***Data Pipeline Technical Document***

**Author**:

Krissanapong Palakham

**Date of Creation:**

19-June-2024

**Only created for Sift Analytics Group (Thailand) Co. Ltd**

Table of Contents

[1. Document Overview 2](#_Toc169731881)

[a. Objective 2](#_Toc169731882)

[b. Purpose Solution 2](#_Toc169731883)

[2. Technical Overview Description 3](#_Toc169731884)

[a. Data Source Investigation 3](#_Toc169731885)

[b. Data Ingestion 3](#_Toc169731886)

[c. ETL Process 3](#_Toc169731887)

[Boilerplate\_extract.ipynb 3](#_Toc169731888)

[Boilerplate\_transform.ipynb 4](#_Toc169731889)

[Main.ipynb 7](#_Toc169731890)

[d. Data Warehouse Loading 8](#_Toc169731891)

[3. Use case situation 8](#_Toc169731892)

[4. Appendix 9](#_Toc169731893)

# Document Overview

## Objective

This document was created for submission to Sift Analytics Group (Thailand) Co. Ltd regarding the job application, as an interview for a data engineer role. The document represented the author's knowledge, capability, and technical skills.

This document represents the data pipeline creation content that implements relevant tools and knowledge about data engineering. In addition, the process and pipeline are created regarding the company’s requirements, which can be viewed on the author’s Git repository.

## Purpose Solution

The solution is to develop the data pipeline regarded to three main areas including data extraction, data transformation, and data loading.

The data pipeline framework below will be more clearly explained and understandable about the methodology of data cleansing and how the project will succeed.

A screenshot of a computer

Description automatically generated

Figure 1.1 Data Pipeline Framework

Figure 1.1 represents the data pipeline framework that the overall project was developed within the Git repository. The data source is a database file that has to be extracted into a Comma-Separated Value (CSV) file. Then, developing using Python language to clean and transform the data regarding the requirements. Finally, loading transformed data into a new database in order to search for more insights. This project has a version control using GitHub.

# Technical Overview Description

## Data Source Investigation

The data source is represented as a database file including 8 tables. After we investigated the data source, we extracted all tables into comma-separated values in order to proceed to the next stage. The list of tables is described below:

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Table Description | Number of Columns | Primary Key |
| Categories | Represented a category of product | 3 | CategoryID |
| Customers | Represented a list of customers including personal data | 8 | CustomerID |
| Employees | Represented a list of employees including internal data | 6 | EmployeeID |
| Orderdetails | Represented a relationship between orders and products, as well as the quantity of products sold | 4 | OrderDetailID |
| Orders | Represented an order which was performed by customers. | 5 | OrderID |
| Products | Represented the product information | 6 | ProductID |
| Shippers | Represented the shipper information | 3 | ShipperID |
| Suppliers | Represented the supplier information | 8 | SupplierID |

Table 2.1 Data source description table

## Data Ingestion

During the data ingestion process, the data source was extracted from the database file into the comma-separated value by utilizing Tableplus. After importing the database file into Tableplus, I selected all tables that were relevant in order to export as the comma-separated value for proceeding data source to the next stage.

The data source has been extracted into the folder named “quiz-de” as the same repository as other processes.

## ETL Process

The ETL Process was separated into three Python files which are performed collaboratively with Jupyter Notebook including main, boilerplate\_extract, and boilerplate\_transform. The description of the working process for each Python file will be discussed below:

### Boilerplate\_extract.ipynb

The boilerplate\_extract file was developed to integrate the function of extracting data from the GitHub repository on a local computer.

# Import required modules

import pandas as pd

import os

This cell of code represents importing the required modules in order to develop this project.

# Set variable

local\_path = "C:/Users/punza/OneDrive/Documents/"

file\_path = f"{local\_path}/GitHub/SIFT-Analytics-DE-Quiz/Part 2/Part 2.2/quiz-de"

This cell of code represents variables to keep the file path including local\_path and file\_path. The local\_path variable is used to keep the local computer path where the GitHub repository was cloned. On the other hand, the file\_path is used to keep all paths including local\_path and the GitHub repository path.

def read\_data():

    """

    To read data from local path; return as a dictionary of DataFrame.

    """

    files\_name: list = []

    data\_dict: dict = {}

    files\_name = os.listdir(file\_path)

    for file in files\_name:

        key = file.split(".")[0]

        data\_dict[key] = pd.read\_csv(file\_path+"/"+file)

    return data\_dict

This cell of code represents a function that reads data from the file path. The files\_name is used to keep all file names in the dataset folder. The data\_dict is used to keep all DataFrame that read from the dataset folder. The output has returned as a data\_dict variable.

### Boilerplate\_transform.ipynb

%run boilerplate\_extract.ipynb

This cell is used to call all functions from boilerplate\_extract.ipynb

# Import required modules

import string

import datetime as dt

This cell of code represents importing the required modules in order to develop this project.

def clean\_phone\_number(phone: str):

    """

    To extract the phone number into numeric values

    """

    for char in phone:

        if char not in string.digits:

            phone = phone.replace(char, "")

    return phone

This cell of code represents a function of phone number cleaning. This function extracts the phone number into a numeric value. If there are characters that are not digits, it will be removed. The output has returned as a phone variable which is a string representing whole digits.

def check\_postal\_code(code: str):

    """

    To check if postal code is alphabet, return false; Else return true.

    """

    for char in code:

       if char in string.ascii\_letters:

            return False

    return True

This cell of code represents a function of postal code checking. If there are some characters in the string represented as an alphabet, it will be returned as False. The output has returned as a Boolean value.

def transform\_suppliers(data: pd.DataFrame):

    """

    To transform the suppliers table; return as a DataFrame.

    """

    # Add supplier\_contact column

    data["supplier\_contact"] = data["Phone"].apply(lambda x:0 if pd.isnull(x) else clean\_phone\_number(x))

    # Transform PostalCode column

    data["PostalCode"] = data["PostalCode"].apply(lambda x:x if check\_postal\_code(x) else 0)

    return data

This cell of code represents a function of the supplier's table transformation. The function added a new column named “supplier\_contact” which extracts the digits from the phone number. If the phone number is null, the value will be zero. In addition, this function transforms the postal code column to keep only digits. If the postal code contains an alphabet, then the value will be zero. The output has returned as a DataFrame.

def transform\_shippers(data: pd.DataFrame):

    """

    To transform the shippers table; return as a DataFrame.

    """

    # Add shipper\_contact column

    data["shipper\_contact"] = data["Phone"].apply(lambda x:0 if pd.isnull(x) else clean\_phone\_number(x))

    return data

This cell of code represents a function of the shipper’s table transformation. The function added a new column named “shipper\_contact” which extracts the digits from the phone number. If the phone number is null, the value will be zero. The output has returned as a DataFrame.

def create\_agg\_table(data\_dict: dict):

    # Change data type of the data

    data\_dict["products"]["ProductID"] = data\_dict["products"]["ProductID"].astype(int)

    data\_dict["products"]["Price"] = data\_dict["products"]["Price"].astype(float)

    data\_dict["orders"]["OrderID"] = data\_dict["orders"]["OrderID"].astype(float)

    # Merge relevant table

    df = pd.merge(data\_dict["products"], data\_dict["orderdetails"], on="ProductID", how="inner")

    df = pd.merge(df, data\_dict["orders"], on="OrderID", how="inner")

    # Extract Year and Month from OrderDate

    df["Year-Month"] = pd.to\_datetime(df["OrderDate"])

    df["Year-Month"] = df["Year-Month"].dt.strftime("%Y-%m")

    # Calculate the sales\_amount

    df["amount"] = df["Price"] \* df["Quantity"]

    #

    df = df.groupby(["Year-Month", "ProductID", "ProductName"]).sum("amount")

    # Compute the percentage\_change

    df["percentage\_change"] = df["amount"].pct\_change() \* 100

    return df

This cell of code represents a function of aggregate table creation. The function is to merge products, orders, and orderdetails tables in order to perform the sales performance for each month as well as the percentage change between monthly. In addition, to create a new table, it is required to transform some data regarding the data type, and data format. The output has returned as a DataFrame.

### Main.ipynb

%run boilerplate\_transform.ipynb

This cell is used to call all functions from boilerplate\_transform.ipynb

data\_dict = read\_data()

This cell of code represents data ingestion by calling read\_data function.

# Clean Suppliers Table

data\_dict["suppliers"] = transform\_suppliers(data\_dict["suppliers"])

display(data\_dict["suppliers"])

# Clean Shippers Table

data\_dict["shippers"] = transform\_shippers(data\_dict["shippers"])

display(data\_dict["shippers"])

# Create aggregate table "product\_sales\_amount\_by\_month"

data\_dict["product\_sales\_amount\_by\_month"] = create\_agg\_table(data\_dict)

display(data\_dict["product\_sales\_amount\_by\_month"])

All of these codes are separated in each cell which represents the data transformation by calling transform\_suppliers, transform\_shippers, and create\_agg\_table functions

file\_path = f"{local\_path}/GitHub/SIFT-Analytics-DE-Quiz/Part 2/Part 2.2/transformed\_data"

This cell of code represents the file\_path variable collecting the path string that we used to keep the transformed data

for key, data in data\_dict.items():

    data.to\_csv(f"{file\_path}/{key}.csv", index = False)

This cell of code represents the methodology in order to save the data into the comma-separated value file within the file path.

## Data Warehouse Loading

During the data warehouse loading process, the transformed data is kept in a folder named “transformed\_data” including 9 comma-separated value files. After we inspected the transformed data and were ready to implement finding insights, we used MySQL Workbench to load all transformed data into the relational database within the local server database.

# Use case situation

From the requirement explanation, we took the use case scenario for our example of implementing the data.

**Situation:**

The company intends to return money to customers over 50 years of age who have made a purchase in 1996 – 1997. Return rates is 50% of purchase amount.

**Solution:**

To perform this situation, we investigate which tables are relevant to perform the SQL script. The first table is a customer that used to find the customer information who are over 50 years old. Then, merge the table between customers and orders table in order to retrieve only customers who making the purchase in 1996-1997. Finally, merge the table between customers, orders, orderdetails, and product\_sales\_amount\_by\_month tables in order to find 50% of the sales amount for each customer.

# Appendix

* 1. Workspace Link (GitHub repository) – <https://github.com/Punny2001/SIFT-Analytics-DE-Quiz>