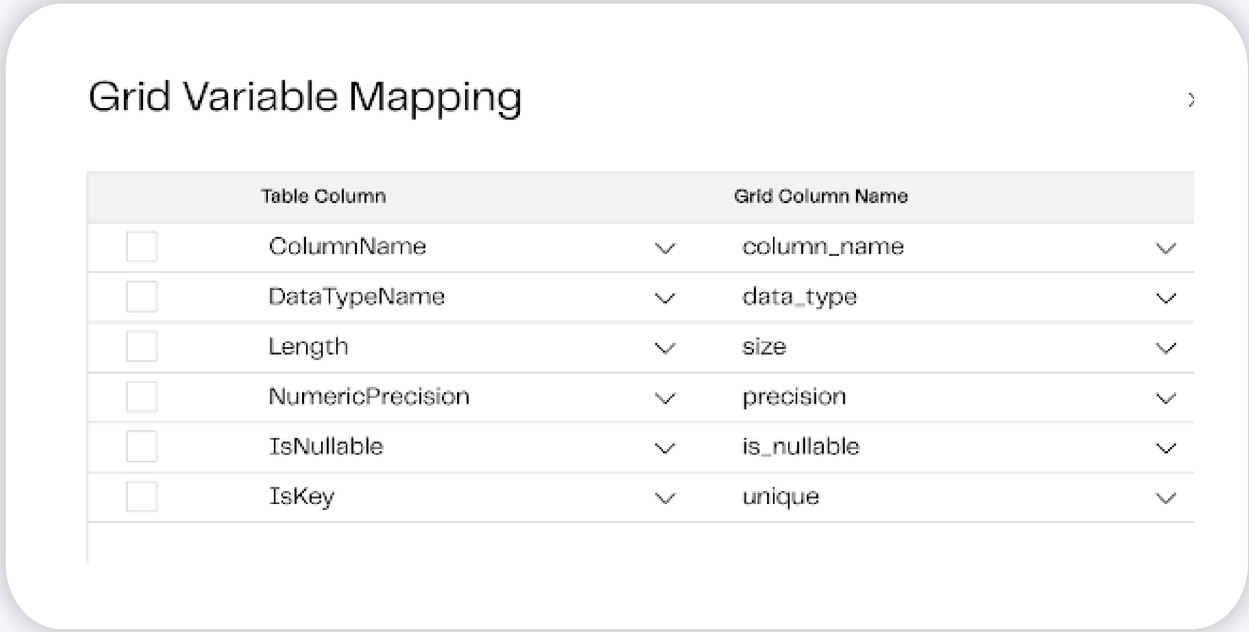
A grid variable is a 2-dimensional variable that can be used to store and loop thru multiple value pairs. This grid variable is used to store the metadata defining the Excel sheet.



04

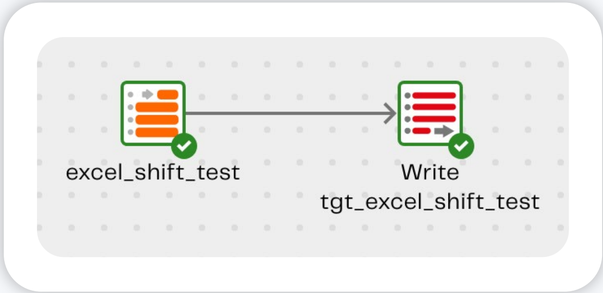
## Query result to Grid

Now that the metadata has been stored in a Snowflake table, use a ‘Query result to Grid’ component to select the metadata columns from the snowflake metadata table and store them in the grid variable.



05

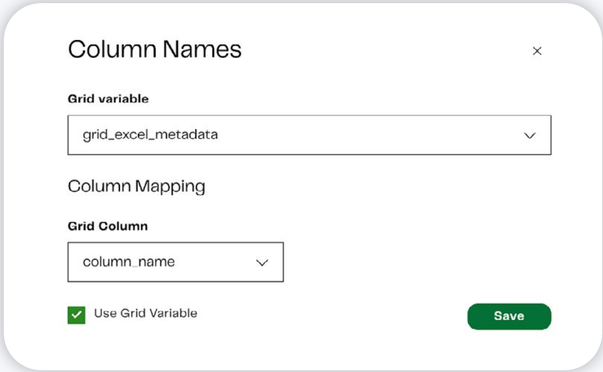
This grid variable can be passed into the transformation job to define the raw table and write it out to a target table. When defining the input table, use the grid variable to define the table layout dynamically.




Use the grid variable to define the columns in the input table.

Any further transformations downstream can also be created dynamically by querying the Snowflake metadata table “INFORMATION_SCHEMA”.columns table.

**Note: The ‘Query result to Grid’ component is only available in Orchestration pipelines. The grid variables and query of metadata have to occur in the orchestration and then be passed into transformations.**





02

# Schema Drift and Dynamic S3 Files

Now that we've navigated the intricacies of schema drift in Excel files in Chapter One, this chapter explores schema drift and the dynamic world of S3 files. It demonstrates how to design data pipelines using Matillion Data Productivity Cloud to adapt to changing data source formats. This chapter will detail strategies for handling various file formats dynamically and managing schema changes in S3 files effectively.

The design pattern is for Snowflake, and requires that your S3 files are of CSV format, using any delimiter, and have a header line defining the data columns in the file

## Dynamic S3 File Loading

To load data from an S3 file into Snowflake, you need to create a table first. This is challenging when you have a large number of files with different schemas or file schemas that are changing. This design pattern is a dynamic pipeline that defines the table before loading the file and is purpose-built for a large number of files.

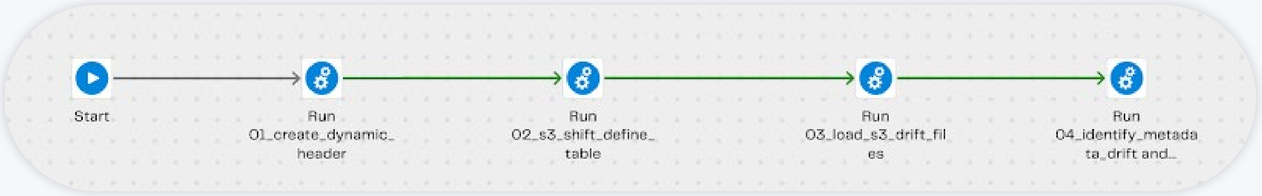
### The dynamic S3 file design pattern has 4 steps to it.

**01** Create a custom file format and a table with a single data column

**02** Parse the S3 file header and create a new table with the correct file schema

**03** Load the S3 file into the new stage table

**04** Compare the stage and target tables to identify metadata drift

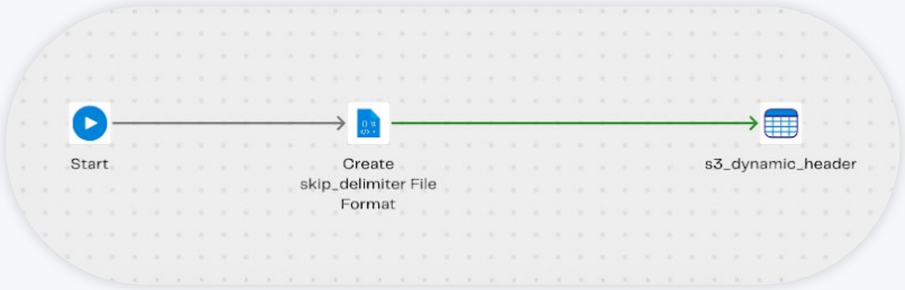


01

## Create a custom file format and raw data table

- The first component creates a custom file format that defines the field delimiter as null. Defining the file format with a null delimiter causes all data to be loaded into a single column. If the file delimiter is not a comma, you may need to create another file format with the correct delimiter.

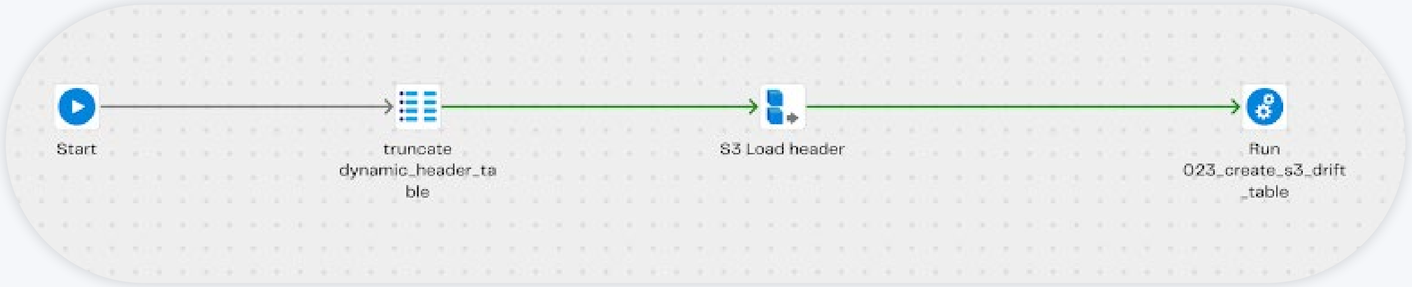
- Create the first landing table to parse the header (s3_dynamic_header). This table will contain a single varchar column with a maximum length (16Mb). This table is used to stage the S3 file and extract the header row.



02

## Parse the S3 file header and create a new table with the file schema

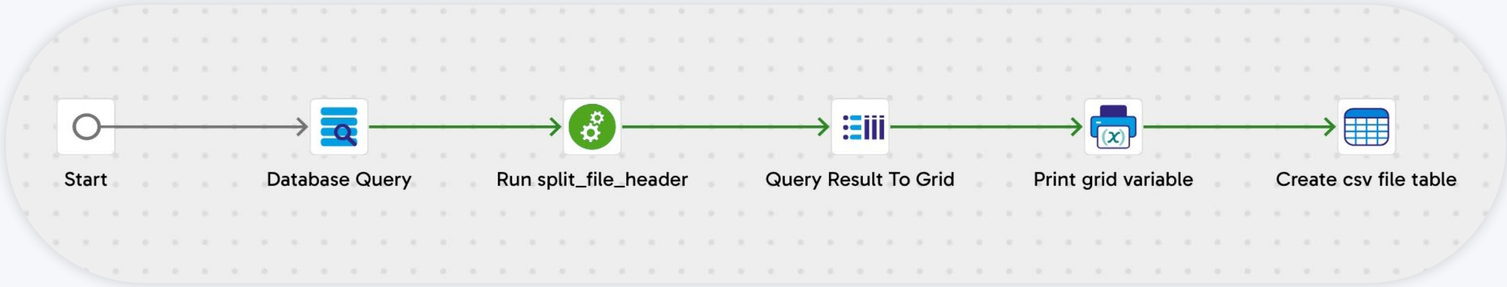
- Truncate the dynamic header table - since this dynamic table gets used for every file, this ensures that the current file being processed is the one staged into the dynamic table to extract the header.
- Load the S3 file into the dynamic table that was created in the first step. This should load all of the data into a single column with no delimiter.
- Run a child pipeline ‘023_create_s3_drift_table’, that parses the header and captures the S3 file met



2a

## 023_create_s3_drift_table

- The first component queries the dynamic table and writes only the header row to a new table.
- Then, a child transformation called ‘split_file_ header’ parses the header and rotates the columns to rows to build the metadata defining the final stage table for the S3 file.
- The query result to grid is querying the table that contains the metadata from the split header and stores it in a grid variable to be used when defining the new table.
- Create csv_file_table creates a new table with all the columns from the header and the metadata that was created on the split_file_header transformation pipeline.



2b

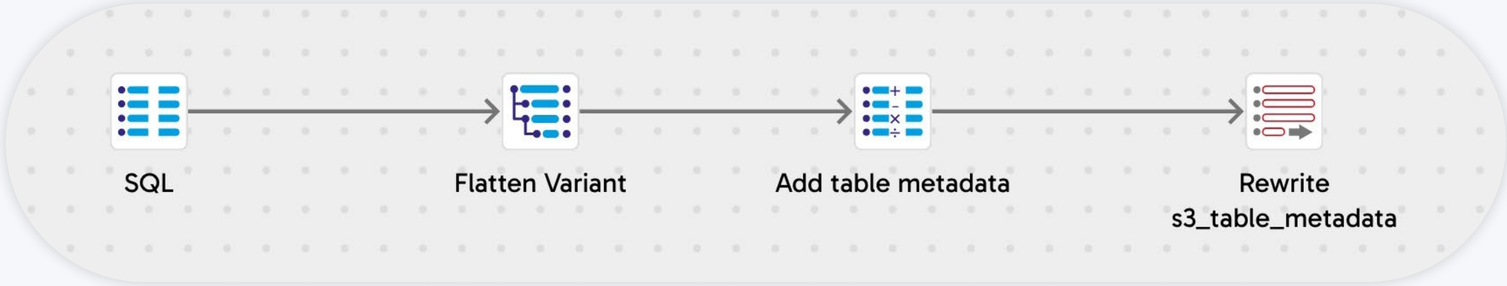
## split_file_header

- In the child pipeline ‘split_file_header’, the header now stored in a new table, is parsed with a split() function in Snowflake. This component is a SQL component to utilise the split function.
- The SQL component has the following SQL:

```
select SPLIT(“header”, ‘${pipe_custom_delimiter}’) as split_header
from “${proj_default_schema}”.”${pipe_dynamic_header_table}”
```

- This query will result in an array of column names. Example:

```
[ “First_Name”, “Last_Name”, “Street Address”, “City”, “State”, “Country”, “Postal code” ]
```




Flatten variant - this component will turn the array of column names into different rows:

Flatten Variant | Sample data

| header |
| --- |
| First_Name |
| Last_Name |
| Street Address |
| City |
| State |
| Country |
| Postal code |

Add table metadata is a calculator component that creates metadata for each column. This component makes every column a varchar(16777816). The expressions correspond to the metadata that is required to create a table. These are the same fields that are in the grid variable.

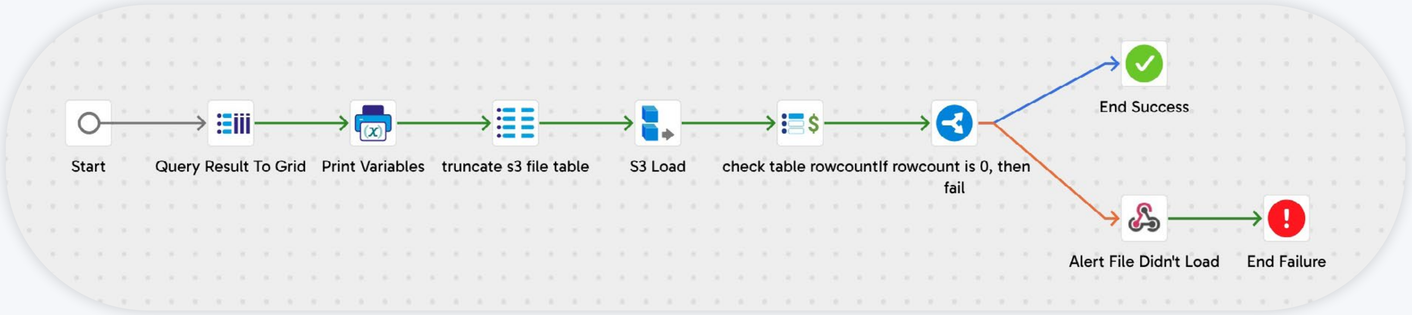
Calculations

| Expressions | data_type |
| --- | --- |
| data_type → | 1 'VARCHAR' |
| size | |
| precision | |
| not_null | |
| default_value | |
| unique | |

03

## Load the S3 file into the newly defined stage table

- Query result to grid queries the newly created S3 stage table to get the columns.
- Then truncates the S3 stage table to make sure the new file is loading to a clean table.
- It loads the S3 file into the stage table, using the custom file format if the delimiter is different from a comma.
- Next, the new table is checked to make sure that some data was loaded before proceeding. If the rowcount is 0, then the job ends in a failure. If the rowcount is greater than zero, this pipeline ends and is passed to next check if the metadata is the same as the previous run.

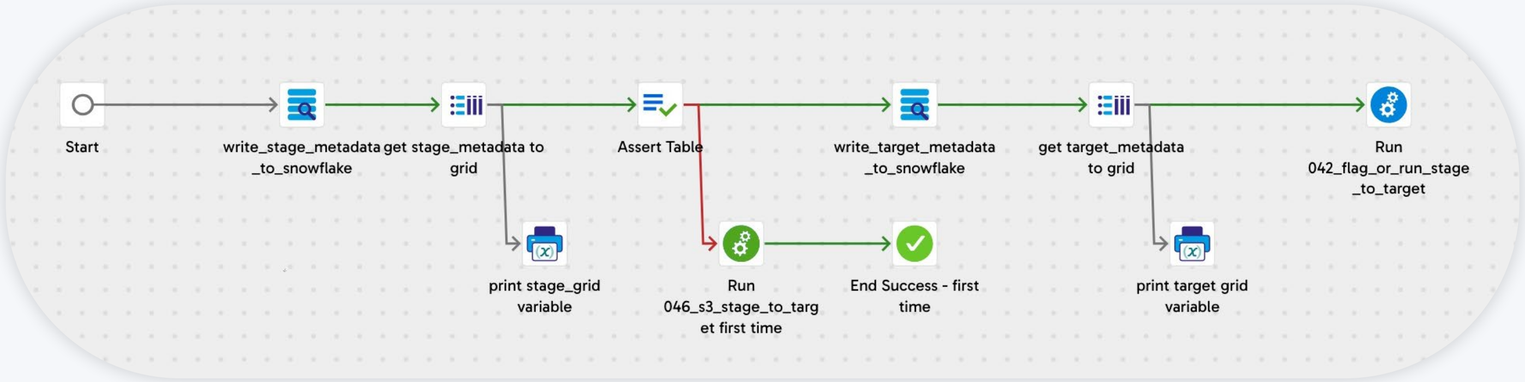


04

# Identify schema/metadata drift or run the S3 stage to target transformation

**Now the data has been loaded, it's time to check for schema drift.**

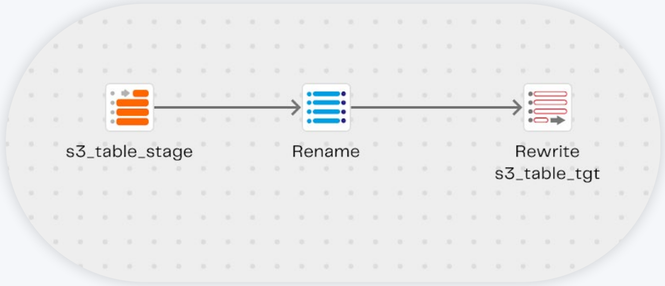
- The first step queries the S3 stage table and then saves the schema into a grid variable.
- Next, it checks to see if the target table exists. If it doesn't exist, it runs the '046_s3_stage_to_target' for the first time and ends successfully.
- If this S3 file has been processed before or the pattern is to process the same S3 file daily for instance, then the target table schema is queried and saved to a grid variable.
- The last step in this process is to run the child pipeline '042_flag_or_run_stage_to_target'.



4a

## 046_s3_stage_to_target

- The dynamic S3 file pattern will handle changing schemas until it is time for the stage data to be written to the target table. Typically these stage to target transformation pipelines will contain business logic and will be transforming the data to a target model purpose built for an analytic use case.



4b

## 042_flag_or_run_stage_to_target

For this step, we can identify if there has been a change to the stage table and exit the pipeline gracefully with a notification that a change is required.

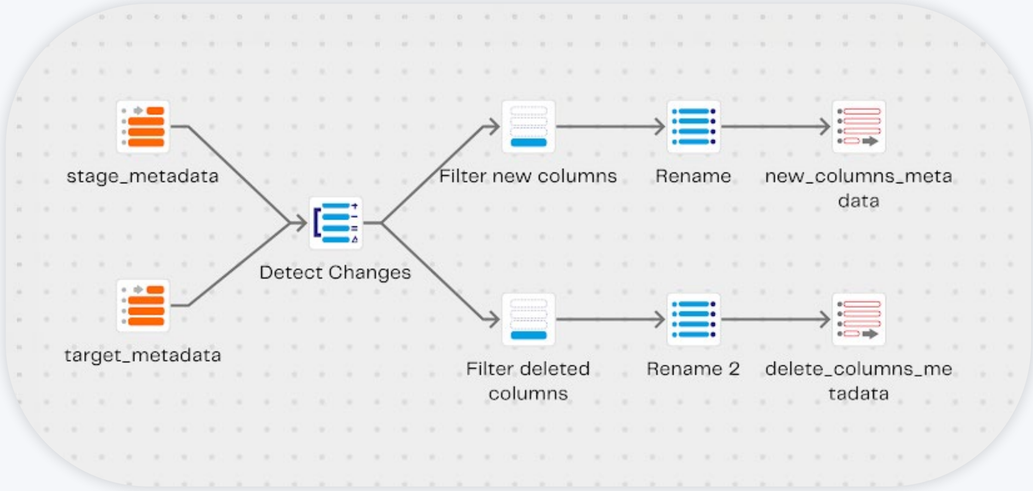
- Run 045_compare_source_target_metadata
- Count rows from new_columns_metadata - queries the new_columns_metadata table for a row count and saves the result into pipe_changes_needed variable
- Count rows from delete_columns_metadata- queries the delete_columns_metadata table for a row count and saves the result into pipe_changes_needed variable
- Changes? Checks the pipe_changes_needed variable for a non-zero value. If there are changes, send an alert to a webhook post, that indicates a change is needed and fails the job to raise the alarm that the process did not complete all the way to the target. If there are no changes, then the pipeline proceeds to run the 046_s3_stage_to_target.



4c

# Run 045_compare_source_target_metadata

- Compares the stage and target metadata tables to determine changes
- New and changed columns are written to the new_columns_metadata table
- Deleted columns are written to the delete_columns_metadata table



# Conclusion

Schema drift poses a significant challenge in data management, especially with CSV files in Amazon S3. This chapter demonstrated effective strategies for detecting and managing schema drift using dynamic S3 file loading. Key steps include creating a custom file format, populating a staging table for header extraction, and using transformation scripts to translate headers into metadata for accurate data storage.

Discrepancies between current and expected schemas indicate schema drift. Automated checks and transformations compare saved schemas from previous loads to the newly extracted schema. If differences are found, the system flags the drift and can halt the process to alert administrators. This approach ensures that schema changes are addressed promptly, maintaining data integrity and supporting analytical accuracy and operational efficiency.

**In the next chapter, we will explore how to design data pipelines using Matillion Data Productivity Cloud to handle schema drift in database tables. This includes checking for schema changes and determining whether to proceed or flag the change.**



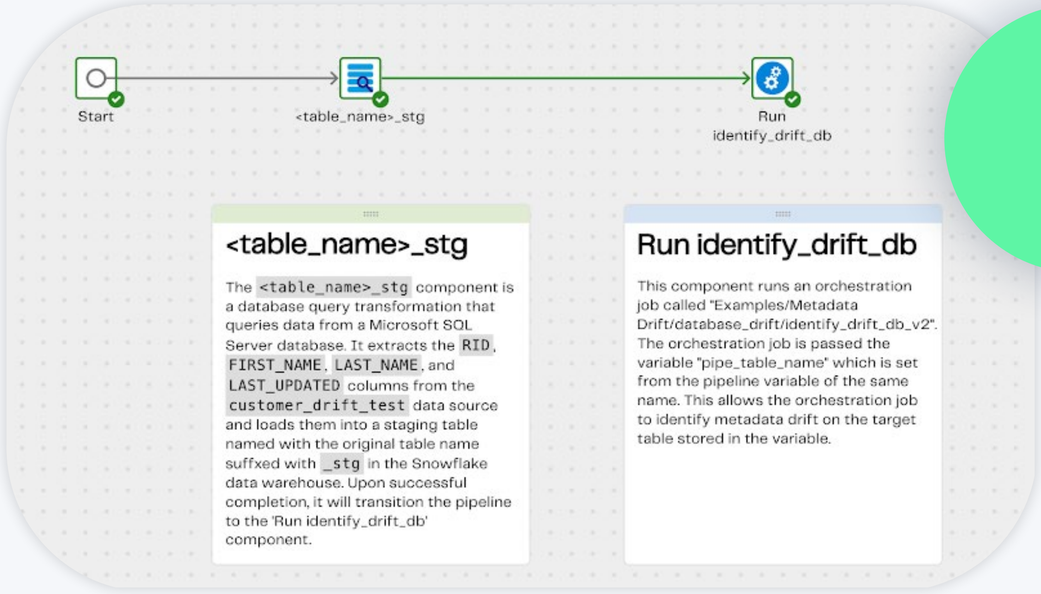
03

# Schema Drift on database tables

**In this chapter, learn how to design data pipelines using Matillion Data Productivity Cloud to manage schema drift in database tables. Discover methods to check for schema changes and determine whether to proceed or flag the change, ensuring seamless data integration and integrity.**

## Initial database load, Identify drift, and Load bronze layer

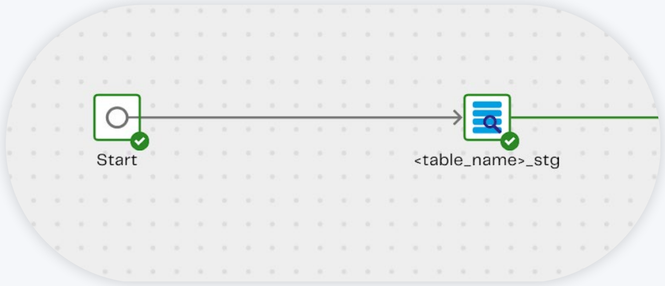
The Matillion Data Productivity Cloud is designed with a Medallion architecture in mind. Source data is extracted from the source database and loaded directly into the Cloud Data Warehouse, maintaining the source schema, otherwise known as the Raw or Bronze layer. The Bronze layer is designed to match the source schema, and this article will demonstrate how to maintain it.



01

## Initial stage data from a database table

Start with a database query component configured to ingest a table from a source database, this one is configured for SQL Server and writes to Snowflake, using a variable to identify the source table and naming the final table as `${var_table_name}_stg` (i.e. ‘customer_stg’).



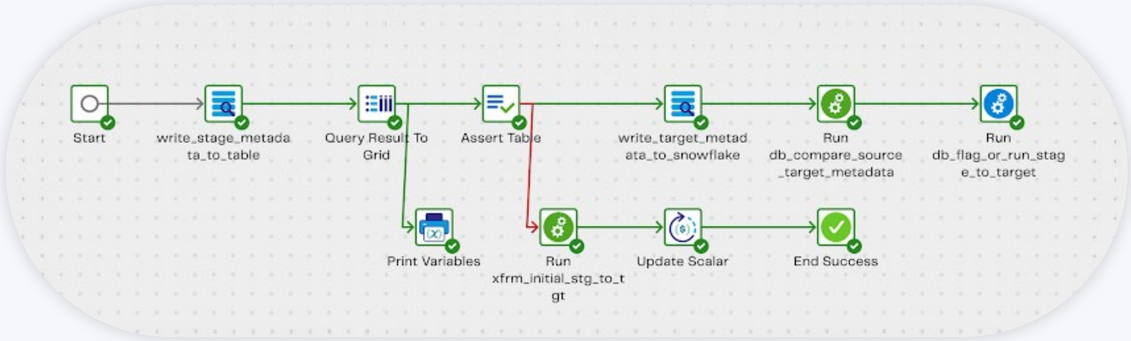
02

# Identify schema / metadata drift

**Now the data has been loaded, it's time to check for schema drift.**

- This step captures the schema metadata into a Snowflake table, saves that metadata to a grid variable, and checks to see if a target table exists.
- If the target table does not exist, it runs the initial stage to target transformation, which creates the table the first time through. After the initial stage to target pipeline runs, it updates a project-level variable, indicating that it's no longer the initial run.
- If the target table exists, the target schema metadata is captured into another table.
- The last step, called the transformation pipeline, compares the two schema tables.

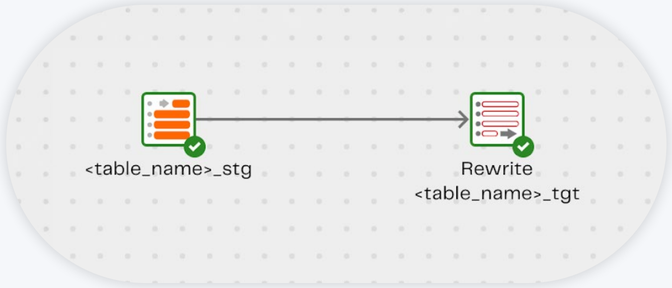
  Note: Make sure to include the 'End Success' component so that the final step in the main pipeline doesn't run to compare source to target.



2a

## xfrm_initial_stg_to_tgt

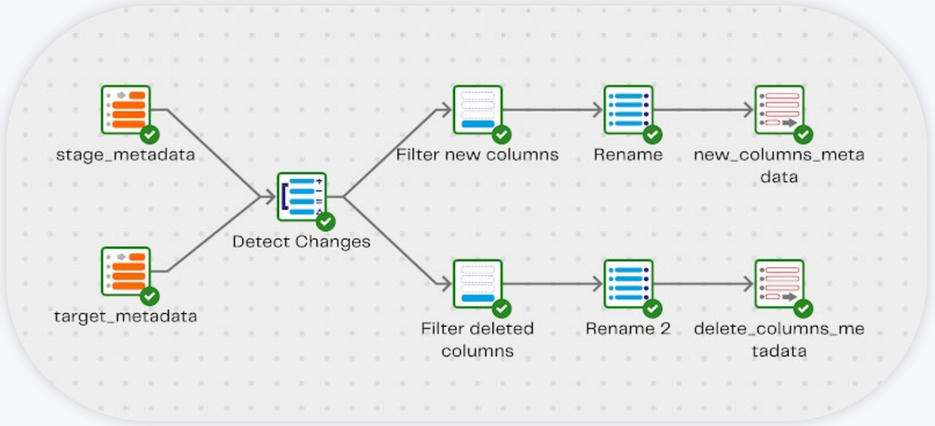
The initial stage-to-target transformation writes the staged data into the target table. The first time, it uses a rewrite table component, which creates the target table dynamically. See the notes section just before the conclusion for how to add additional metadata columns.




2b

## Compare stage and target metadata

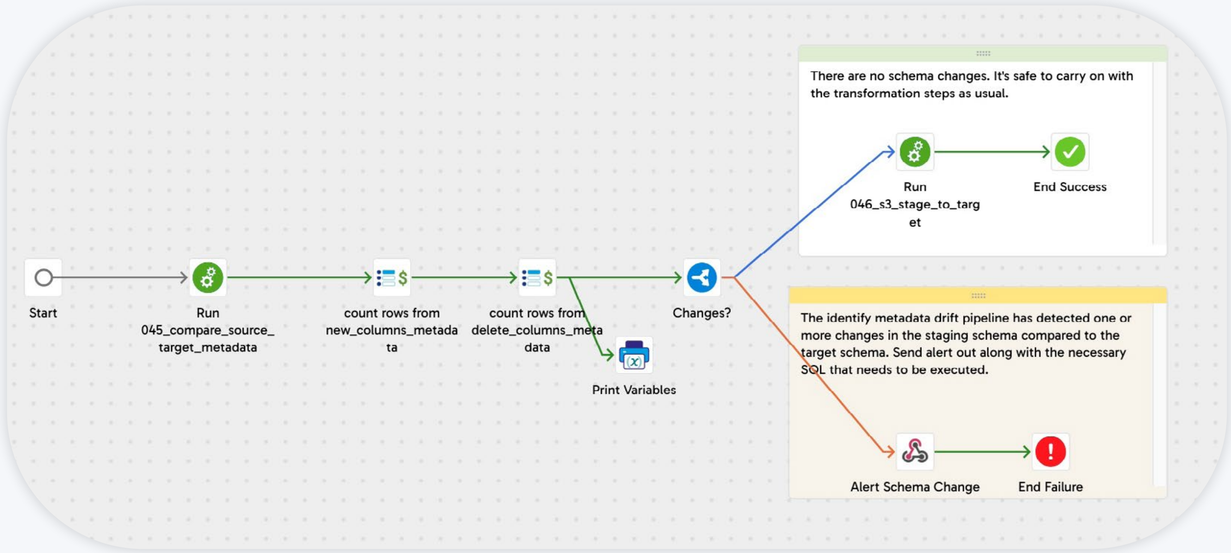
The stage and target metadata tables created in the last step are used along with the ‘detect changes’ component which identifies any new, changed or deleted rows and writes the columns out to tables.



2c

## Flag drift or run stage to target transformation

- This step first checks the new or deleted columns tables created in the previous step to determine if there are any rows in either table.
- If there are changes, then a webhook alert is sent, and the job ends in failure.
- If there aren't any changes, it proceeds to run the xfrm_stg_to_tgt pipeline.




## Notes

If you want additional columns added such as load timestamps and source metadata, add these in the initial database query component so that the two tables are the same and can be compared in the later step.

Set the mode to 'Advanced' and enter a SQL Query, adding the metadata fields to the initial stage query. This example shows all columns from the source, the current timestamp used to populate a field for stage_load_dttm, and some constants to load source_schema and source_table. These can also be variables if you are iterating through different schemas and tables.

### SQL Query

```
select *,
current_timestamp as stage_load_dttm,
'SQLServer' as source_database,
'store' as source_schema,
'customer' as source_table
from angie.customer_drift_test
```

# Conclusion

**Successfully managing schema drift is vital for maintaining data integrity and efficiency in the world of data. This eBook explored the complexities of schema drift across various data sources, providing comprehensive solutions to ensure robust data management using Matillion Data Productivity Cloud.**

01

Chapter One focused on schema drift in Excel files. We discussed how schema changes, such as added columns, renamed fields, or data type modifications, can disrupt data pipelines. By leveraging dynamic queries and metadata handling, we outlined a method to create resilient data pipelines that adapt to these changes seamlessly.

02

Chapter Two shifted the focus to S3 files, demonstrating the challenges of handling diverse and dynamic file formats stored in cloud storage. We explored a design pattern for dynamically processing large numbers of S3 files, ensuring that schema drift is detected and managed efficiently. The process involved custom file formats, parsing headers, and creating stage tables to match the evolving file structures.

03

Chapter Three addressed schema drift in database tables. We illustrated how to design data pipelines that check for schema changes in source database tables and determine whether to proceed or flag the changes. By automating the stage-to-target load for the bronze layer and comparing metadata, we ensured that any schema discrepancies are promptly detected and managed, maintaining data integrity throughout the process.

Across all chapters, a common thread has been the emphasis on dynamic handling and automated checks to manage schema drift. By comparing current and target schemas, our approach flags discrepancies and can halt processes for manual intervention if necessary. This proactive method ensures that schema changes do not go unnoticed, preserving data integrity and supporting analytical accuracy and operational efficiency

As we conclude this eBook , the strategies discussed provide a robust framework for managing schema drift across various data sources. By implementing these techniques, data teams can maintain the integrity and reliability of their data pipelines, ensuring accurate and efficient data processing in an ever-changing data landscape. Thank you for reading, and we encourage you to explore the Matillion Data Productivity Cloud further to enhance your data management practices.

Book a demo