[Rooms to go]

**Prepared by:** Srikanth Kondapalli  
**Role:** Data Integration Developer  
**Date:** 03/21/2025

**Optimized ETL Pipeline for Data Processing**

**Introduction**

In a fast-growing e-commerce retail business, efficiently managing large-scale sales data and customer orders is critical for data-driven decision-making. The system processes multiple CSV files along with detailed JSON data from MongoDB, but faces challenges in scalability, data integrity, and analytics performance.

To overcome these issues, I have designed an optimized ETL (Extract, Transform, Load) pipeline that automates data ingestion, transformation, and storage in Amazon Redshift. This solution includes robust logging, effective error handling, and real-time monitoring via Slack notifications, thereby enhancing overall system reliability and performance.

B**usiness Challenges & Requirements**

**Challenges:**

* CSV sources with redundant and inconsistent data.
* Duplicate records affecting data accuracy.
* Missing values in price fields requiring dynamic imputation.
* Unstructured categories needing standardization.
* MongoDB stores order data in deeply nested JSON format.
* Customer, order, and product details need to be extracted and structured.
* Redshift does not support JSON natively in all clusters.
* Need to extract and query order details from JSON stored in Redshift.

**Requirements:**

* CSV files from a directory.
* Clean data: remove duplicates, handle missing values, format dates.
* Standardize categories and classify products into Expensive or Affordable.
* Extract customer details (Name, Email).
* Extract order details (Order ID, Date, Total price).
* Extract each item in the order (Product name, Quantity, Price).
* Convert JSON to a structured CSV or SQL format.
* Load transformed data into Amazon Redshift.

**ETL Pipeline overview**

The ETL pipeline is designed using Python to efficiently extract, transform, and load sales and order data into Amazon Redshift. It automates data ingestion from CSV files and nested JSON from MongoDB, performs data cleansing, standardization, and structuring, and loads the processed data into Redshift for seamless querying and analytics.

**Technologies Used:**

* **Python** (for ETL processing)
* **Pandas** (for data transformation)
* **PyMongo** (for MongoDB extraction)
* **SQLAlchemy** (for Redshift integration)
* **Slack API** (for real-time notifications)
* **Notebooks (Google collab, Jupyter notebook)**
* **MongoDB Compass (**Data Flattening)

**Solution Implementation**

**Scenario 1: Large-Scale Sales Data Processing**

A screenshot of a computer

Description automatically generated**Fig 1**

* The first step involves extracting sales data from CSV files using Google Colab and Python.
* This involves importing the Pandas library, which plays a key role in data manipulation and analysis.
* Convert raw Excel data into CSV format and perform the necessary transformations.

A screenshot of a computer

Description automatically generated **Fig 2**

A screenshot of a computer

Description automatically generated

**Fig 3**

* Detect missing data and determine data types.
* Identify and eliminate duplicate values

A screenshot of a computer

Description automatically generated

**Fig 4**

* Filter data with first word in category column and categorize them based on price
* Categorizes products into "Expensive" or "Affordable" based on their price range

A screenshot of a computer

Description automatically generated

**Fig 5**

* To load clean data into Redshift, first we need to create a workspace and namespace.
* I assigned IAM policies to the S3 bucket, granting full access to load CSV files.
* Using the COPY command, the data from the S3 bucket was successfully loaded into Redshift for querying

A screenshot of a computer program

Description automatically generated

**Fig 6**

* Success message will be sent to slack.
* By using Google colab and slack webhooks a success notification will be sent to slack Data Integration team.

A screenshot of a computer

Description automatically generated

**Fig 7**

**Scenario 2:**

To develop an efficient data pipeline in Python, I utilized Jupyter Notebook alongside MongoDB Compass. By linking MongoDB with Jupyter Notebook, I was able to flatten and transform the nested JSON data according to the requirements.

A screenshot of a computer

Description automatically generated

**Fig 8**

* Extract customer details, including name and email.
* Retrieve order details such as order ID, date, and total price.
* Extract individual items from each order, including product name, quantity, and price.
* The transformed data in a structured CSV format is stored successfully.

A screenshot of a computer program

Description automatically generated

**Fig 9**

**Scenario 3:**

* Amazon Redshift lacks native support for the JSON data type. Instead, JSON data can be stored as VARCHAR or SUPER (available in Redshift Spectrum or RA3 instances).
* While designing tables for order data using the JSON function, I encountered this constraint. To resolve it, I opted to store JSON data using VARCHAR(MAX).
* The desired output was successfully obtained by implementing this approach.

A screenshot of a computer

Description automatically generated

**Fig 10**

A screenshot of a computer

Description automatically generated

**Fig 11**

**Automation**

1. **AWS Step Functions + Lambda + S3 + Redshift (AWS-Based Solution)**

How It Works:

* Use AWS Step Functions to orchestrate the ETL pipeline.
* AWS Lambda functions trigger different ETL steps, such as:
  + Extracting CSV files from S3
  + Transforming data using Pandas
  + Loading into Redshift using the COPY command
* Set up an S3 Event Trigger to automatically start the pipeline when a new file is uploaded.
* Integrate with CloudWatch for monitoring and error handling.

**Automation Flow:**

S3 Upload → 2. Lambda Trigger → 3. Transform Data (Lambda/Glue Job) → 4. Load into Redshift → 5. Send Slack Notification via SNS

1. **Snowflake + dbt + Air byte (Cloud-Native ETL)**

How It Works:

* Use Airbyte to extract data from CSVs and MongoDB into Snowflake.
* dbt (Data Build Tool) for transformation (cleaning, deduplication, and structuring).
* Schedule ETL workflows in Snowflake Tasks for automation.
* Monitor pipeline execution via Snowflake Query History or external monitoring tools.

**Automation Flow:**

1. Airbyte Sync → Extracts data into Snowflake.
2. dbt Transformations → Cleans, flattens, and formats data.
3. Snowflake Task Scheduler → Runs SQL scripts for automated ETL.
4. Slack/Webhook Notification → Notifies upon success/failure.

**Conclusion**

In this project, I have successfully designed and implemented an optimized ETL pipeline to handle large-scale sales and order data efficiently. By leveraging Python, Pandas, PyMongo, and SQLAlchemy, the pipeline automated data extraction, transformation, and loading into Amazon Redshift while ensuring data integrity, scalability, and improved analytics performance.

Key achievements include:

* **Data Cleaning & Transformation**: Standardized categories, removed duplicates, and handled missing values dynamically.
* **Efficient JSON Processing**: Flattened nested MongoDB data and converted it into a structured format suitable for Redshift.
* **Optimized Data Storage**: Overcame Redshift's JSON limitations by using VARCHAR(MAX), ensuring compatibility and query efficiency.
* **Seamless Integration**: Automated data ingestion from multiple sources, including CSV and MongoDB, while utilizing Slack notifications for real-time monitoring.

By implementing this ETL pipeline, the e-commerce business can now handle large-scale sales data more efficiently, enabling faster decision-making and more accurate insights. This solution provides a solid foundation for future scalability, ensuring that data processing remains reliable, and performance driven as the business continues to grow