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

a. 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.

b. 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.

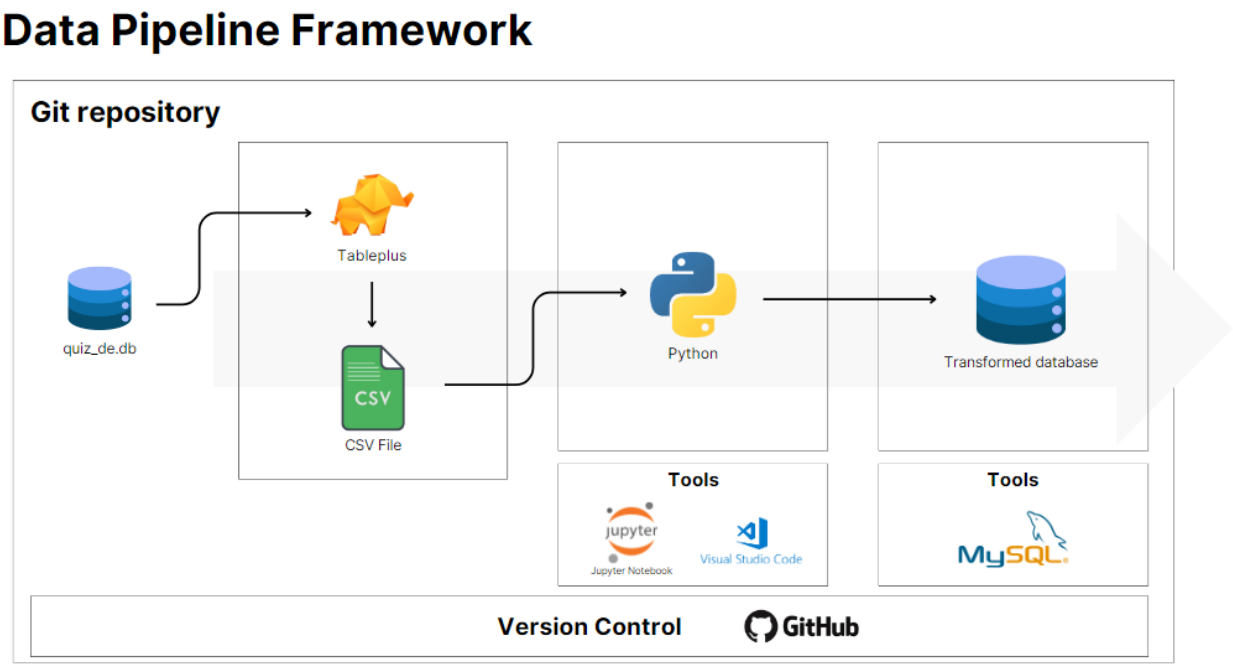

Figure 1.1 Data Pipeline Framework

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.

2. Technical Overview Description

a. 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

b. 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.

c. 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.

d. 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.

3. 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.

4. Appendix

a. Workspace Link (GitHub repository) – https://github.com/Punny2001/SIFT-Analytics-DE-Quiz