Matillion Security Controls: Enable User Auditing (Part 2 of 3)

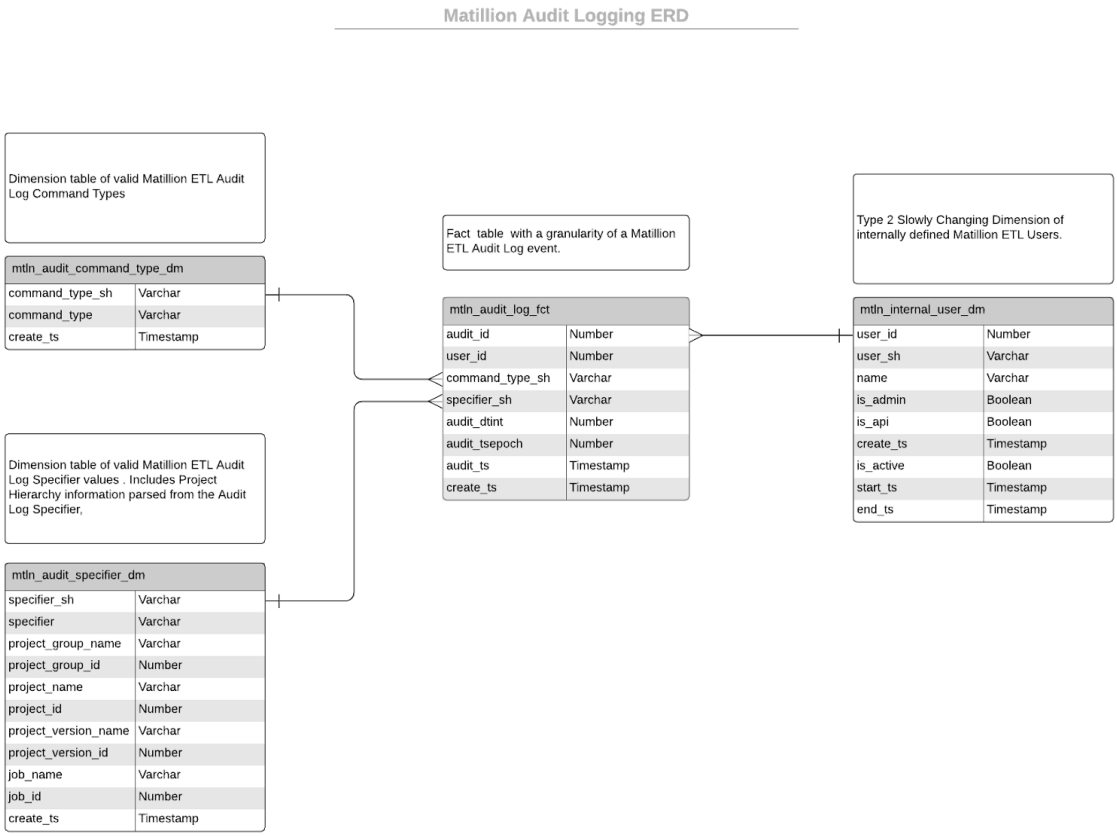
User Auditing Part 2 – Dimensional Modeling and Implementing Type 2 Dimensions

In **Part 1 of this series** on Matillion security controls and how to enable user auditing, we showed how to use Matillion ETL to integrate with a couple Matillion v1 API endpoints as sources of User and Audit Logging data. In Part 2 of this series, we show how to use Matillion ETL to implement a common data design pattern, Dimensional Modeling, to get the most insights out of this User and Audit Logging data. If Dimension Modeling is new to you, or would like to learn more about this frequently used data modeling technique, Ralph Kimball's book, **The Data Warehouse Toolkit,** is a great resource!

Data Model

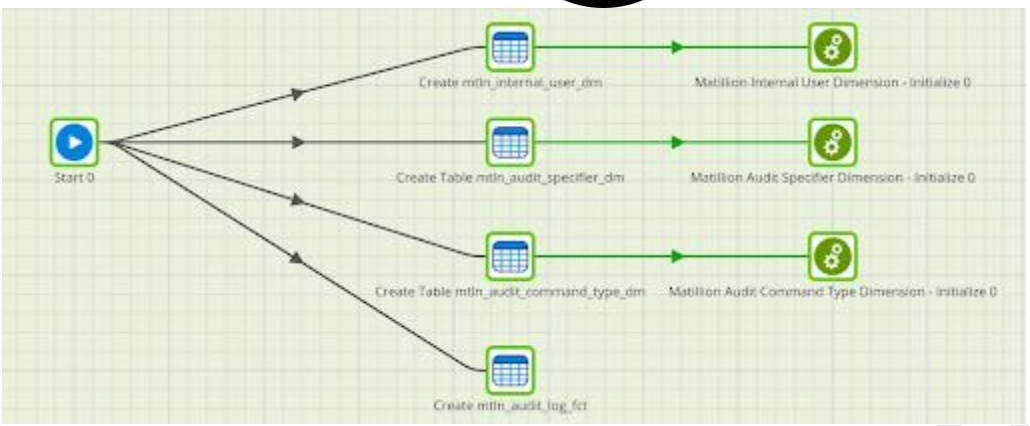
A traditional Fact/Dimension model is a good fit for storing user audit information for advanced reporting. But, before we can start developing the Matillion ETL jobs to manage Fact and Dimension tables, we need to first

design these tables. User audit events can be represented as a “fact” that occurred at a specific time, with related “dimensions” that describe aspects of the audit event. Additionally, each “user” can have some access related attributes that can change over time. As such, a **Type 2 Slowly Changing Dimension** is a nice way to capture how a user’s attributes can change over time while preserving that history of change.

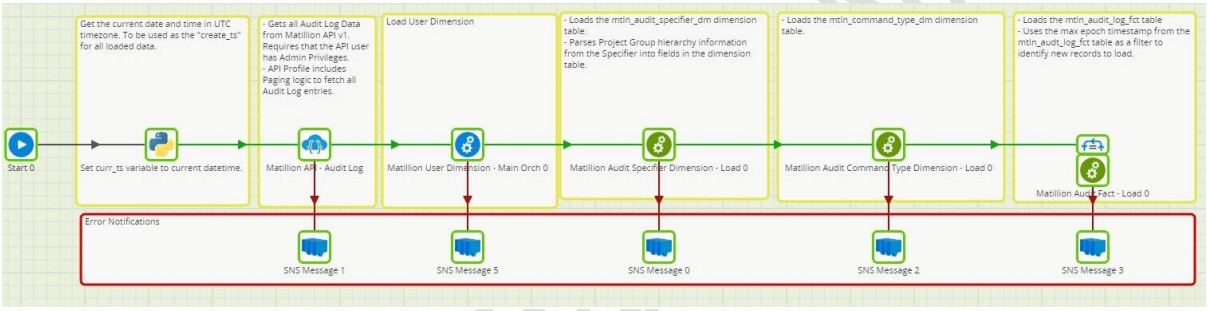


Managing DDL with Matillion ETL

The “Security DDL Setup” Orchestration Job creates all of the Fact and Dimension tables represented in the Entity Relationship Diagram (ERD) shown above. Additionally, the Orchestration Job calls Transformation Jobs that help to initialize the Dimension tables with default data that can be used to represent an “Unknown” or failed lookup against the Dimension table.



Main Orchestration



As is the case for all Matillion ETL jobs, there is one Main Orchestration Job that executes Components and other nested Orchestration and Transformation Jobs. This is the main orchestrator of the workflow, defining dependencies and order of events. Dependencies are defined based on the **output nubs** of each component within the job. Notifications are also included in the job design, using an **SNS Message** component, to notify when any specific step in the Orchestration has failed.

SCD Type 2 – User Dimension (mtln_internal_user_dm)

The jobs that manage the User Dimension table demonstrate common Matillion ETL techniques and design patterns for managing Slowly Changing Dimensions (SCD). In this example, we demonstrate how to implement a **Type 2** Slowly Changing Dimension workflow.

Table Metadata

Note the following SCD-specific fields added to this dimension table:

- **is_active** – This boolean field is used to identify the currently active dimension row. This gives a view as to what the user and their related attributes look like currently.
- **start_ts** – This timestamp field defines when the dimension row was active, as of a certain time. When initially loading users, we will populate this field with a default timestamp in the past.
- **end_ts** – This timestamp field defines when the dimension row was active until. We will populate this field with a default timestamp in the future for currently active users and will update this value when the dimension row is no longer active.

Job Design

The primary Orchestration job that manages the User Dimension table does the following (in order):

1. Set values for job variables
2. Stages Matillion ETL User data into Snowflake
3. Prepares User Dimension data
4. Applies changes to the User Dimension table

Avoiding Race Conditions

An important aspect to understand about the demonstrated design pattern is the use of two Transformation Jobs to identify and execute data changes on a target table. The first Transformation Job evaluates the changes that need to be applied to the target table and stages the prepared data into an intermediary table. The second Transformation Job applies the staged data changes to the target table. This design pattern addresses the race condition that would otherwise be encountered when trying to apply multiple DML

operations (insert/update/delete) to a table that is also used as an input in the same Transformation Job. The race condition occurs when multiple DML operations are executed as separate statements and the first executed DML affects the data set used in the subsequent DML operations.

The nature of this Type 2 Slowly Changing Dimension is such that there may be more than one type of DML operation that needs to be applied to the table. In this case, a set of data needs to be ***inserted*** (New or Changed records) and another set of data needs to be ***updated*** (Changed or Deleted records), both against the same Dimension table. This Dimension table is also used as an Input in the logic of the Prepare job. To avoid a race condition on applying changes back to the Dimension table, the changes to the Dimension table are first staged via the Prepare Transformation job and then the DML statements are executed in the Apply Transformation job.

Set Job Variables

Here we are using a **Python Script** component to set the value of **Job Variables** that are used throughout the User Dimension jobs.

Current Timestamp

First, we are getting a current timestamp that will be used throughout the rest of the job in a job variable named "curr_ts". The reason for setting a current timestamp in this manner (as opposed to using Snowflake's CURRENT_TIMESTAMP function in the child Transformation Jobs) is so that we have a single consistent timestamp for all data affected by this run of the job. Also note that there is a Job Variable named "tzone" to define the timezone that the current timestamp should be generated for. We are using UTC timezone in this example.

```
import pytz from datetime import datetime tz = pytz.timezone(tzone) curr_time = datetime.now(tz)
curr_time.strftime('%Y-%m-%d %H:%M:%S') context.updateVariable('curr_ts',str_time
```

Initialize Dimension

Next, because we are using the Jython interpreter, we are able to directly query the User Dimension table to determine if this is an initial run of populating the dimension table. The "init_dm" job variable will let us know if

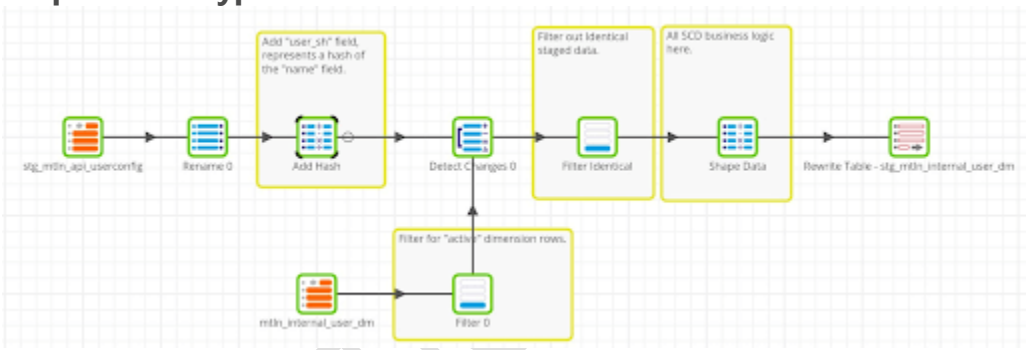
this is an initialization of the dimension. In a subsequent Transformation we will use this information to determine if the “start_ts” column for new dimension records should be set to a default timestamp in the past or if the current timestamp should be used. This is a good example of using a Python Script component to easily set variable values at runtime.

```
cursor = context.cursor() cursor.execute('select count(*) from "mtln_internal_user_dm"') rowcount = curs
                                                    rowcount > 0:   context.updateVariable('init_dm',0)
```

Stage User Data

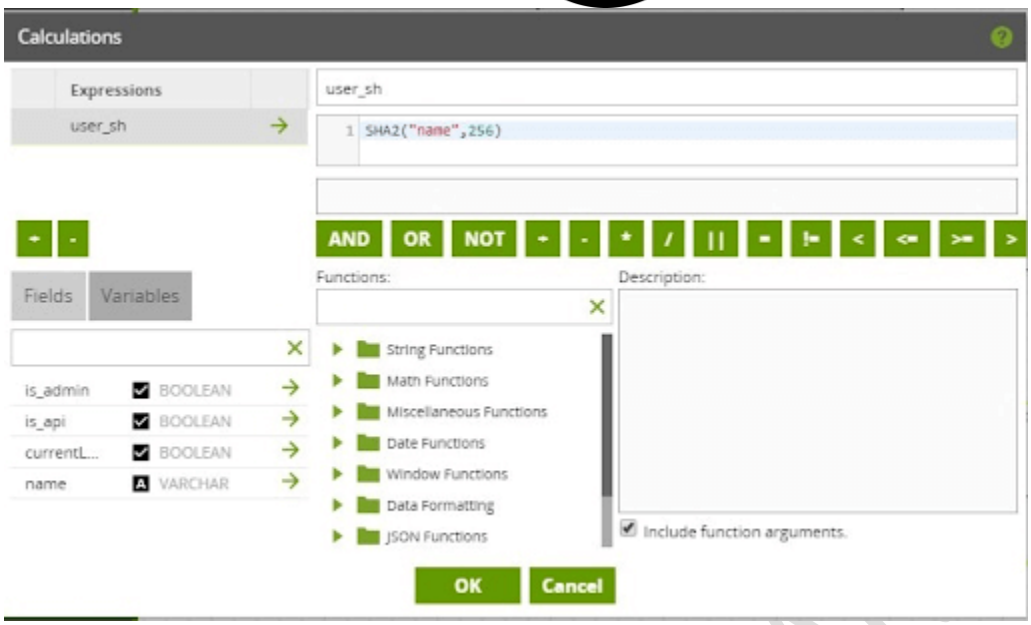
In this step an API Query component is used to fetch all User data from Matillion and stage it into a table. The API Profile used as the data source, **userconfig**, was detailed earlier.

Prepare SCD Type 2 Data



The general Matillion ETL design pattern for implementing slowly changing dimensions has previously been detailed in the article **Type 6 Slowly Changing Dimensions Example.** See that article for additional examples of implementing SCD design patterns. In this Transformation Job, the incoming staged data is compared against the target Dimension table. Any new, changed or deleted records have some data transformations applied to them and the resulting data set is then staged into an intermediary table, to be used in a subsequent Transformation Job.

Prepare: user_sh



A specific design decision was made in the dimension tables to add a column that is populated with a hashed string that represents a unique entity in the dimension. That unique entity can be represented by a composite or single key, where the related fields are passed through a **SHA-2** algorithm to return back a hashed string. Having this hashed string makes it simpler, and more consistent, to design a common pattern for comparing incoming staged data against the target dimension table. In the case of the User Dimension, the "name" field is what uniquely identifies an entity in the dimension. The hashed "name" string is then stored in the "user_sh" field. Because this is a Type 2 SCD, a single user record can have multiple rows in the User Dimension table (as the attributes of that user changes over time). All occurrences of the same user in the User Dimension will therefore have the same "user_sh" value.

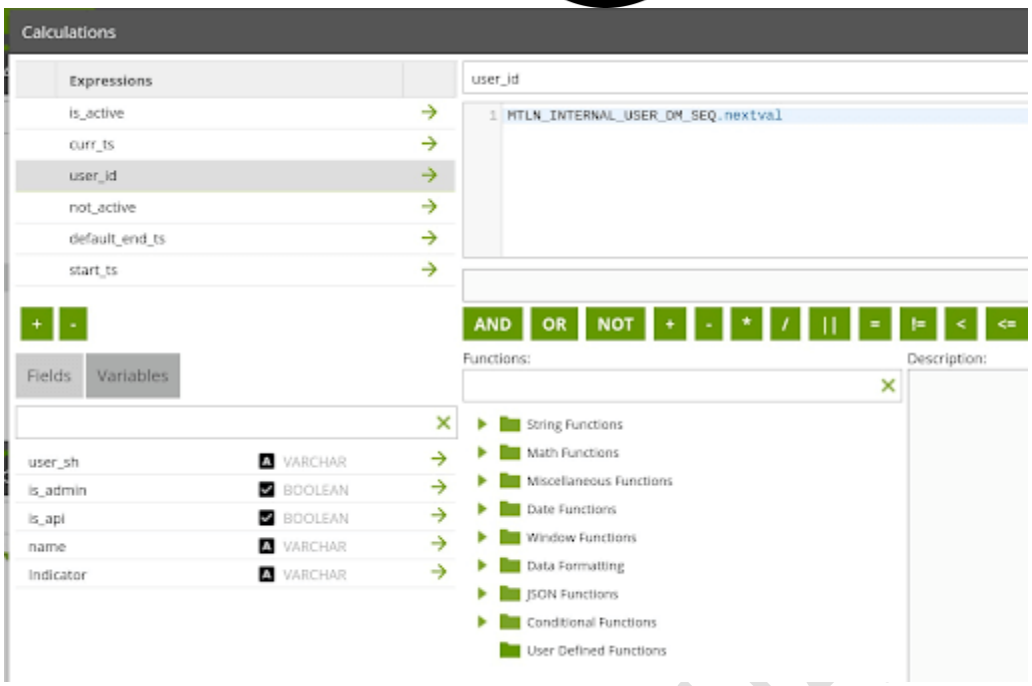
Prepare: Detect Changes

One of the most important Matillion ETL components in this design pattern is the **Detect Changes** component. This component allows us to compare the incoming staged data against the target dimension table and add an indicator for every row of staged data to denote if it is New, Changed, Deleted or Identical. When comparing the incoming staged data against the data in the target dimension table, we only need to look at the currently active dimension row for each user. And, for the purposes of populating the User Dimension table, we can filter out any Identical rows of data. Also note that when joining the data sets, the hashed field (user_sh) is used.

Prepare: Shape Data

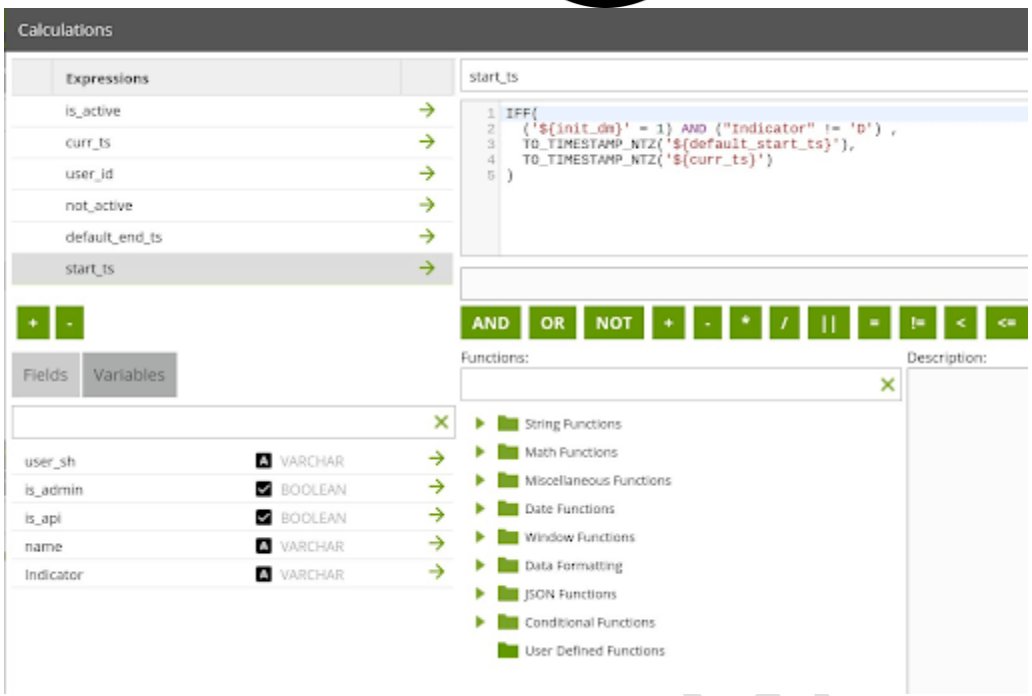
Once we have filtered out Identical records from the incoming staged data, a **Calculator** component is used to add new fields or modify existing fields with specific business logic. This is what we call "shaping" the data. All data transformation business logic is encapsulated into this single component so as to make the overall job design simpler to manage. The resulting filtered and transformed data is then staged into an intermediary table, which will be used in a subsequent Transformation Job. A couple things to call attention to in this step are as follows.

Prepare: user_id



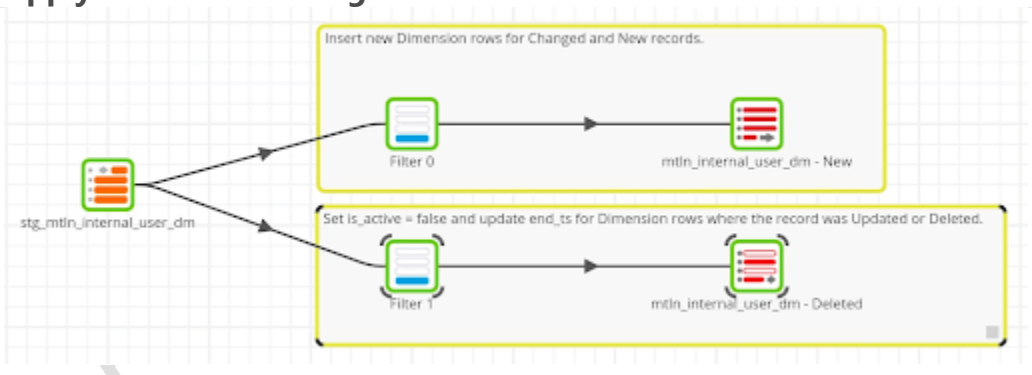
A “user_id” field has been added, which uniquely identifies a single record in the User Dimension table. In this instance, a strong **Snowflake Sequence** named “MTLN_INTERNAL_USER_DM_SEQ” was created and used here to assign “user_id” values. As previously mentioned, because this dimension is Type 2, a single User can have multiple entries in this dimension table. The “user_id” field will be referenced in the related Audit Log Fact table, allowing the association of an Audit Log event to a specific User record from a point in time.

Prepare: start_ts



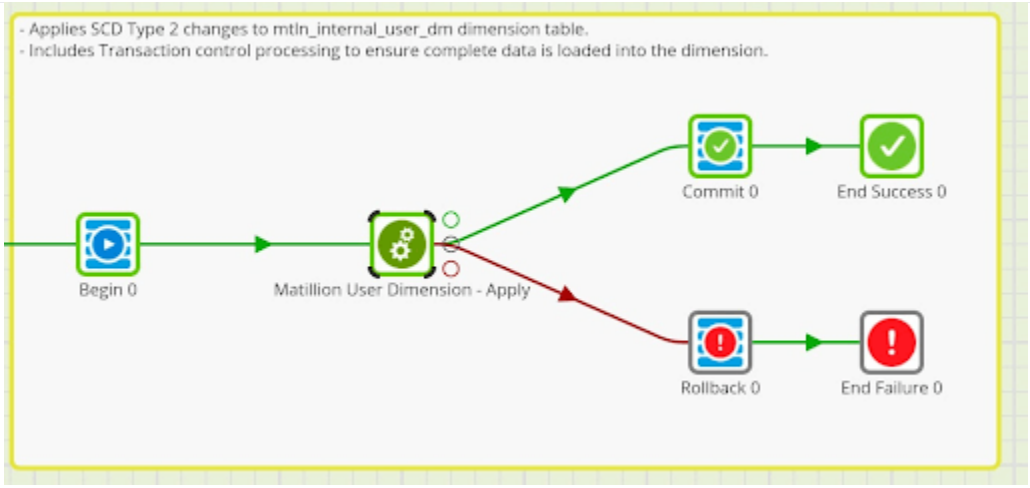
The “start_ts” SCD field represents when the Dimension row was active as of. The logic in the Calculator expression for this field accounts for if the User Dimension is being initialized. If the dimension is being initialized, the “start_ts” will get a default timestamp in the past. Otherwise, it will get the value of the current timestamp.

Apply Dimension Changes



This Transformation Job takes the previously staged SCD Type 2 data and applies the appropriate data changes to the User Dimension table. Staged records will represent New, Changed or Deleted records. For New records, a row will be inserted into the User Dimension table. For Deleted records, the existing “active” row (is_active = TRUE) will be updated. For Changed records, a new “active” record is inserted and the previously “active” record is updated.

Transaction Control



The Orchestration Job that calls the Transformation Job to Apply the SCD data changes includes Transaction control components (**Begin**, **Commit**, **Rollback**). When the Transformation Job executes, there are two separate SQL queries executed in the cloud data warehouse, one SQL query per output. The Transaction control logic ensures that when applying the SCD data changes, all operations must succeed for the data changes to be applied. If there is a failure in one of the two outputs in the Transformation Job, nothing will get applied. This ensures that the data in the User Dimension table does not get into an undesirable state (i.e. more than one "active" record for a user).

Summary

In this article we defined a dimensional data model for storing Matillion ETL UserConfig and Audit Log data to make our data analytics ready. Then, we took a detailed look at a Matillion job that populates and manages Matillion User data in a Type 2 Slowly Changing Dimension. By decomposing this job, we touched on techniques and points of consideration when implementing SCD design patterns in Matillion ETL. In the final part of our 3-part series on Matillion security controls and user auditing, we will provide a similar detailed look at using Matillion ETL Audit log data to populate Facts and Type 0 Dimensions.

To learn more about security controls and cloud security, download our ebook, **The Data Leader's Guide to Cloud Security and Data Architecture.**

Reference :

https://www.matillion.com/blog/matillion-security-controls-enable-user-auditing-part-2-of-3