# 1. Introduction

The **DEX Ingestion Framework** is a Python-based automation tool that converts data from an Excel file into YAML configuration files. It automates the process of generating YAML files for data ingestion tasks, such as loading data from a source database to Redshift. The framework also interacts with JSON files to dynamically update column names, add additional columns (ETL columns, OGG columns), and ensure data consistency based on specific conditions. By reducing manual effort and minimizing errors, the framework enhances efficiency in large-scale data ingestion workflows.

# 2. Project Overview

The **DEX Ingestion Framework** is designed to automate and streamline the ingestion of structured data from source systems into a target database (Redshift). This framework reads configuration details from an Excel template, processes metadata from JSON files, and generates YAML configurations and SQL queries for seamless data transfer.

# 3. Key Features

* **Excel-Based Configuration**: The framework extracts metadata from an Excel template, providing a structured and configurable approach to defining source and target database details. This reduces manual effort and ensures consistency across multiple ingestion workflows.
* **JSON Schema Processing**: Reads schema definitions from JSON files to dynamically structure the ingestion process. This allows for flexible schema management and adaptation to changing source table structures.
* **Automated Column Renaming**: Detects SQL reserved keywords in column names and automatically renames them to avoid conflicts. This ensures smooth execution without syntax errors.
* **YAML Generation**: Produces YAML configuration files (DBtoRedshift.yml & OGGToRedshift.yml) to define ingestion tasks. These YAML files are used for automating creating tasks and ensuring seamless data transfer to Redshift.
* **SQL Query Generation**: Automatically generates SELECT queries for view creation. It includes conditions for PII/SPII masking; ensuring sensitive data is protected during ingestion.
* **Secret Management Integration**: Fetches database credentials securely from a secret\_name.json file, preventing hardcoded sensitive information and improving security in the ingestion process.
* **Additional Columns**: If the source database type is not 'Sybase', the framework automatically adds ETL-related columns to track metadata such as created by, updated time, and change flags, aiding in auditability.
* **OGG Handling**: If the ingestion task involves OGG replication (OGGToRedshift), the framework ensures that OGG-specific columns like sequence numbers and timestamps are properly handled for accurate replication.
* **Historical Tables**: The framework supports the ingestion of archived or historical tables, creating separate ingestion configurations to handle historical data alongside active datasets.
* **Partitioning**: For large tables, automatic partitioning is applied based on reliable date columns and table size. This optimizes query performance and improves data retrieval efficiency.

# 4. Benefits

This framework offers several advantages:

* **Automation:** Eliminates manual YAML & SQL file creation, reducing human errors.
* **Scalability:** Easily adaptable to new tables and source systems.
* **Data Security:** Supports masking of PII/SPII data through SQL transformations.
* **Standardization:** Ensures consistent data transformation across ingestion pipelines.
* **Flexibility:** Allows customization based on source system types and ingestion needs.

# 5. How to Use the Framework

**Step 1: Prepare the Input Files**

* Place the ingestion metadata Excel file (DEX-Table\_Ingestion\_Template-V1.xlsx) in the working directory.
* Ensure that JSON schema files are available in the srcl folder.
* Verify that the secret\_name.json file contains correct database credentials.

**Step 2: Execute the Framework**

1. Run the main script:
2. The script will:

* Read the Excel file and extract table details.
* Process JSON schema files to extract column metadata.
* Generate YAML configurations (DBtoRedshift.yml, OGGToRedshift.yml).
* Generate SQL queries and store them in the srcl\_vw folder.

**Step 3: Review the Outputs**

* YAML Files: Used for configuring ETL jobs.
* SQL Files: Used for running transformations and secure data selection.
* Logs: Console logs help track processing steps and errors

# 6. Folder Structure

Excel to YAML Converter/

├── dex\_ingestion/ # Main Python package

│ ├── \_\_init\_\_.py # Makes the folder a Python package

│ ├── constants.py # Contains constants like KEYWORDS\_TO\_RENAME

│ ├── excel\_processor.py # Processes the Excel file

│ ├── main.py # Main script to run the project

│ ├── sql\_generator.py # Generates SQL files

│ ├── utils.py # Utility functions (e.g., rename\_column\_if\_keyword)

│ └── yaml\_generator.py # Generates YAML files

├── requirements.txt # Lists all dependencies

├── secret\_name.json # Contains secret names for source databases

├── srcl/ # Folder for input JSON files

│ └── ebt\_rm\_data.json # Example JSON file

├── srcl\_vw/ # Folder for generated SQL files (created automatically)

├── temp/ # Folder for updated JSON files (created automatically)

└── DEX-Table\_Ingestion\_Template-V1.xlsx # Input Excel file

**Explanation of Each File/Folder**

1. **Excel to YAML Converter/**:

This is the root folder of the project. It contains all the files and subfolders.

1. **dex\_ingestion/**:

This is the main Python package for the project. It contains all the Python scripts.

1. **\_\_init\_\_.py**:

This file makes the dex\_ingestion folder a Python package, allowing you to use relative imports (e.g., from .excel\_processor import process\_excel).

1. **constants.py**:

Contains constants like KEYWORDS\_TO\_RENAME, which are used to rename columns that match SQL keywords.

1. **excel\_processor.py**:

Reads and processes the Excel file (DEX-Table\_Ingestion\_Template-V1.xlsx) to extract metadata.

1. **main.py**:

The main script that orchestrates the entire process. It calls functions from other modules to process the Excel file, generate YAML files, and create SQL files.

1. **sql\_generator.py**:

Generates .sql files based on the JSON files and saves them in the srcl\_vw folder.

1. **utils.py**:

Contains utility functions like rename\_column\_if\_keyword, which renames columns if they match SQL keywords.

1. **yaml\_generator.py**:

Generates YAML files (DBtoRedshift.yml and OGGToRedshift.yml) based on the processed data.

1. **requirements.txt**:

Lists all the Python packages required to run the project. Install them using pip install -r requirements.txt.

1. **secret\_name.json**:

Contains a list of secret names for source databases. Used to populate the source\_secret\_name field in the YAML files.

1. **srcl/**:

Contains the input JSON files (e.g., ebt\_rm\_data.json). These files define the structure of the tables being ingested.

1. **srcl\_vw/**:

This folder is created automatically and contains the generated .sql files.

1. **temp/**:

This folder is created automatically and contains the updated JSON files with the schemaName changed to "temp".

1. **DEX-Table\_Ingestion\_Template-V1.xlsx**:

The input Excel file that contains metadata about the data ingestion process.

# ****7.Troubleshooting****

1. **Missing Excel File**:

Ensure the DEX-Table\_Ingestion\_Template-V1.xlsx file is present in the root folder.

1. **Missing JSON Files**:

Ensure the JSON files are present in the srcl folder.

1. **Missing Secret File**:

Ensure the secret\_name.json file is present in the root folder.

1. **Dependencies Not Installed**:

Run pip install -r requirements.txt to install the required packages.

# ****8.Future Enhancements****

1. **Support for Additional Constructs**:

Currently, the project supports only two constructs: DBtoRedshift and OGGtoRedshift.

We plan to extend support to **all constructs available in DEX** to make the tool more versatile.

1. **File Ingestion**:

We have plans to add support for **file ingestion** , enabling the tool to handle file-based data sources.

1. **Enhanced Error Handling**:

Improve error handling and logging to make the tool more robust and user-friendly.