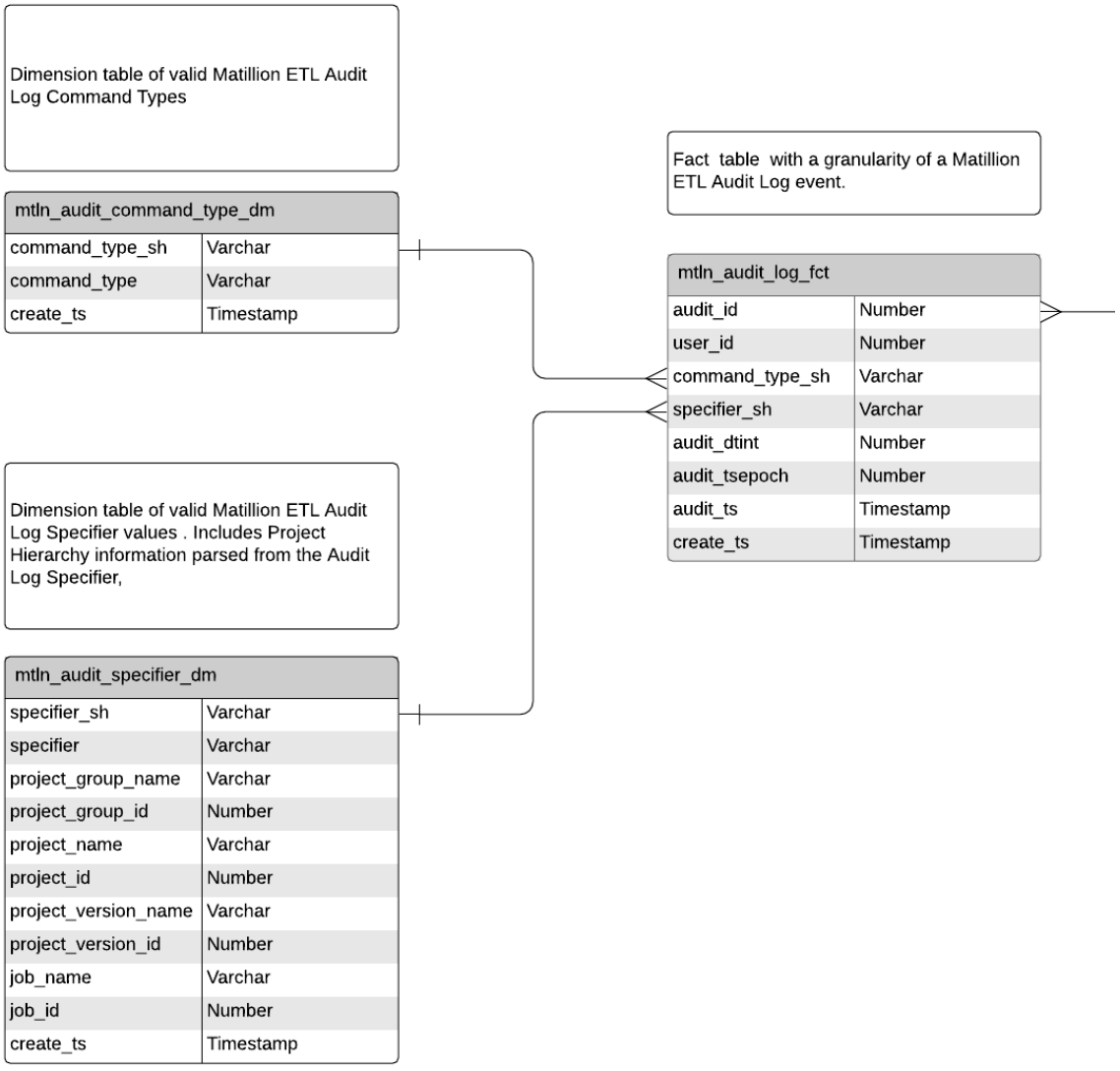
Matillion Security Controls: Enable User Auditing (Part 3 of 3)

Part 3: Processing Audit Log Data – Facts and Type 0 Dimensions

In the first two parts of this series, we demonstrated **how to use the Matillion v1 API layer as a source of data**, **designed a dimension model to store User and Audit Log data** and showed **how to load a Type 2 Slowly Changing Dimension of User data** . In Part 3 of this series, we show how to populate the Facts and Dimensions with Audit Log data from Matillion.

Audit Log data is accessible through the Matillion UI, giving Admin users the ability to review all user activity within a Matillion ETL instance. To access this same data more programmatically, there is a **Matillion API v1** endpoint that can be used. The API Profile used to get Audit Log data from Matillion ETL was previously highlighted **in Part 1 of this series**. Once the raw Audit Log data has been staged, it can be used to manage the related Dimension and Fact tables.

Table Metadata

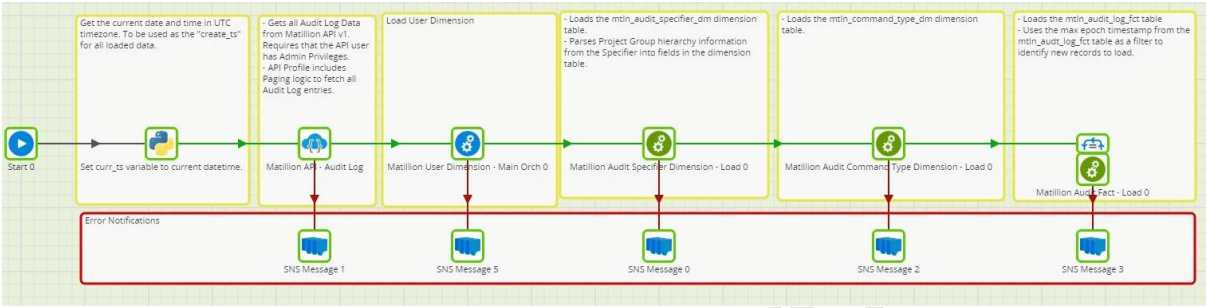


In this example, we are using Matillion ETL's Audit Log data to populate a Fact table and two related Dimensions. Every Audit Log record has the following data attributes associated with the event that was captured:

- A unique numeric ID for each Audit Log event
- The User associated to the event
- A Command Type, which categorizes the event
- A Specifier, which provides the details of the event itself
- A Timestamp (in **Unix Time** format) of when the event occurred

From this data, we use the Specifier and Command Type attributes to populate Dimension tables. Both Dimension tables are treated as **SCD Type 0**, with only inserts of new Dimension rows ever expected. The related Fact table has a granularity of a single Audit Log event.

Job Design

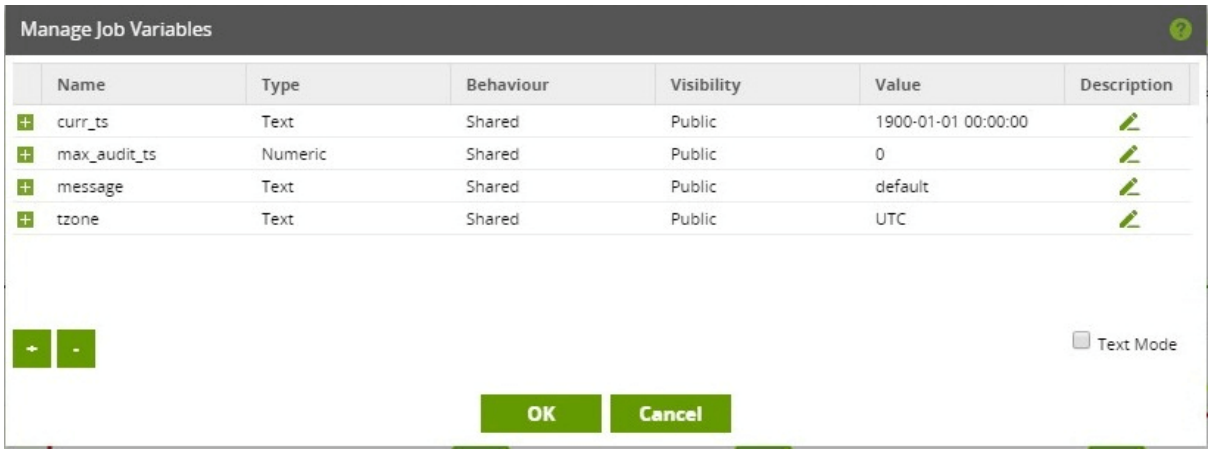


As is the case with all Matillion ETL jobs, there is a Main Orchestration job that serves as the primary orchestrator of processes that execute when the job runs.

The general order of events in the Main Orchestration are as follows:

1. Set job variable values
2. Stage raw data from the source (See **the prior article in this series** on using the Matillion API v1)
3. Identify and Apply data changes on Dimension tables
4. Identify and Apply data changes on Fact tables

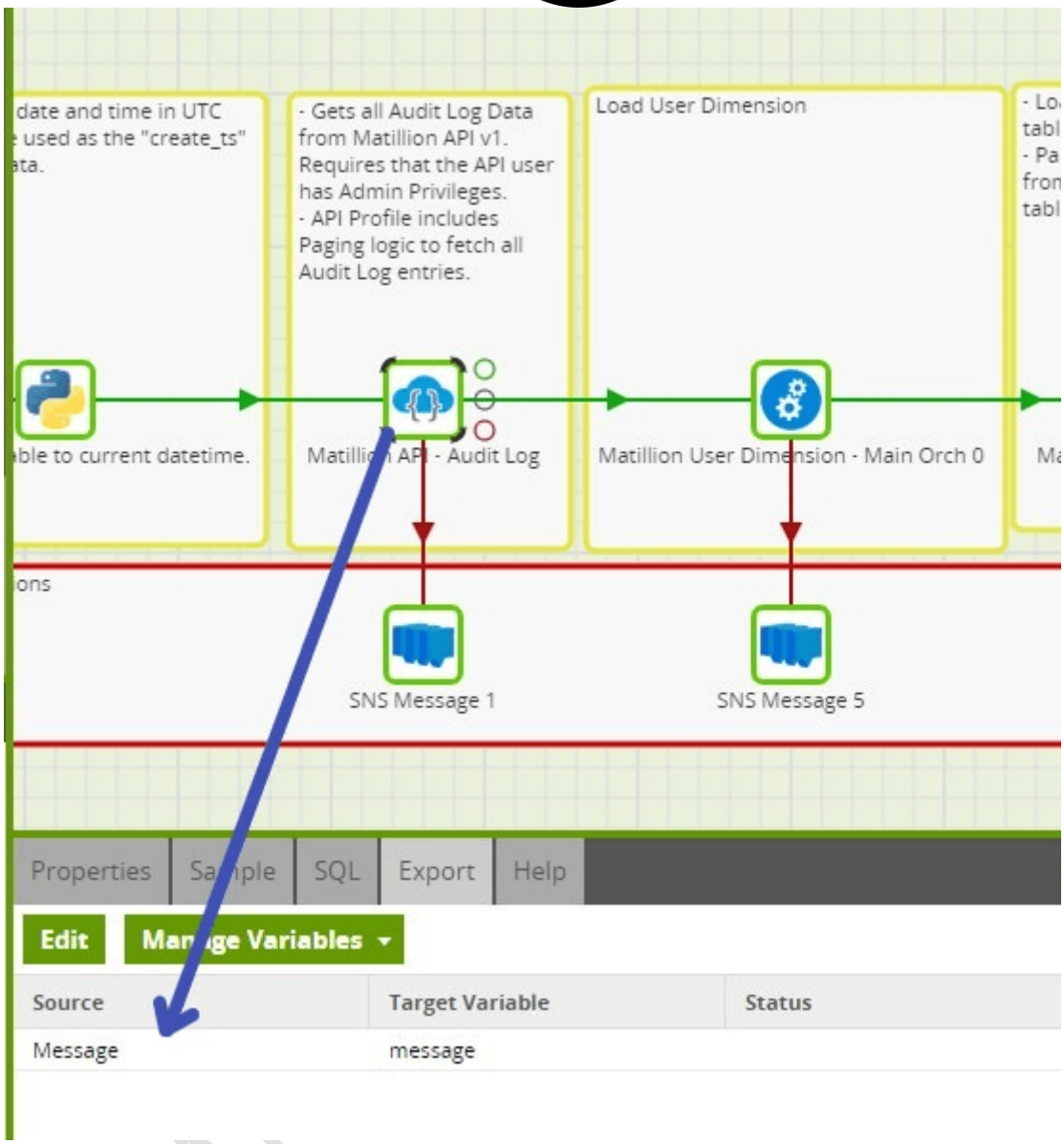
Job Variables



The Main Orchestration job has the following **Job Variables**, the values of which are passed into nested child jobs.

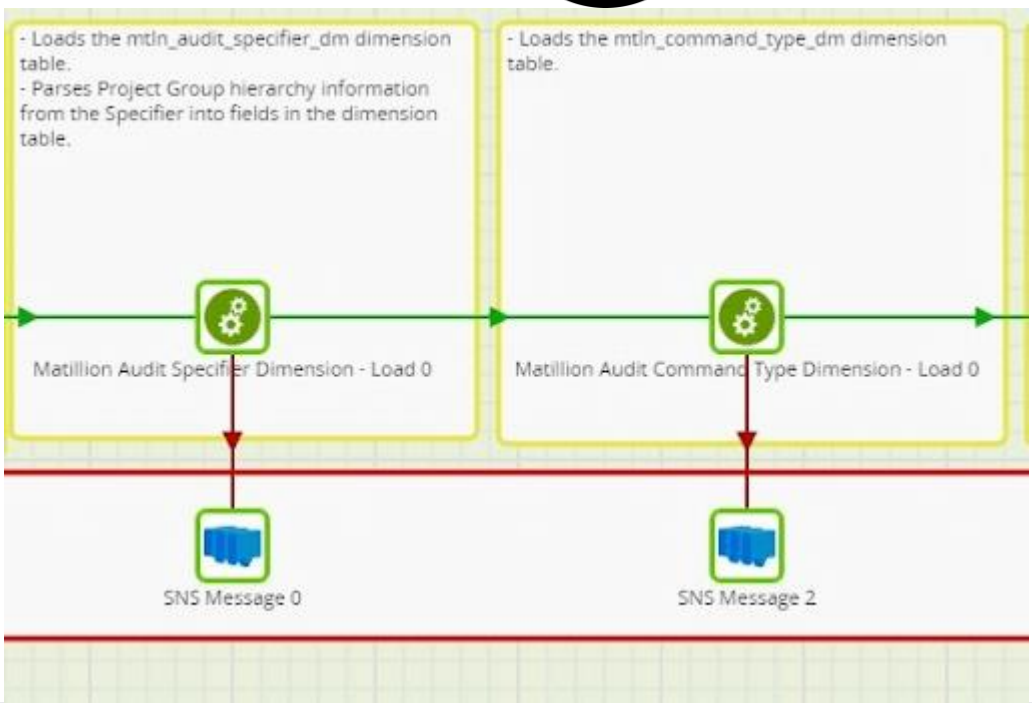
- **curr_ts** – A current timestamp that is used throughout a job execution. Value is calculated once at runtime, **as described in Part 2 of this series.**
- **max_audit_ts** – The maximum timestamp of Audit Log events that are in the target Fact table. This is a numeric because Audit Log event timestamps are stored in Unix timestamp format.
- **message** – Used to store the message from the execution of a component. Used for notifications sent through an **SNS Message** component.
- **tzone** – Defines the timezone in which to calculate the current timestamp (curr_ts) in.

Notifications



In the Main Orchestration Job, the **SNS Message** component is used to notify when any step in the Orchestration fails. The ${message} job variable is used to store the message returned from the component that has failed. The message and other metadata from the component execution can be passed to a variable through the "Export" tab of the Component.

Dimension Tables



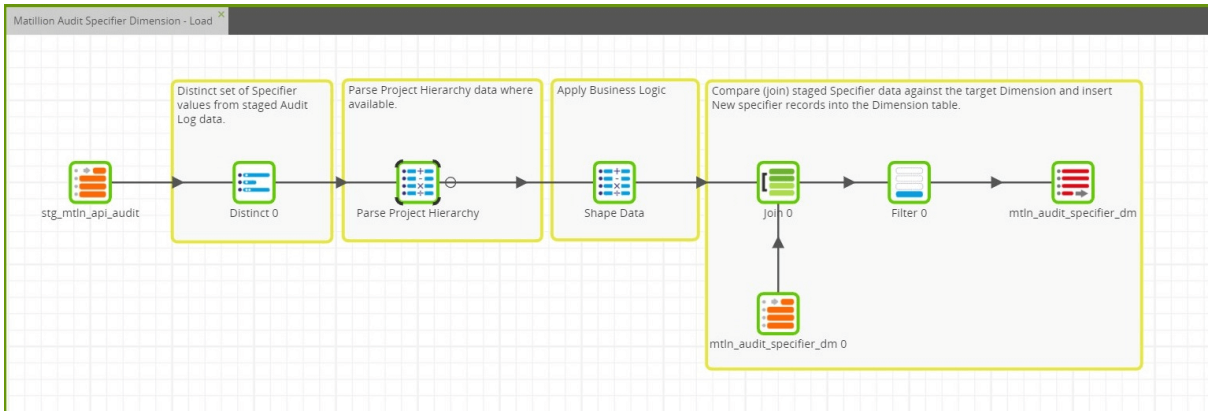
Job Design

Dimension Keys

Both Dimension tables that are populated from data coming from Audit Log events have a surrogate key added that is a result of hashing the natural key. The details of this was covered in **Part 2 of this technical article series** when detailing the User Dimension.

Adding a surrogate key in this manner can make it simpler when referencing the data in these Dimensions in other related Facts. For instance, if the Dimension tables are Type 0 or Type 1, when deriving the Dimension Keys for populating the Fact table(s), it is not necessary to lookup the Dimension Key from the Dimension table. The Dimension Key can be calculated by using the same hashing algorithm on the natural key data, which typically exists in the raw data being loaded into the Fact table.

SCD Type 0 – Audit Specifier Dimension (mtln_audit_specifier_dm)



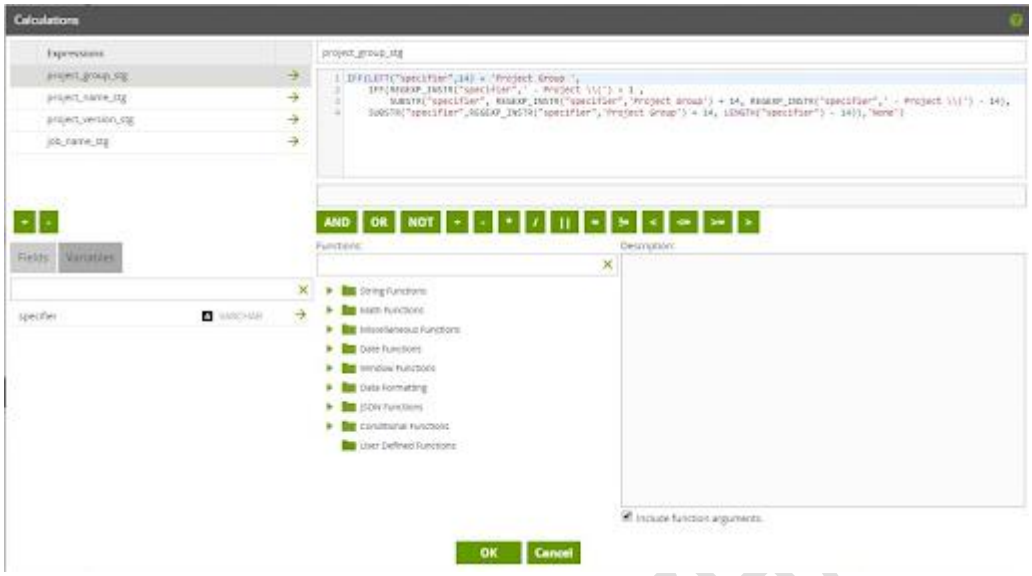
For every Audit Log event there exists a "Specifier" attribute. This attribute contains semi-structured data that defines the area within Matillion ETL that the audit event took place within. In the Audit Specifier Dimension table, there will be one row per distinct Specifier. Also, because of the nature of the Specifier data, we can treat the Audit Specifier Dimension as a Type 0 SCD. There will be no updates to a record in this dimension once an entry has been created.

Parsing Project Hierarchy

The Specifier can contain Project hierarchy information, specifically Project Group > Project > Version > Job, where applicable. Also, every level of the Project hierarchy returns a Name and an ID. The ID is particularly helpful as its value does not change, even if the Name of the entity does change. The Transformation job that manages the Audit Specifier Dimension parses the Project hierarchy information from the Specifier so that they can be stored into explicit fields in the Dimension table. This will prove useful when summarizing or pivoting the Audit Log fact data around any particular level of the Project hierarchy.

In this case, Snowflake is used as the target cloud data warehouse. So, within a **Calculator** component, **Snowflake REGEX functions** were used to help with parsing the Project Hierarchy data from the Specifier data. Similar functions are available in Redshift and BigQuery as well, although the exact syntax will differ between cloud data warehouse platforms. To keep the overall parsing logic simpler to understand and debug, the parsing logic was broken out into two steps.

The first step in parsing the Project Hierarchy data isolates the data for each level of the hierarchy, returning the Name and ID information for each level.



Example output at this stage:

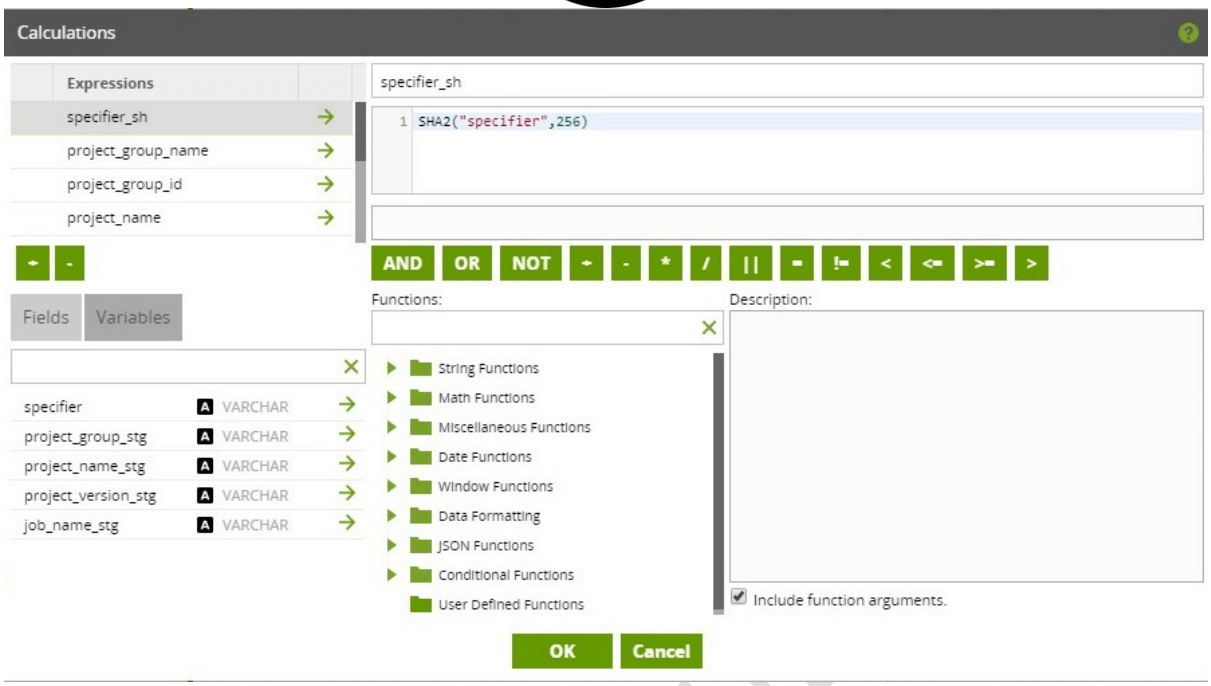
Properties | Sample | Metadata | Lineage | SQL | Help

Data | Row Count | Filter Not Set

specifier	project_group_stg	project_name_stg	project_version_stg	job_name_stg
Project Group (Demons...	(Demonstration Projects) [60]	(Dan DOrazio) [288542]	(default) [288543]	(Demo flights) [305...
Project Group (Demons...	(Demonstration Projects) [60]	(Arawan Gajajiva) [165072]	(default) [165073]	(xform flights) [198...
Project Group (Demons...	(Demonstration Projects) [60]	(Arawan Gajajiva) [165072]	(default) [165073]	(xform flights) [198...
Project Group (Demons...	(Demonstration Projects) [60]	(Veronica Kupetz) [345556]	(default) [345557]	(Demo ELT) [345623]
Project Group (Demons...	(Demonstration Projects) [60]	(Veronica Kupetz) [345556]	(default) [345557]	(Demo ELT) [345623]
Project Group (Demons...	(Demonstration Projects) [60]	(Veronica Kupetz) [345556]	(default) [345557]	(Demo ELT) [345623]
Project Group (Demons...	(Demonstration Projects) [60]	(Damian Chan) [293461]	(default) [293462]	None
Project Group (Demons...	(Demonstration Projects) [60]	(Veronica Kupetz) [345556]	(default) [345557]	(Demo ELT) [345623]
Project Group (Demons...	(Demonstration Projects) [60]	(Damian Chan) [293461]	(default) [293462]	(Temp Orch) [35881...
Project Group (Demons...	(Demonstration Projects) [60]	(Veronica Kupetz) [345556]	(default) [345557]	(Demo Transformat...
Project Group (Demons...	(Demonstration Projects) [60]	(Damian Chan) [293461]	(default) [293462]	(Temp Orch) [35881...
Project Group (Demons...	(Demonstration Projects) [60]	(Damian Chan) [293461]	(default) [293462]	(Temp Orch) [35881...
Project Group (Demons...	(Demonstration Projects) [60]	(Damian Chan) [293461]	(default) [293462]	(Temp Orch) [35881...
Project Group (Demons...	(Demonstration Projects) [60]	(Damian Chan) [293461]	(default) [293462]	(Temp Orch) [35881...

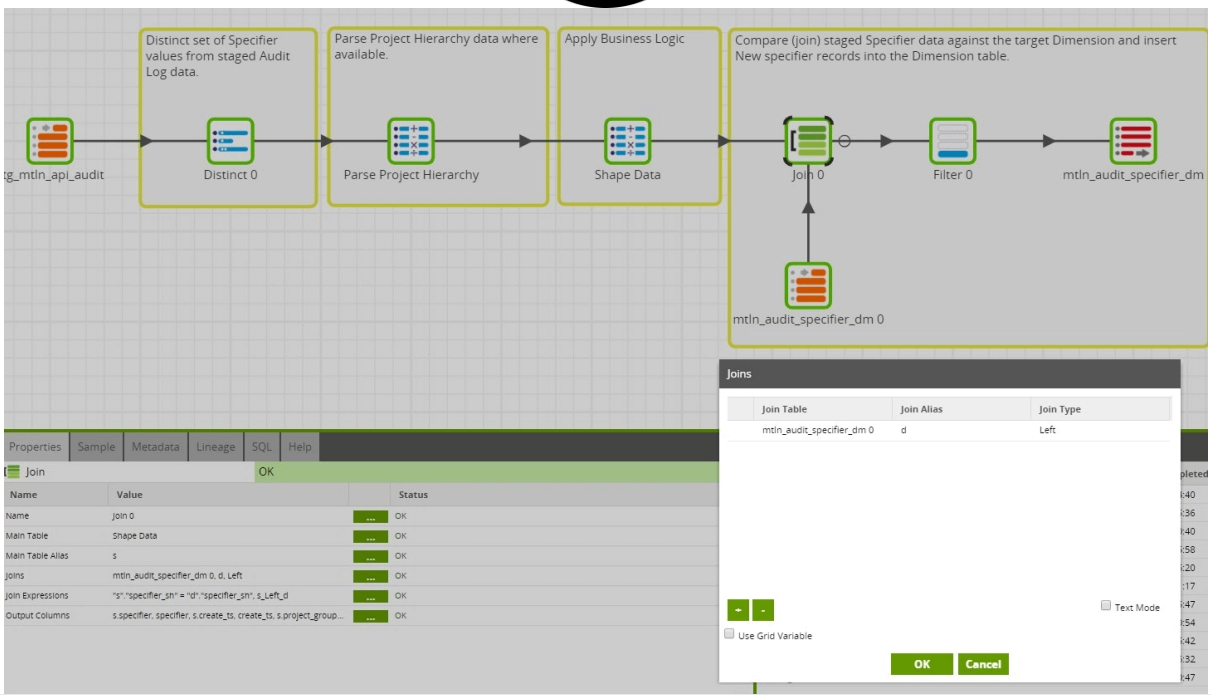
Shape Data

The next step in the Transformation Job is to “shape” the incoming data, which further parses the Project Hierarchy attributes and adds dimension attributes, such as deriving a hash for the specifier and the addition of a create timestamp field.

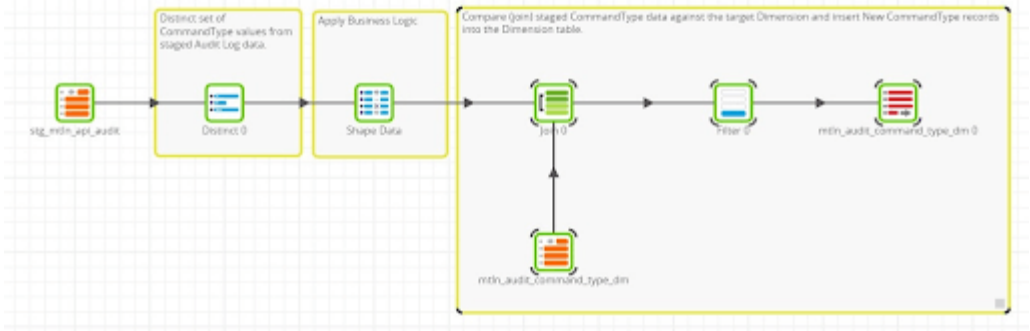


Apply Changes

Because this is a Type 0 SCD, there will only ever be inserts performed against this Dimension table. As such, we do not have to worry about accounting for race conditions in the job design. Therefore, applying the DML to the Dimension table can be done in the same Transformation job. Because we are only interested in identifying data to be inserted, instead of using a Detect Changes component, a Left Join is used.

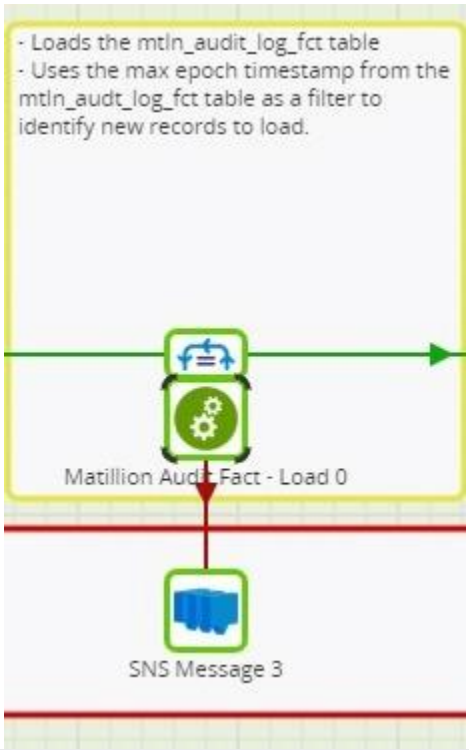


SCD Type 0 – Audit Command Type Dimension (mtln_command_type_dm)



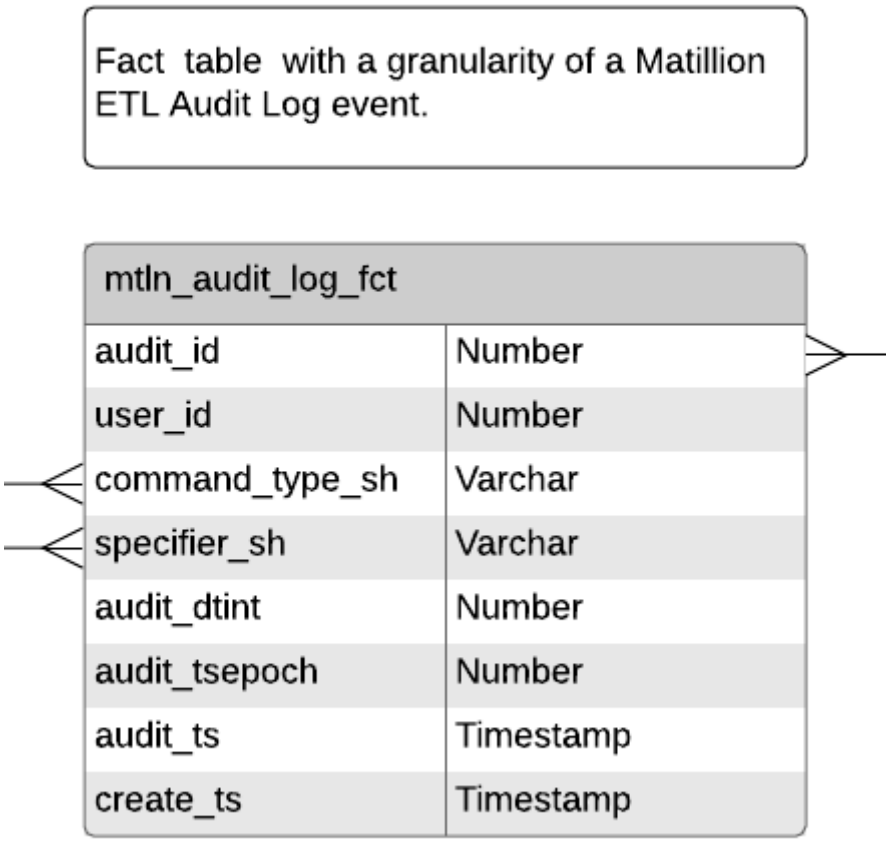
As can be seen when comparing the two Transformation Jobs, the design pattern for managing the Audit Command Type Dimension is almost identical with how the Audit Specifier Dimension is managed. The only difference is the extra transformation logic in the Audit Specifier Dimension for parsing Project Hierarchy attributes.

Fact Table



Audit Log Event Fact (mtln_audit_log_fct)

Table Metadata



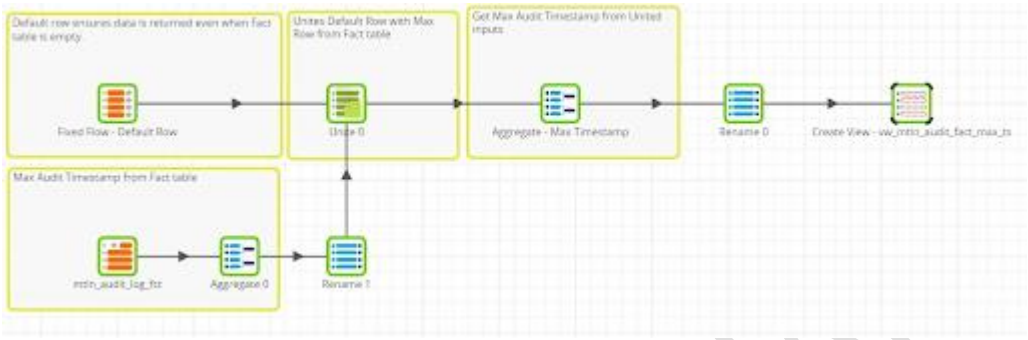
High Water Mark

When fetching Audit Log data from the Matillion API v1, all Audit Log records from Matillion are staged. While Matillion ETL does purge Audit Log history, there will typically be Audit Log events staged that have already been loaded into the Audit Log Event Fact table. To ensure these rows are not loaded into the Fact table again, we use a common **High Water Mark design pattern**.

The general idea is to identify a field in the Fact table that can be used as a "high water mark" for data in the table. In the case of Matillion Audit Log data, there is an "ID" which is unique per Audit Log Event and there is an Audit Timestamp that represents the date and time of the Audit Log Event. For the purpose of using one of these as the "high water mark", we use the Audit Timestamp. Since the Audit Timestamp is a Unix timestamp, we can treat it as a numeric value that is sequential in nature.

To implement the High Water Mark design pattern, we do the following things:

1. Create a view in the cloud data warehouse that returns the “max” value of the high water mark column from the Fact table. (see the attached example job)

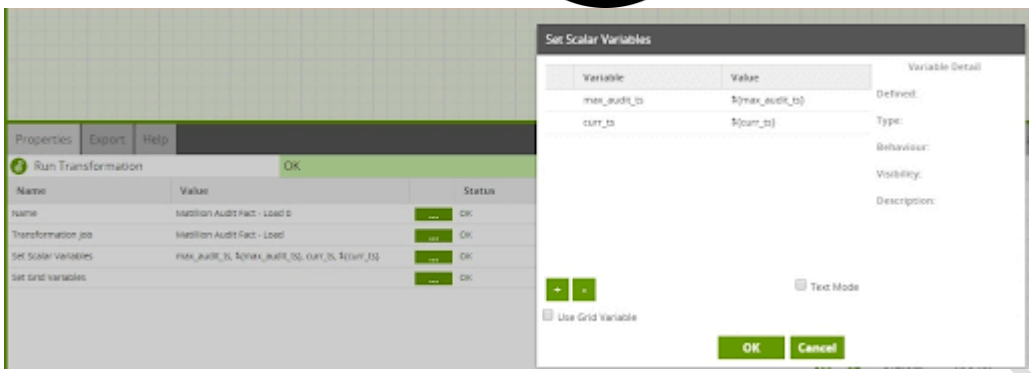


2. Use a Job Variable to store the current high water mark value and a Table Iterator in the Main Orchestration to populate the Job Variable at runtime.

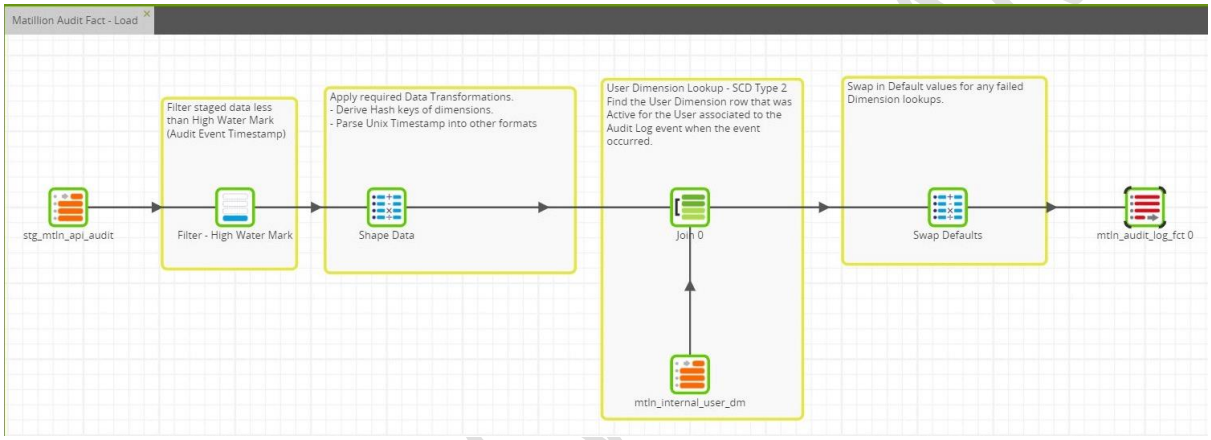
Table Iterator — OK

Name	Value		Status
Name	Table Iterator 0	...	OK
Database	[Environment Default]	...	OK
Schema	[Environment Default]	...	OK
Target Table	vw_mtln_audit_fact_max_ts	...	OK
Concurrency	Sequential	...	OK
Column Mapping	max_timestamp, max_audit_ts	...	OK
Order By		...	OK
Sort	Ascending	...	OK
Break on Failure	No	...	OK
Record Values in Task History	Yes	...	OK

3. Pass the value of the Job Variable to the nested Transformation Job that populates the Fact table.



Load Fact Data



The Transformation job that loads data into the Fact table does the following things:

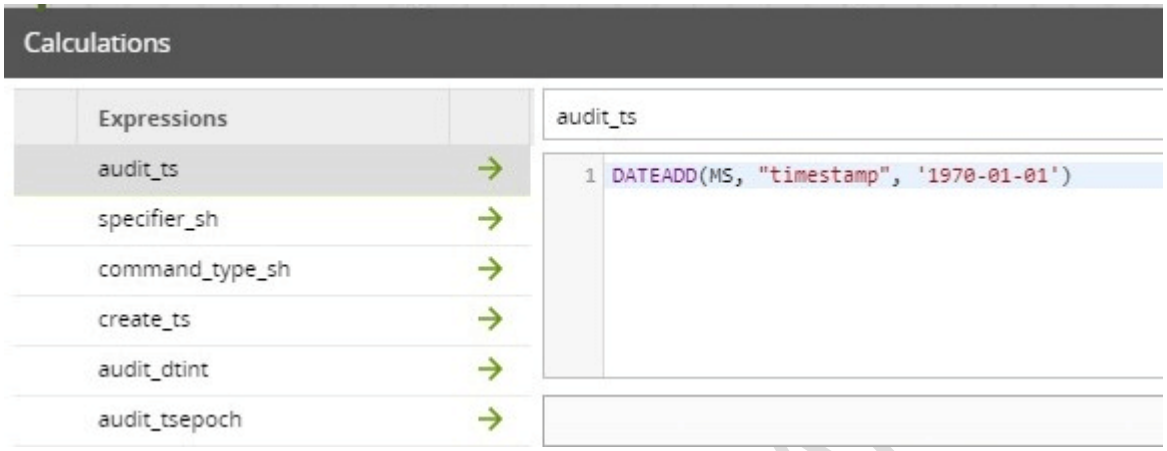
1. Filter staged data based on the High Water Mark (Audit Event Timestamp)
2. Shape the data
3. Perform Dimension Lookups
4. Swap in Defaults
5. Insert into the Fact table

Filter Staged Data

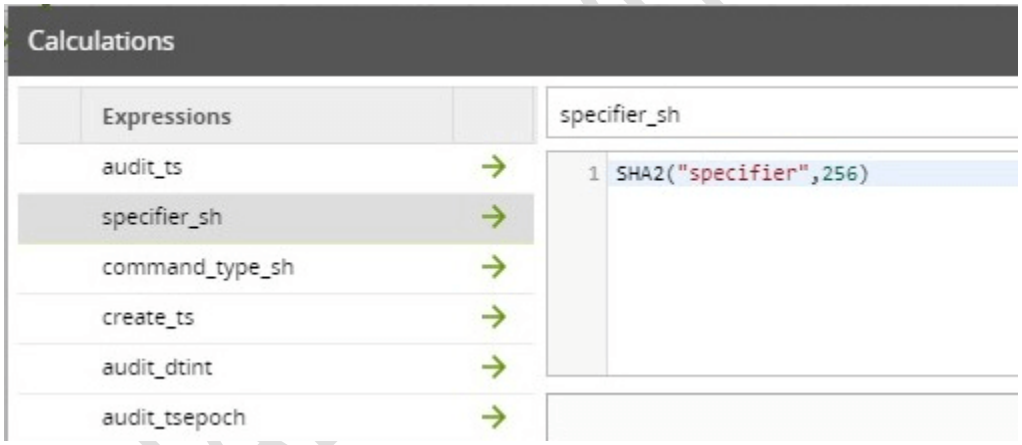
The Transformation job gets the High Water Mark value passed to it from the Orchestration that calls it. The Job Variable is used in the Filter component to filter the staged data based on the current High Water Mark.

Shape Data

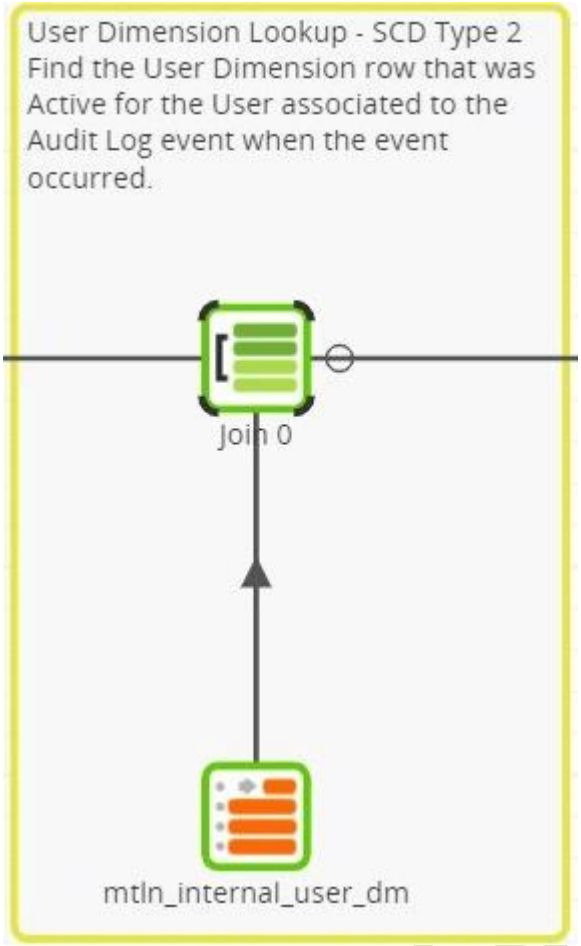
In this step, a **Calculator** component is used to leverage Snowflake functions to apply transformation logic to "shape" the data. Transforming the Unix timestamp into a standard timestamp format is an example of this:



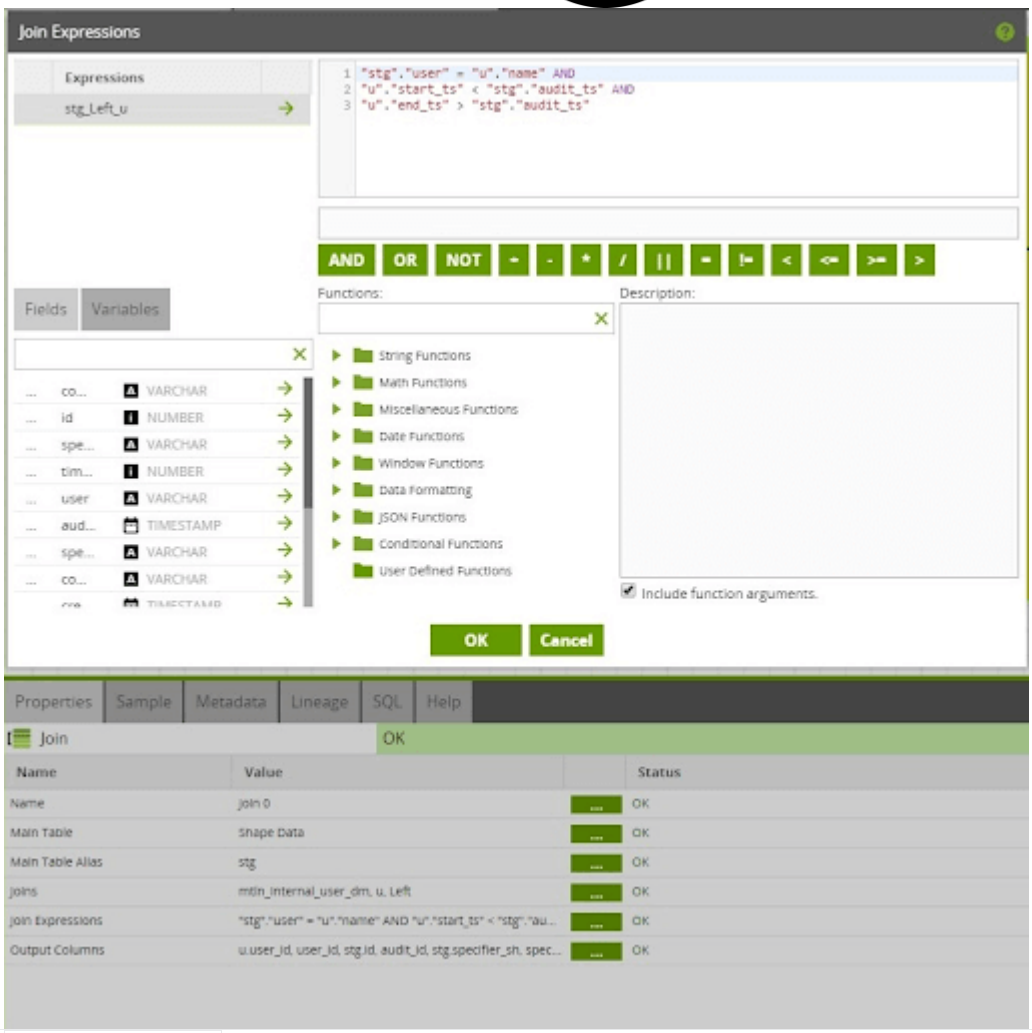
Also note this is the step where the Type 0 dimension keys are derived. This is done by generating the hash of the natural key:



User Dimension Lookup

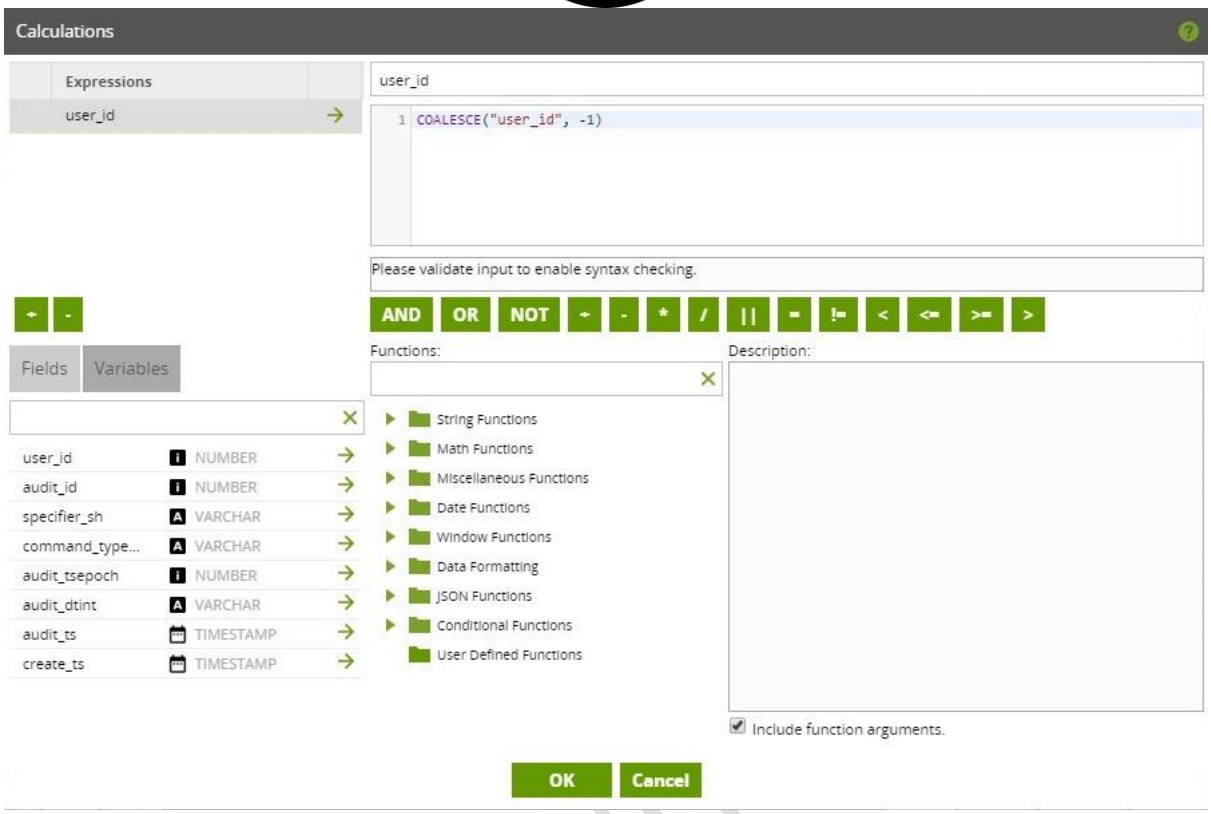


Since the User Dimension is a Type 2 SCD, deriving the hash of the natural key is not sufficient. A lookup on the User Dimension table is required to identify the dimension row that represents the User record that was active at the time of the Audit Log event. The logic required to do this lookup is defined within the Join Expression of the Join component:



Swap Defaults

It is generally best practice to never have any NULL values stored in a Fact table. With this in mind, another Calculator component is used to swap in a default value for any dimension keys that could not be derived. All Dimension tables were initialized with a default row that represents an “Unknown” entity in the Dimension. Specifically, in this job, a default row is swapped in for any failed lookups on the User Dimension.



Summary

In this article we showed how to use Matillion ETL to populate a Fact and Type 0 Dimensions with Matillion Audit Log data. By decomposing the related jobs, we showed techniques in Matillion ETL around how to implement common tasks in SCD design patterns. Through all of the topics discussed in this 3 part series, we provided a way of automating the consumption and transformation of Matillion User and Audit Log data into an analytics ready format. With these steps, you can implement a key detective security control around your Matillion ETL architecture, and accomplish an important step in your overall enterprise security plan.

Reference :

https://www.matillion.com/blog/matillion-security-controls-enable-user-auditing-part-3-of-3