Denodo Logic Replacement

1. Summary

Detailed below is a proposed approach for migrating the current transformation logic from Denodo to Snowflake. The process can be broken into 4 main parts.

1. Extracting the transformation view logic from Denodo.
2. Cleaning and refactoring the logic to be Snowflake compatible while preserving the business logic.
3. Deploying the views to Snowflake without breaking view dependencies.
4. Validating the logic is successfully refactored into snowflake.

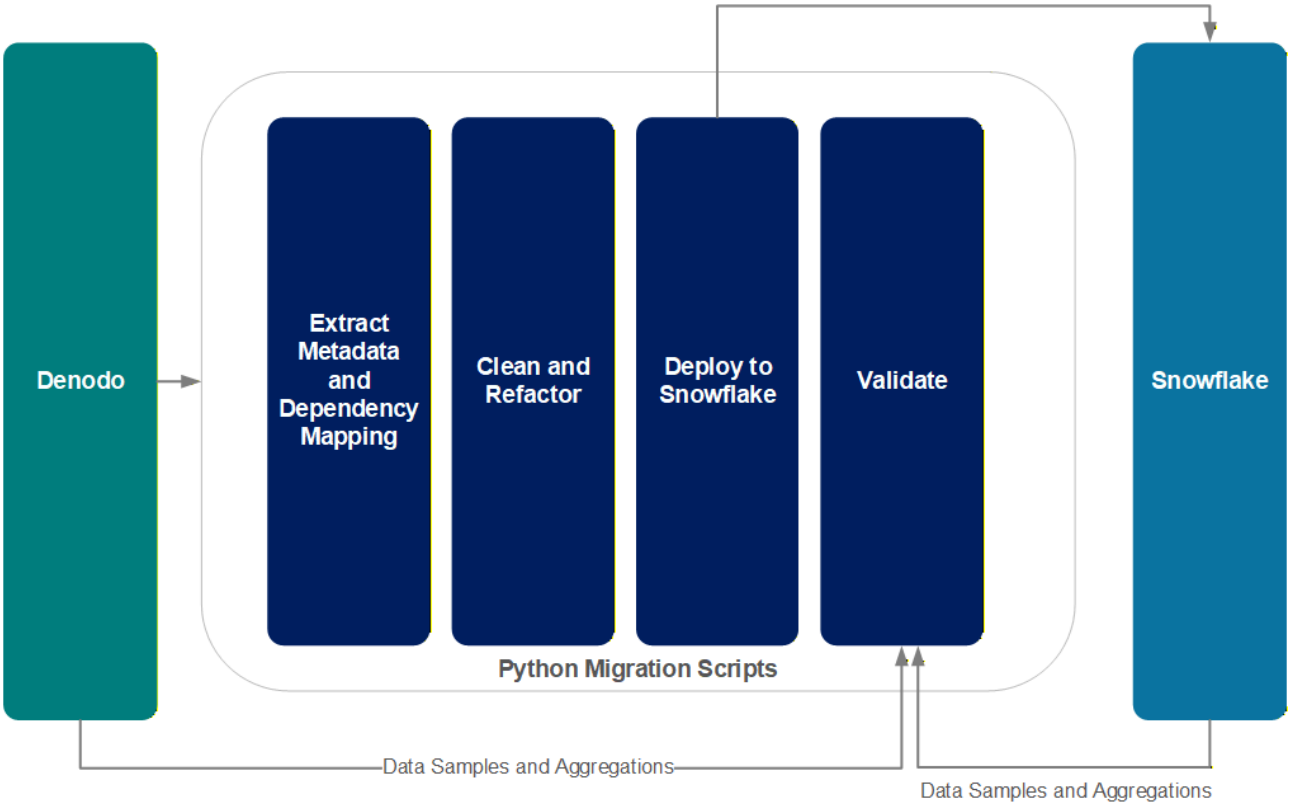

Figure 1: High Level Overview

2. Phase 1: Denodo Logic Extraction

2.1. Summary

We will automate the extraction of transformation logic from Denodo for accuracy and repeatability. Instead of manually copying SQL code, a script will connect to the Denodo system and download the extract the metadata directly.

This will reduce the potential for human error when copying the extremely large number of views and integration logic.

Denodo views are built in layers; starting from the raw base layers, progressing in complexity as more integration logic is used to maniupate these base views and create more views on top of them forming a hierarchical strcuture.

To simplify this complexity, we will programmatically capture a views dependencies by using built in Denodo functions and then ordering the views by their dependencies. In this way we can recreate the views in the correct order within Snowflake.

2.2. Technical Implementation

We can use the Denodo ODBC driver combined with the turbodbc library to connect to Denodo. This allows for OAUTH integration, which is necessary as the current Denodo environment does not allow plain-text password gerneration for security.

The extraction script will use this ODBC connection to query Denodo's internal stored procedures:

- `GET_VIEWS()`: Gets the VQL definition for virtual Denodo scripts. VQL being Denodo-specific SQL.
- `VIEW_DEPENDENCIES()`: Maps the lineage to ensure views are recreated in the correct order in Snowflake. There will be an additional sorting algorithm that we will use to ensure views are ordered correctly.

3. Phase 2: Logic Cleaning and Refactor

The SQL code exported from Denodo (VQL) is not directly compatible with Snowflake. It contains Denodo-specific metadata and their own syntax.

To rewrite these VQL scripts in Snowflake SQL we can automate some of the cleaning up by replacing Denodo functions and syntax with their Snowflake equivalents by using a mapping dictionary.

As well as this we will create a Raw Srouce to Denodo Name Mapping. Since Snowflake will now hold the raw data as it was landed from the source, we must account for naming discrepancies to make sure that the refactored SQL correctly points to the appropriate Snowflake objects while maintaining the busines naming conventions established in Denodo.

There may of course still be some manual work required in the cases where a function doesn't map 1-1 to a Snowflake equivalent. These functions that are not easily replaceable can be flagged with a script, so that manual attention can be given to them.

4. Phase 3: Creation in Snowflake

The next phase we will deploy the SQL scripts to snowflake in batches to avoid wasting processing capacity if there are any scripting mistakes and errors from the previous phases.

The scripts will be deployed with respect to the view hierarchy established in Phase 1, so that batches of views with no dependencies will be deployed first, followed by views that only have dependencies on those first set deployed and so on.

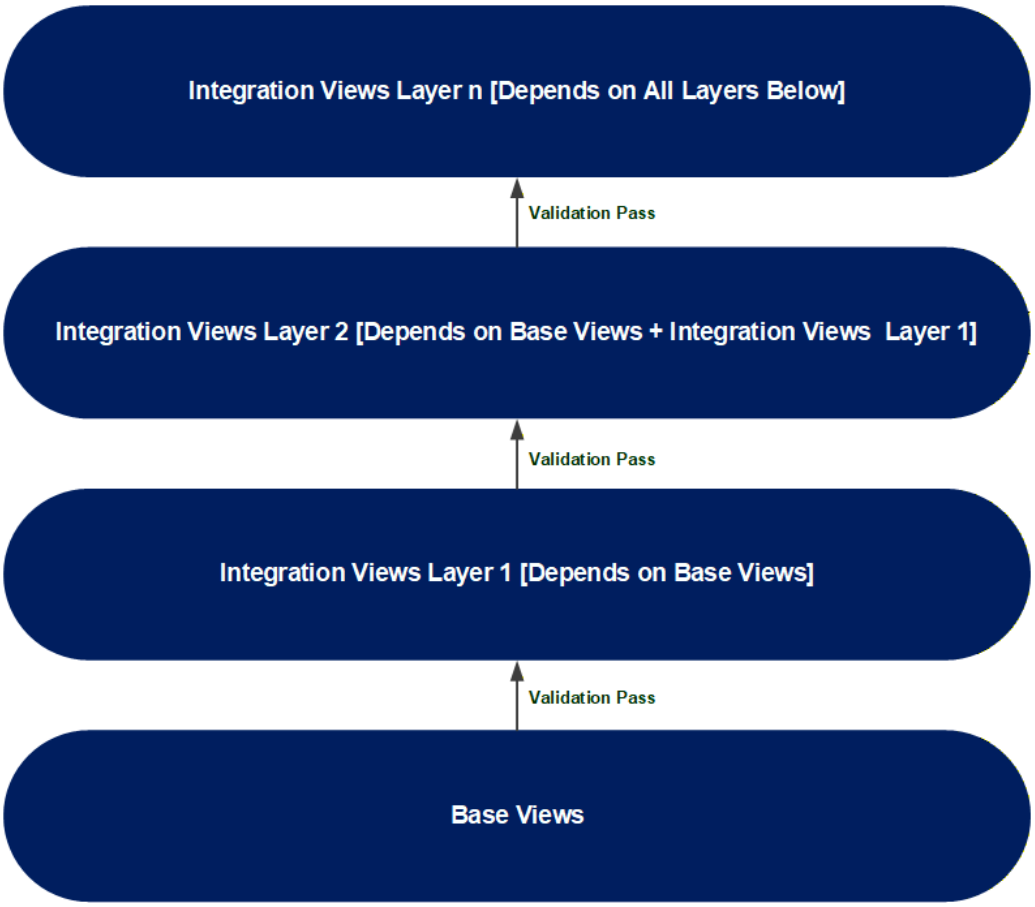

Figure 2: View Dependency & Deployment Hierarchy

5. Phase 4: Validation

This phase will run concurrently with the creation phase. We will validate each layer before proceeding to dependent layers. This will ensure that the foundational data is correct before additional complexity is added with views on top of other views, meaning that errors can be found sooner rather than later.

It may be possible to automate parts of this with python pulling data from Denodo and Snowflake, to run row checks, schema checks, and converting data to strings then using a hash algorithm to ensure that data from both systems is exactly the same.