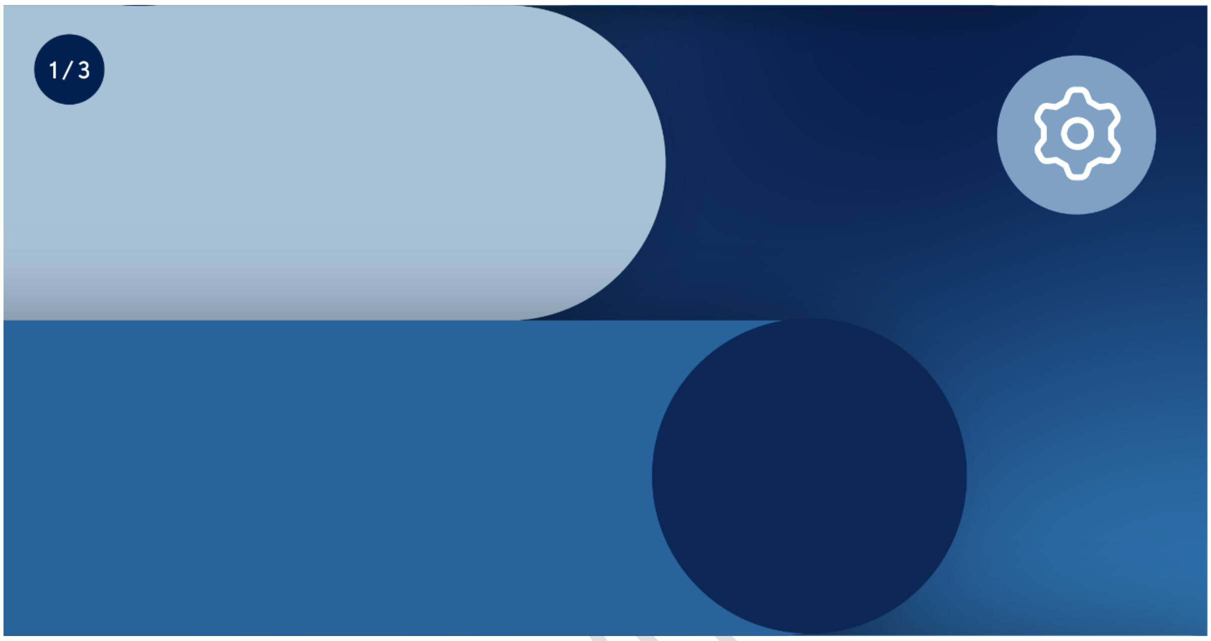
How to Use Matillion's Database Query Component to Ingest Microsoft SQL Server Data in Snowflake

Reference : https://www.matillion.com/blog/how-to-use-matillions-database-query-component-to-ingest-microsoft-sql-server-in-snowflake



Problem statement:

While many organizations have started or made their journey into the cloud, there still remains a need to operate and run traditional RDBMS environments, like MS SQL Server, on-prem. Legacy applications may interact and operate from these instances, yet their data continues to be siloed from the broader reporting elements that are now running off of data clouds such as Snowflake.

This siloed data can lead to inaccurate, slow, or inconsistent reporting which may have an impact on the broader business. Data teams are tasked with bringing data over to Snowflake. However, many of the traditional tools available to them do not offer the speed, ease of use, and reliability that the Matillion Data Productivity Cloud (DPC) offers.

This article provides a quick walk-through of what DPC offers for users who use MS SQL server data and Snowflake! This is a three-part series where we will cover the following:

Part 1 - Ingesting SQL server data using Database Query component (basic and advanced)

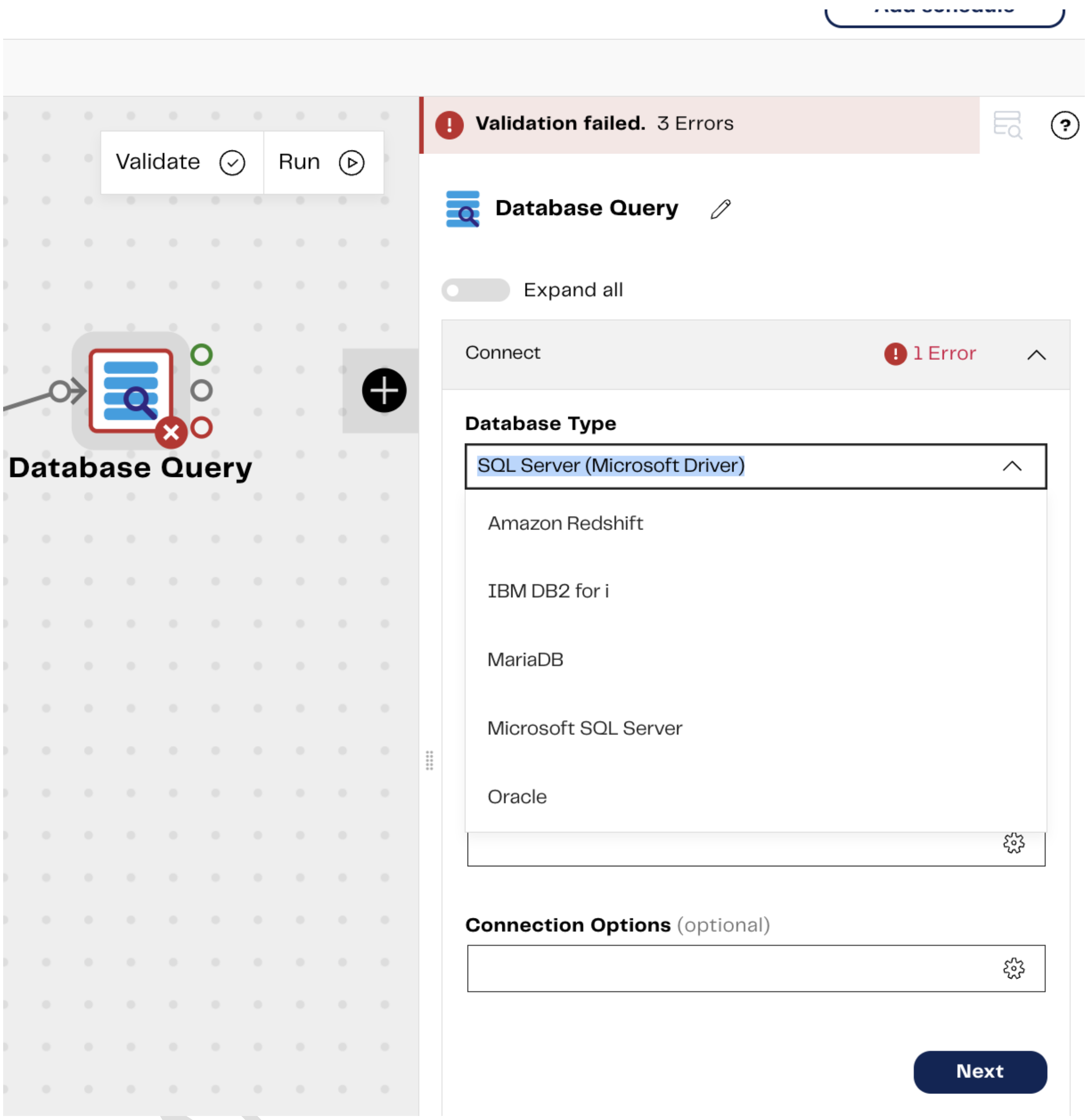
Part 2 - Reverse ETL to write data from Snowflake back to MS SQL (coming soon)

Part 3 - Matillion's CDC offering for MSSQL (coming soon)

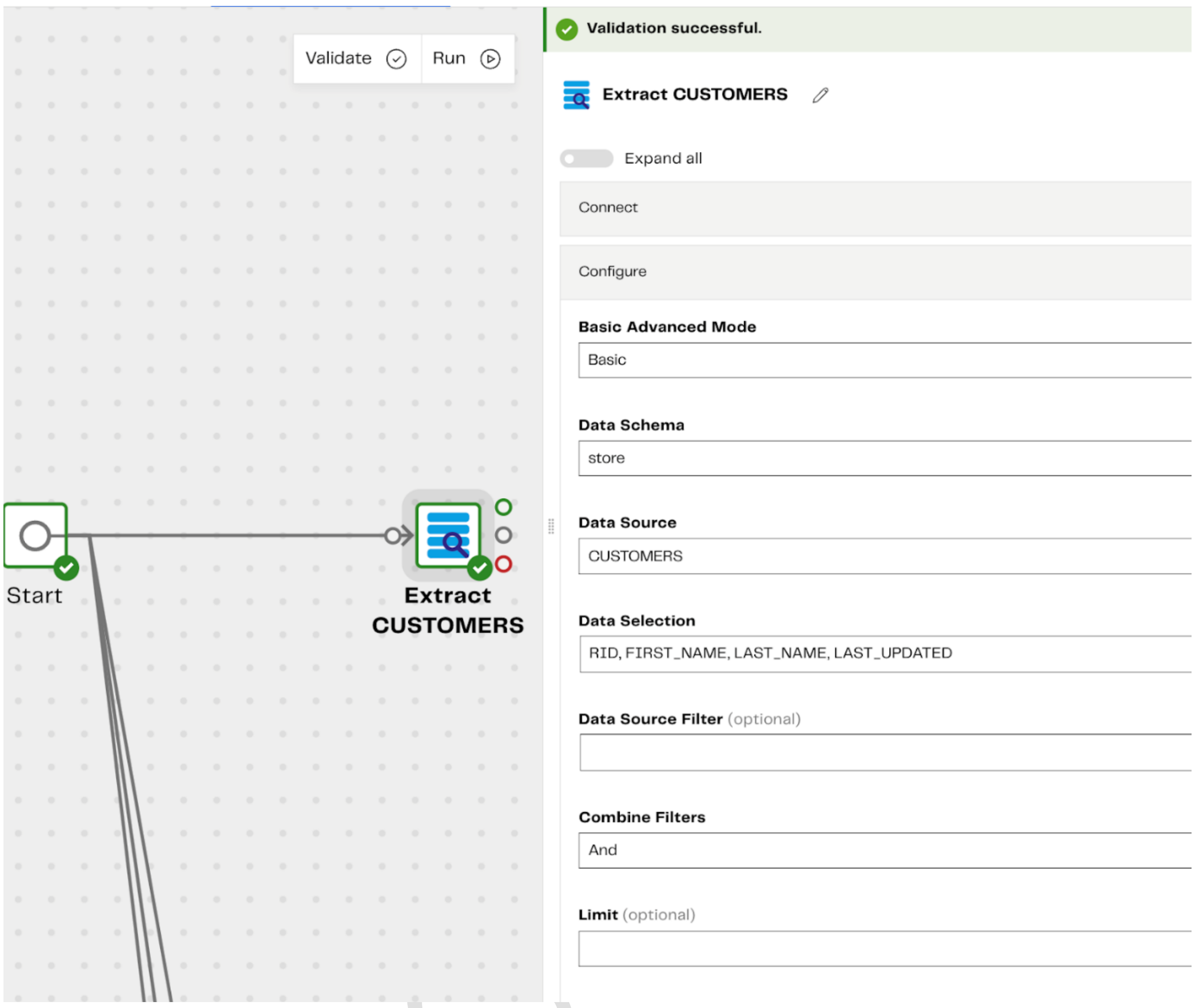
Configuring the Database query component for MSSQL

Pulling a single table from MSSQL to Snowflake

In an orchestration pipeline, we will use the **database query** component to connect to our SQL server database and pull data into Snowflake. This component offers multiple drivers for SQL servers as well as drivers for other RDBMS like Oracle, PostgreSQL, and MySQL.


Validate
Run
Validation failed. 3 Errors
Database Query
Expand all
Connect
1 Error
Database Type
SQL Server (Microsoft Driver)
Amazon Redshift
IBM DB2 for i
MariaDB
Microsoft SQL Server
Oracle
Connection Options (optional)
Next

In this configuration, we will use our **basic** mode to have Matillion guide you in a drop-down fashion to source the data you want from MSSQL over to Snowflake. The prompts populate lists of values (from MSSQL [source] and Snowflake [target]) to reduce user error and ensure successful pipeline runs. This also allows self-service by users who may not possess deep technical knowledge to build effective pipelines with minimal to no support. For this example, I will extract the **CUSTOMERS** table from MSSQL and load it into Snowflake.



Validation successful.

Extract CUSTOMERS

Expand all

Connect

Configure

Basic Advanced Mode: Basic

Data Schema: store

Data Source: CUSTOMERS

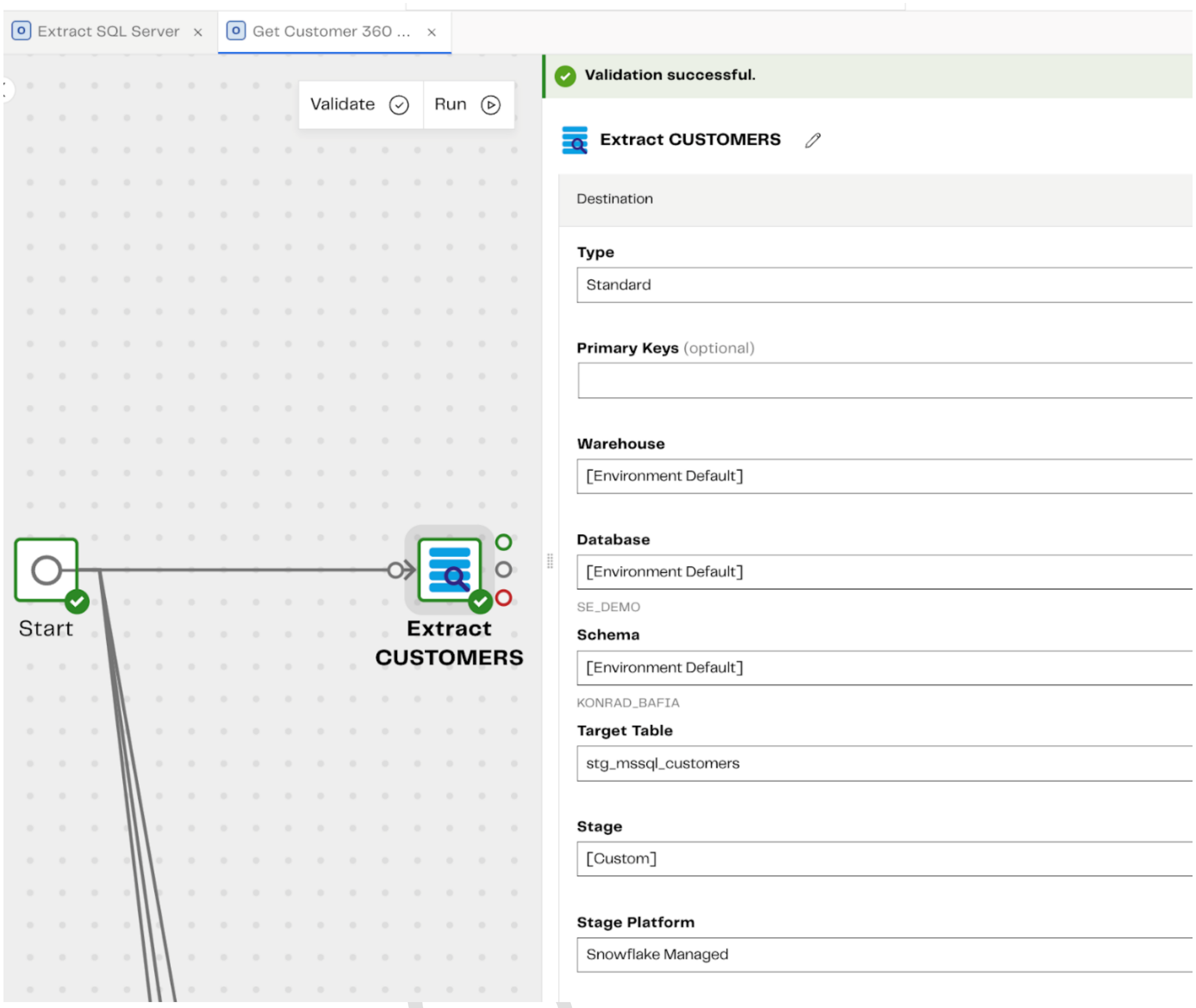
Data Selection: RID, FIRST_NAME, LAST_NAME, LAST_UPDATED

Data Source Filter (optional):

Combine Filters: And

Limit (optional):

One of the best features about Matillion is that you do not have to create the target table in advance of loading data. Instead, we simply tell Matillion where we want to put this sourced data in Snowflake. This is done in the **destination** tab of the database query component properties. Matillion will infer the appropriate schema on the Snowflake side from the metadata coming from MSSQL, and simply create the table on the fly in Snowflake then load it all in one motion. Talk about productivity!



Validation successful.

Extract CUSTOMERS

Destination

Type: Standard

Primary Keys (optional):

Warehouse: [Environment Default]

Database: [Environment Default]
SE_DEMO

Schema: [Environment Default]
KONRAD_BAFIA

Target Table: stg_mssql_customers

Stage: [Custom]

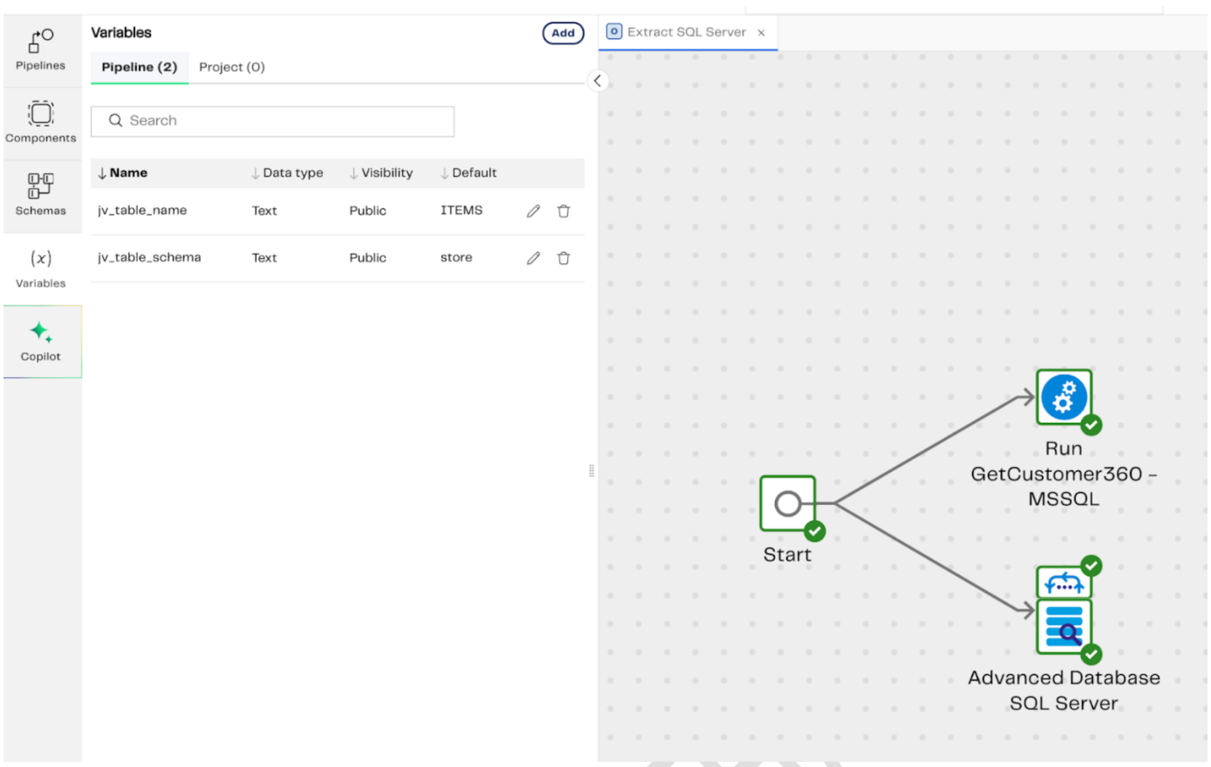
Stage Platform: Snowflake Managed

The component is now ready to run. When complete, a new table will be created by Matillion named *stg_mssql_customers* in Snowflake.

Pulling multiple tables using dynamic SQL and iterators

While there may be use cases where you want to sync one-off tables from MSSQL to Snowflake as shown above, it is more likely that you will want to sync dozens if not hundreds of tables over. Doing that in a one-by-one basis would not be an effective development approach, so using a combination of variables, iterators, and dynamic SQL is a better approach and more modular in design.

Matillion supports many variable types, one of which is a pipeline variable which is essentially a scalar value that is local to a pipeline. In this example, I will create 2 variables (jv_table_name and jv_table_schema) to store table names and schema values respectively.



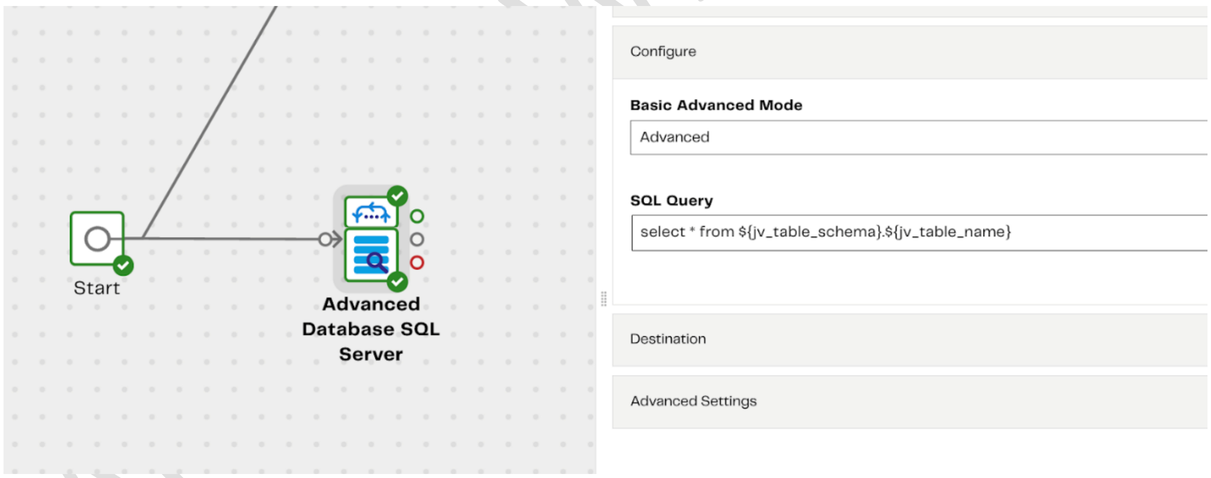
Variables

Pipeline (2) | Project (0)

Search

Name	Data type	Visibility	Default
jv_table_name	Text	Public	ITEMS
jv_table_schema	Text	Public	store

I will also re-use the **database query** component in my pipeline but instead of using the **basic** mode, I will use the **advanced** mode which will allow me to write a SQL query against the source MSSQL database. This is also where I will reference my variables to make my SQL statement dynamic.

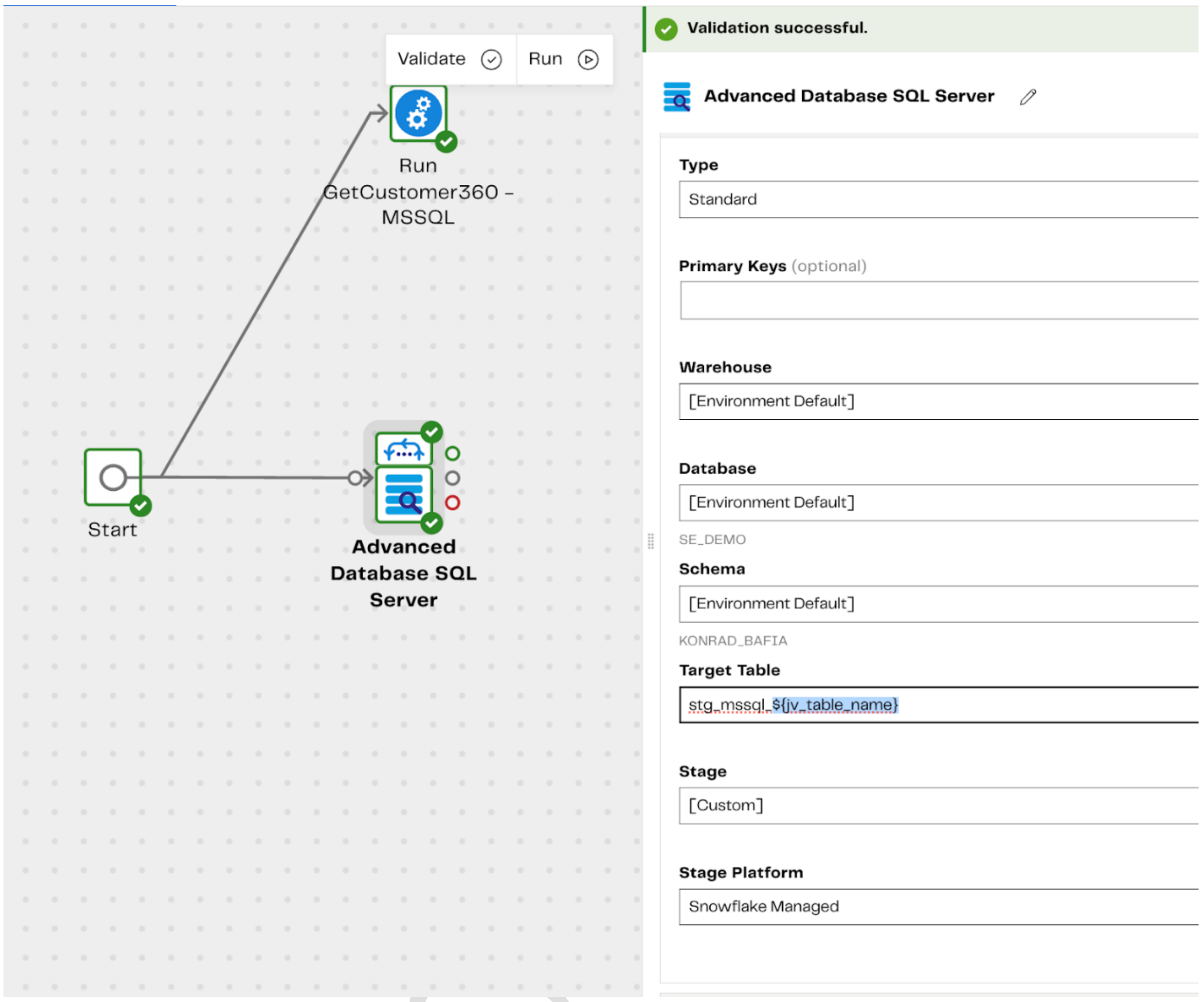


Basic Advanced Mode: Advanced

SQL Query:

```
select * from ${jv_table_schema}.${jv_table_name}
```

Note the ${*variable_name*} syntax to reference the variables I created earlier. Also, notice how our target table name is referencing the *${jv_table_name}* variable in our destination tab. This is important because each table has different metadata, so Matillion will create new tables and populate the variable value at run time.



Validation successful.

Advanced Database SQL Server

Type
Standard

Primary Keys (optional)

Warehouse
[Environment Default]

Database
[Environment Default]
SE_DEMO

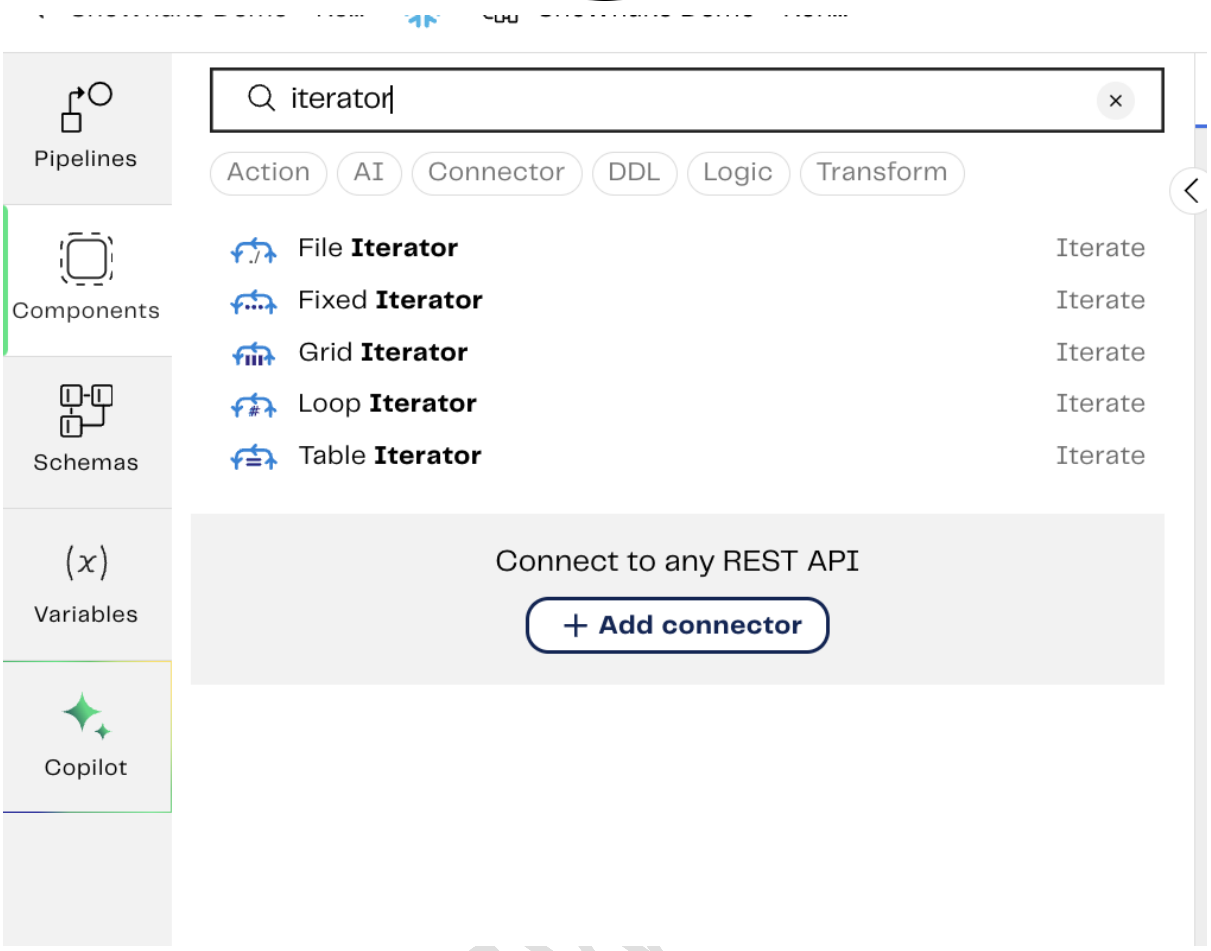
Schema
[Environment Default]
KONRAD_BAFIA

Target Table
stg_mssql_${jv_table_name}

Stage
[Custom]

Stage Platform
Snowflake Managed

Finally, in order for this to work, we need to attach an iterator on top of our **database query** component to set the loop that Matillion should go through. Matillion offers many different types of iterators for your loop-based pipelines. For this example, we will use the **fixed iterator** which is a fixed list of values to loop over for simplicity in this demo.

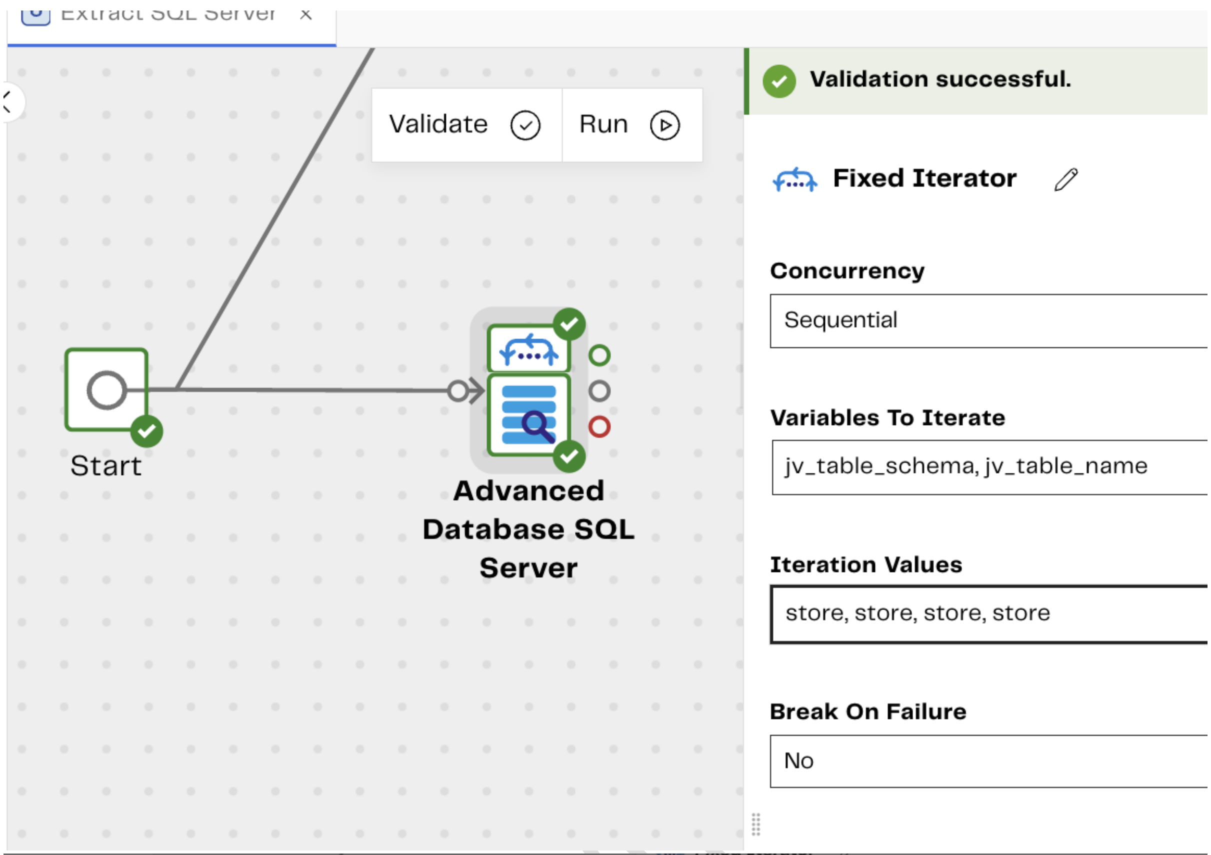


File **Iterator**	Iterate
Fixed **Iterator**	Iterate
Grid **Iterator**	Iterate
Loop **Iterator**	Iterate
Table **Iterator**	Iterate

Connect to any REST API

+ Add connector

By attaching the fixed iterator to our database query component, we can configure it to assign values to our variables at run time. We have set four combinations of schema + table name that will ultimately create 4 new tables in Snowflake at run time with the appropriate data coming from MSSQL.



Validation successful.

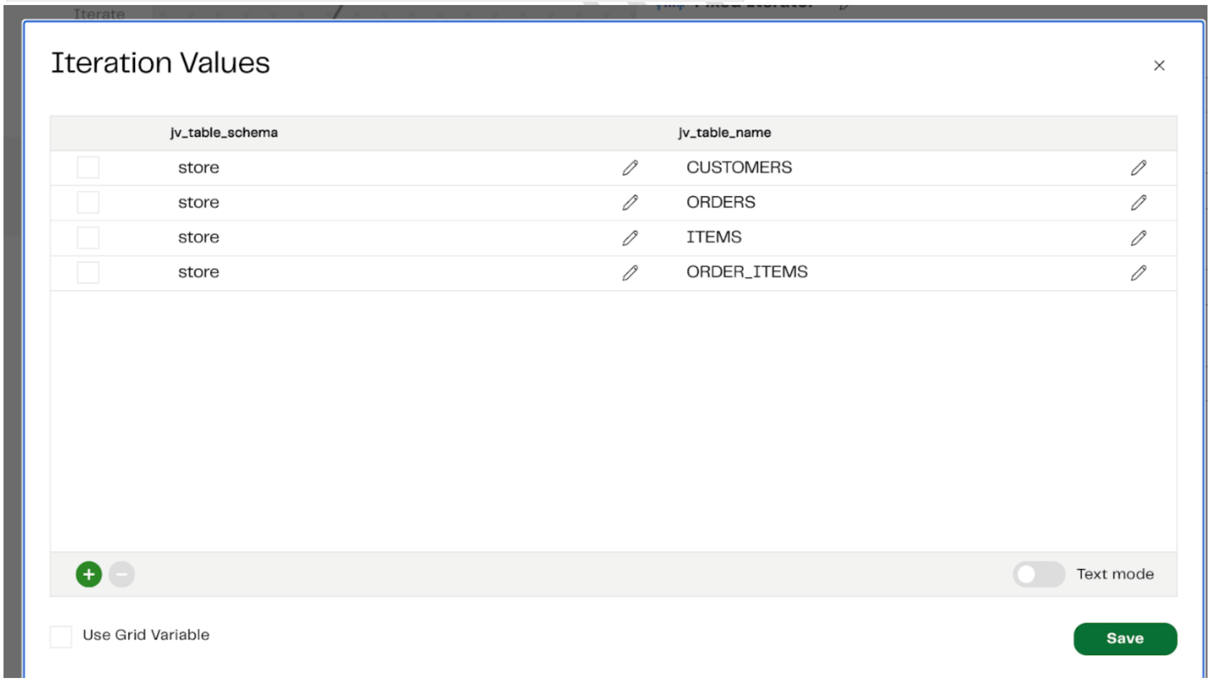
Fixed Iterator

Concurrency
Sequential

Variables To Iterate
jv_table_schema, jv_table_name

Iteration Values
store, store, store, store

Break On Failure
No



Iteration Values

jv_table_schema	jv_table_name
store	CUSTOMERS
store	ORDERS
store	ITEMS
store	ORDER_ITEMS

Imagine doing this on a larger scale. For example, you can configure Matillion to pull data from MSSQL information_schema to pull a list of tables from the schemas you want to copy over. Then you can systematically pass that information_schema data into variable(s) in Matillion to loop over a job such as this and clone an entire database from MSSQL to Snowflake in a few hours. This is a common use case that we have solved hundreds of times across many customers.

Reverse ETL data from Snowflake back to SQL Server using Matillion's Output Components

This is part two of a three-part series about Matillion's capabilities with Snowflake and Microsoft SQL Server. Please refer to Part 1 of this series, where we covered ingesting data from MSSQL into Snowflake. Data must first exist in Snowflake before you can push it to MSSQL—a process known as Reverse ETL.

What is Reverse ETL?

We all know that making changes in an organization is difficult. It is even more difficult when changes are required to technology that is dependent on/for other technology or business applications. This is typically the case with transactional RDBMS instances like Microsoft SQL Server.

While many have made the leap to the cloud and adopted modern data platforms such as Snowflake, the scope of that change often doesn't include moving off of RDBMS like MSSQL because often these are the data sources for legacy applications that are not suited or compatible with the cloud. These applications typically have been around for years if not decades, and ripping out their core database to move to the cloud is a huge if not impossible, task for many data teams. These on-prem RDBMS aren't elastic like their cloud counterparts, yet customers and internal stakeholders alike are demanding data requirements and enhancements. So, how can we help?

Well, one way we can help is by relieving the MSSQL instance from any analytical or computational workloads and having it solely focus on transactional events from the underlying data application or consumer. We can follow the steps we did in part 1 of our series and replicate the required transactional data into Snowflake from MSSQL, then use the power and elasticity of the data cloud to transform/aggregate/calculate the data at scale, and then simply send inserts/updates back to MSSQL using Matillion's output components.

This activity is commonly referred to as "reverse ETL" in the data engineering space, and it is a trivial task to achieve in Matillion's Data Productivity Cloud!

What is in Scope

Before we jump into our walk-through, it is important to set the scope of what Matillion offers in regard to MSSQL.

In scope:

- Pulling data from MSSQL into cloud data platforms like Snowflake
- Pushing data to MSSQL from cloud data platforms like Snowflake

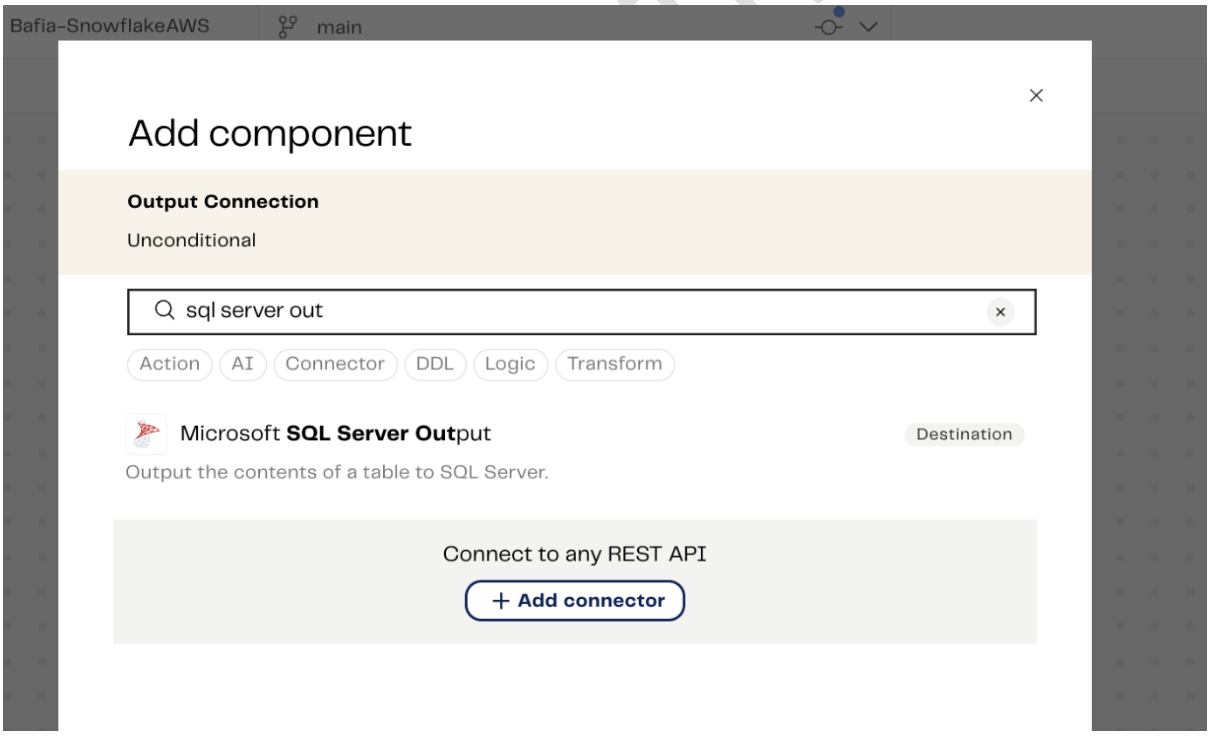
Not in scope:

- Creating transformation jobs in Matillion to be executed in MSSQL
- Transforming data in-flight between MSSQL and cloud data platforms like Snowflake
- Executing custom scripts on MSSQL
- Pushing 'ad-hoc' data on the fly without it first existing in a Snowflake table

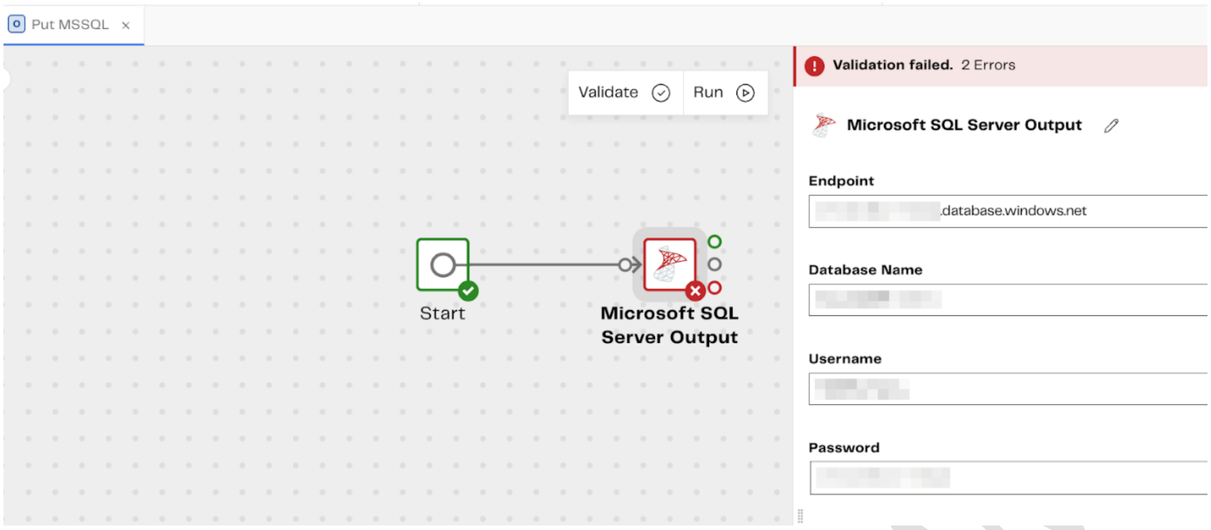
Configuring the Microsoft SQL Server Output component

Pushing a single table from Snowflake to MSSQL

In an orchestration pipeline, we will start with the **Microsoft SQL Server Output** component to send data from a Snowflake table to our SQL server database.



For this component, the first set of properties pertains to your MSSQL instance that you want to connect to. You can read more about these fields in our documentation here: https://docs.matillion.com/data-productivity-cloud/designer/docs/sql-server-output/



Validation failed. 2 Errors

Microsoft SQL Server Output

Endpoint
[redacted].database.windows.net

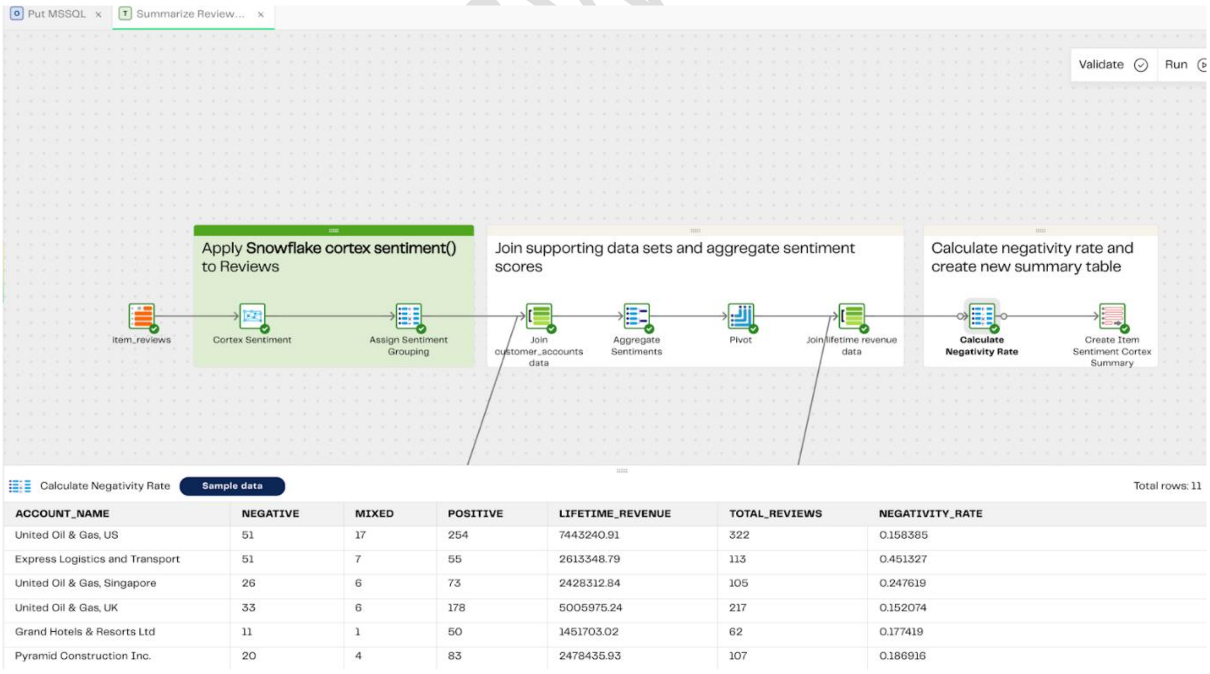
Database Name
[redacted]

Username
[redacted]

Password
[redacted]

Once you have entered the required MSSQL information, the next set of properties pertains to what data you want to send from Snowflake to MSSQL.

*In this scenario, our legacy application captures customer reviews in MSSQL from our legacy application. However, doing sentiment analysis on these reviews is difficult and compute-intensive against our on-prem MSSQL instance. So we have replicated the review data into Snowflake and used CortexML sentiment to calculate a negativity rate for each of our product suppliers into a table called **item_sentiment_cortex_summary**. Now we want to send this analysis data back to MSSQL so our application engineers can join to it within our legacy app to provide a new feature to our application customers.*

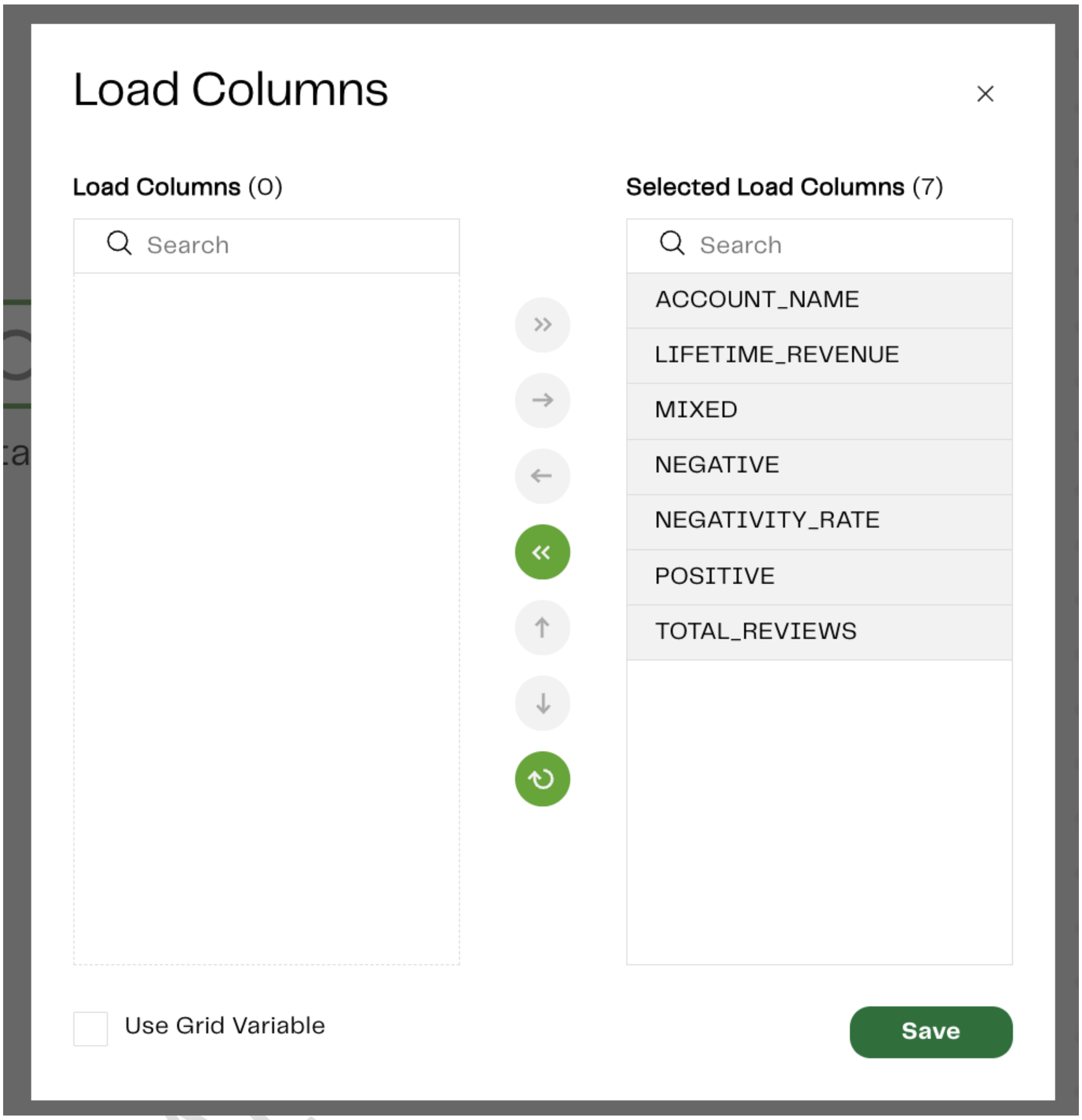

Put MSSQL
Summarize Review...
Validate
Run
Apply Snowflake cortex sentiment() to Reviews
Join supporting data sets and aggregate sentiment scores
Calculate negativity rate and create new summary table
item_reviews
Cortex Sentiment
Assign Sentiment Grouping
Join customer_accounts data
Aggregate Sentiments
Pivot
Join lifetime revenue data
Calculate Negativity Rate
Create Item Sentiment Cortex Summary
Sample data

Calculate Negativity Rate

Total rows: 11

ACCOUNT_NAME	NEGATIVE	MIXED	POSITIVE	LIFETIME_REVENUE	TOTAL_REVIEWS	NEGATIVITY_RATE
United Oil & Gas, US	51	17	254	7443240.91	322	0.158385
Express Logistics and Transport	51	7	55	2613348.79	113	0.451327
United Oil & Gas, Singapore	26	6	73	2428312.84	105	0.247619
United Oil & Gas, UK	33	6	178	5005975.24	217	0.152074
Grand Hotels & Resorts Ltd	11	1	50	1451703.02	62	0.177419
Pyramid Construction Inc.	20	4	83	2478435.93	107	0.186916

Let's select our *item_sentiment_cortex_summary* table in Snowflake in our **Source Table** property. We want the table in MSSQL to have a dpc_ prefix, so we will set the value in our **Target Table** property to *dpc_item_sentiment_cortex_summary*. We will also choose all of our columns in the **Load Columns** property.


Load Columns

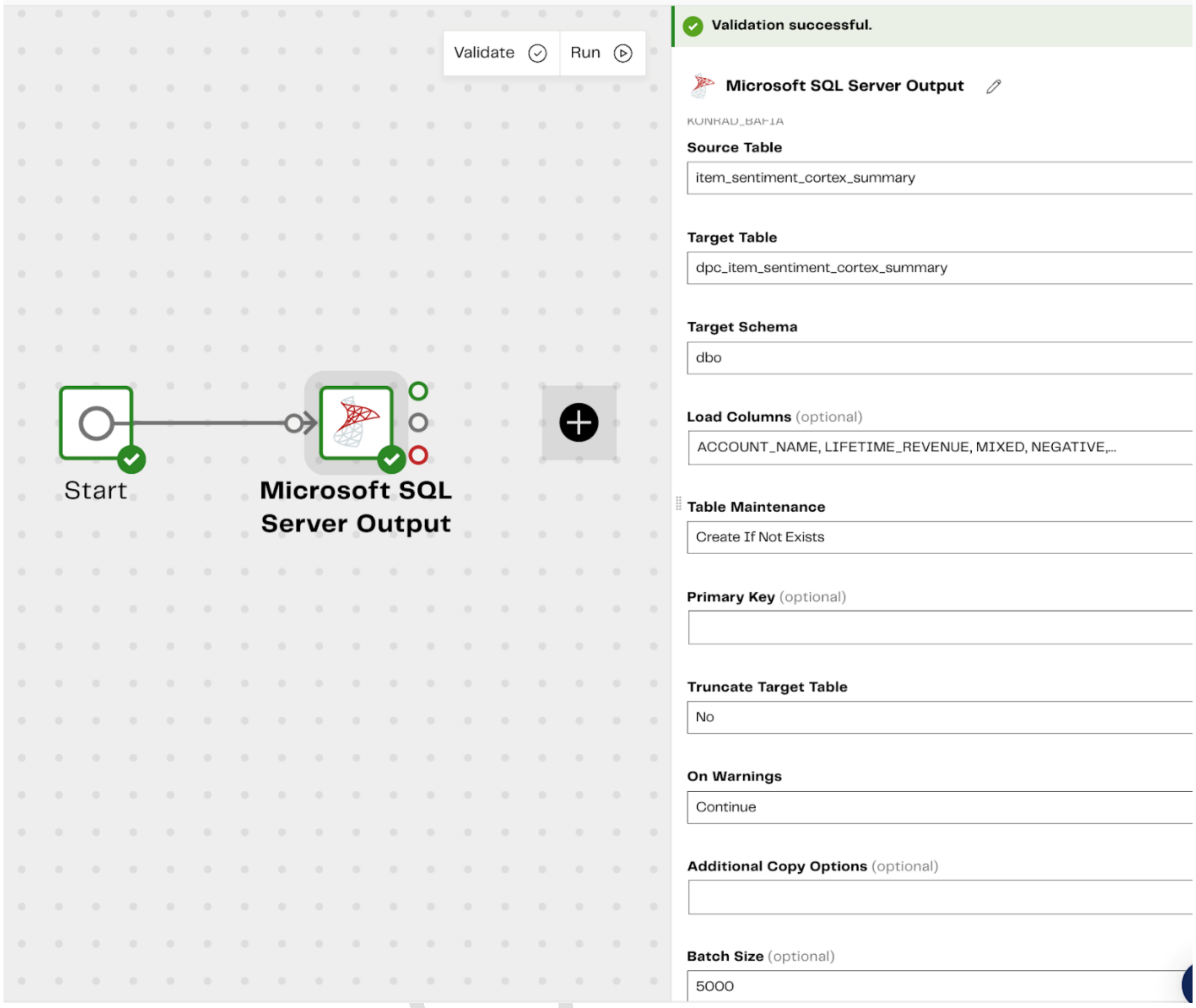
Load Columns (0)

Selected Load Columns (7)

- ACCOUNT_NAME
- LIFETIME_REVENUE
- MIXED
- NEGATIVE
- NEGATIVITY_RATE
- POSITIVE
- TOTAL_REVIEWS

Use Grid Variable

For **Table Maintenance**, we will choose Create If Not Exists. This will create the table on the fly in the schema we specified, which is a nice feature since we don't have to worry about creating the target table in MSSQL. The last property we will set is **Truncate Target Table** to *True* because we want to truncate/insert this table every time it runs. You can also up the batch size if you are moving a lot of data but we will keep it at 5000 for now. Our component should be in a valid state after all of the properties are set.



Validation successful.

Microsoft SQL Server Output

KONRAD_BAFIA

Source Table: item_sentiment_cortex_summary

Target Table: dpc_item_sentiment_cortex_summary

Target Schema: dbo

Load Columns (optional): ACCOUNT_NAME, LIFETIME_REVENUE, MIXED, NEGATIVE,...

Table Maintenance: Create If Not Exists

Primary Key (optional):

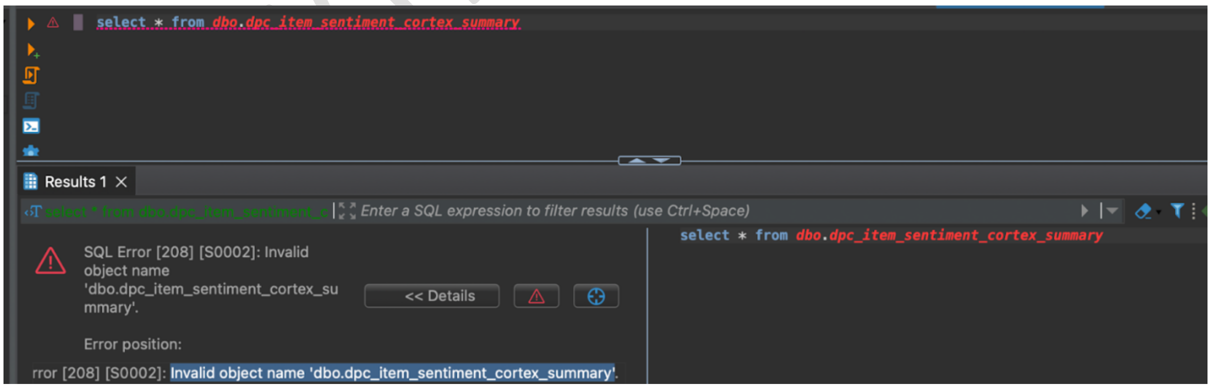
Truncate Target Table: No

On Warnings: Continue

Additional Copy Options (optional):

Batch Size (optional): 5000

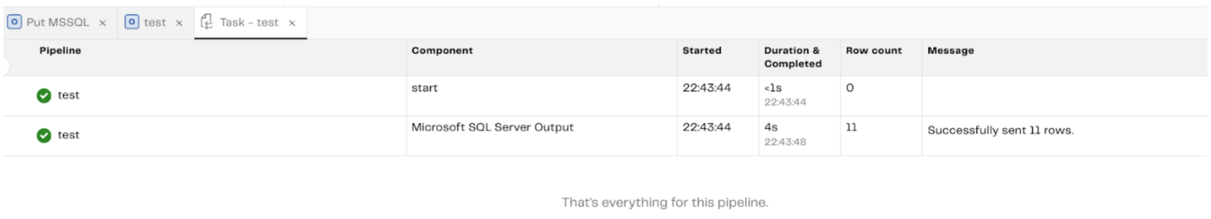
As proof, here is our MSSQL database showing that this table does not exist before our pipeline has run in Matillion:



```
select * from dbo.dpc_item_sentiment_cortex_summary
```

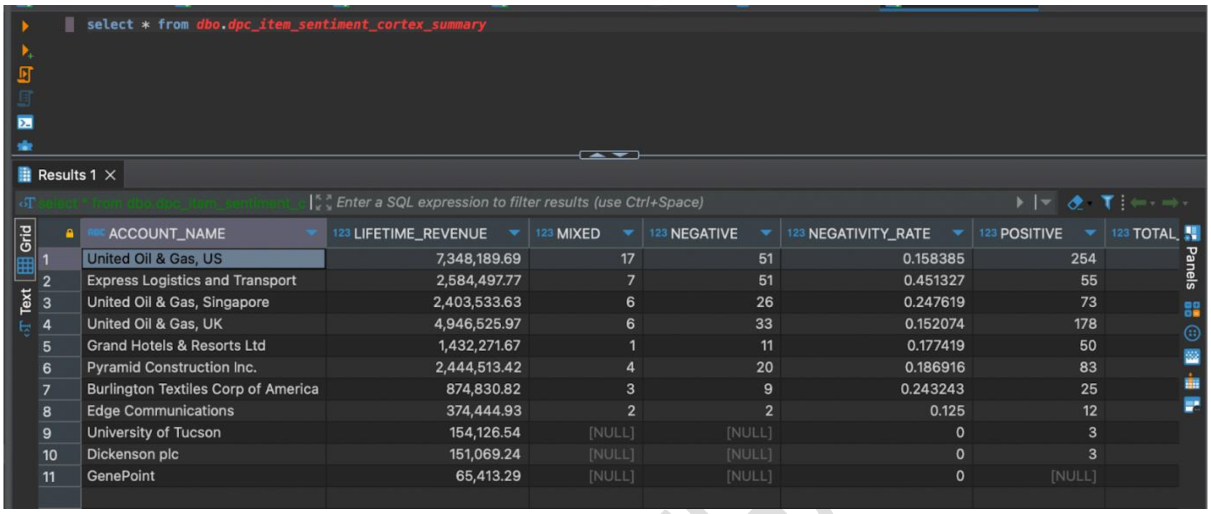
```
select * from dbo.dpc_item_sentiment_cortex_summary

SQL Error [208] [S0002]: Invalid object name 'dbo.dpc_item_sentiment_cortex_summary'.

Error position:
rror [208] [S0002]: Invalid object name 'dbo.dpc_item_sentiment_cortex_summary'.
```

Now let’s run our newly created MSSQL Output job. You can see that the pipeline ran successfully and we should have a newly created table in our dbo schema with 11 records loaded from Snowflake.



Pipeline	Component	Started	Duration & Completed	Row count	Message
test	start	22:43:44	<1s 22:43:44	0	
test	Microsoft SQL Server Output	22:43:44	4s 22:43:48	11	Successfully sent 11 rows.

That's everything for this pipeline.


```
select * from dbo.dpc_item_sentiment_cortex_summary
```

	ACCOUNT_NAME	LIFETIME_REVENUE	MIXED	NEGATIVE	NEGATIVITY_RATE	POSITIVE	TOTAL_
1	United Oil & Gas, US	7,348,189.69	17	51	0.158385	254	
2	Express Logistics and Transport	2,584,497.77	7	51	0.451327	55	
3	United Oil & Gas, Singapore	2,403,533.63	6	26	0.247619	73	
4	United Oil & Gas, UK	4,946,525.97	6	33	0.152074	178	
5	Grand Hotels & Resorts Ltd	1,432,271.67	1	11	0.177419	50	
6	Pyramid Construction Inc.	2,444,513.42	4	20	0.186916	83	
7	Burlington Textiles Corp of America	874,830.82	3	9	0.243243	25	
8	Edge Communications	374,444.93	2	2	0.125	12	
9	University of Tucson	154,126.54	[NULL]	[NULL]	0	3	
10	Dickenson plc	151,069.24	[NULL]	[NULL]	0	3	
11	GenePoint	65,413.29	[NULL]	[NULL]	0	[NULL]	

And that's it! With a few properties set on our Output components, you can send critical data back to Microsoft SQL Server instances that enhance your legacy applications while still leveraging the power of Snowflake and Matillion to do the heavy lifting and allow a more stable transactional database for your apps.