**Title**: Technical Solution Document for Building Ultimate Parent Loan ID in Loan Hierarchy

**Author:** Rohan Tandel

Table of Contents

[Overview 3](#_Toc145882895)

[Objective 3](#_Toc145882896)

[Solution 4](#_Toc145882897)

[1. SQL 4](#_Toc145882898)

[2. Python 8](#_Toc145882899)

[RESOURCES 11](#_Toc145882900)

[Problem 11](#_Toc145882901)

[SQL Script: 12](#_Toc145882902)

[PYTHON Script: 14](#_Toc145882903)

[CSV: 16](#_Toc145882904)

[Loan\_data.csv 16](#_Toc145882905)

[Loan\_data\_more\_loan\_id.csv 16](#_Toc145882906)

# **Overview**

Our project involves processing source data provided in the format of **loan\_id** and **parent\_loan\_id**. The primary objective is to establish a clear hierarchy within this data and determine the "Ultimate\_parent\_loan\_id" for each entry. This task will be accomplished using either SQL or Python, depending on our chosen implementation approach.

# **Objective**

In the realm of data processing, understanding relationships and hierarchies within datasets is often pivotal for informed decision-making and analysis. Our project addresses this imperative need by developing a solution to process source data provided in the **loan\_id** and **parent\_ loan\_id** format.

Our primary objective is to establish a hierarchical structure that connects each **loan\_id** to its **ultimate\_parent\_loan\_id**, thereby providing clarity and context to the relationships within the dataset. In the provided data, relationships between **loan\_id** and **parent\_loan\_id** are crucial to building the hierarchy. Additionally, we acknowledge that a **parent\_loan\_id** can exist as both a **loan\_id** and a **parent\_loan\_id**, with a null **parent\_loan\_id** indicating the highest level of hierarchy.

# **Solution**

We will implement this solution using either SQL or Python. The chosen implementation method will be tailored to efficiently handle the data transformation requirements.

## **SQL**

We assume that the source data is already loaded into a stage layer table in the database. We will use **Recursive CTE** (common table expression)to build the hierarchical structure in the format loan\_id, parent\_loan\_id and ultimate\_parent\_loan\_id.

The script below first creates temporary table to replicate stage layer table in the Enterprise Data Warehouse. We use this temporary table to build the Recursive CTE. The script was executed in Google BigQuery environment.

CREATE OR REPLACE TEMPORARY TABLE tt\_loan\_rel

(

loan\_id INT64,

parent\_loan\_id INT64

);

INSERT INTO tt\_loan\_rel

VALUES

(100,600),

(200,NULL),

(600,200),

(300,NULL)

;

WITH RECURSIVE parent\_child\_rel

AS

(

SELECT loan\_id,

parent\_loan\_id,

loan\_id AS ultimate\_parent\_loan\_id

FROM tt\_loan\_rel

WHERE parent\_loan\_id IS NULL

UNION ALL

SELECT t.loan\_id,

t.parent\_loan\_id,

ultimate\_parent\_loan\_id

FROM parent\_child\_rel AS p

JOIN tt\_loan\_rel AS t

ON t.parent\_loan\_id = p.loan\_id

)

SELECT \*

FROM parent\_child\_rel p

ORDER BY 3,1;

Output:

A screenshot of a computer

Description automatically generated

We will now build the temporary table with additional loan\_id’s to build upto 8 levels of hierarchy.

CREATE OR REPLACE TEMPORARY TABLE tt\_loan\_rel

(

loan\_id INT64,

parent\_loan\_id INT64

);

INSERT INTO tt\_loan\_rel

VALUES

(100,600),

(200,NULL),

(600,200),

(300,NULL),

(400,600),

(500,400),

(700,500),

(800,300),

(900,800),

(1000,900),

(1001,900),

(1002,1001),

(1003,1002),

(1004,1003),

(1005,1004);

WITH RECURSIVE parent\_child\_rel

AS

(

SELECT loan\_id,

parent\_loan\_id,

loan\_id AS ultimate\_parent\_loan\_id

FROM tt\_loan\_rel

WHERE parent\_loan\_id IS NULL

UNION ALL

SELECT t.loan\_id,

t.parent\_loan\_id,

ultimate\_parent\_loan\_id

FROM parent\_child\_rel AS p

JOIN tt\_loan\_rel AS t

ON t.parent\_loan\_id = p.loan\_id

)

SELECT \*

FROM parent\_child\_rel p

ORDER BY 3,1;

Output:

A screenshot of a computer

Description automatically generated

## **Python**

We assume the data is available in CSV file with format “Loan\_id,Parent\_Loan\_id”. The file contains headers, and all the data is in Integer data type.

We will do recursive call to function/methodto build the hierarchical structure in the format loan\_id, parent\_loan\_id and ultimate\_parent\_loan\_id. We use pandas dataframe to load the data and to build the final dataset which can be loaded into the Enterprise Data Warehouse.

1. The script below first reads data from CSV file into pandas dataframe **src\_df.** The file path is passed in as argument when running the script.
2. New dataframe **trgt\_df** is created using **src\_df** and adding new column **Ultimate\_Parent\_Loan\_ID** with default value as None.
3. Function find\_ult\_prnt\_loan\_id() is used to perform recursion to build the hierarchy. We assign the values to **Ultimate\_Parent\_Loan\_ID** in this step.
4. Finally we call the function load\_into\_database() to load the data into database. In the script the function is just a place holder. We can add code here to load data into our target database.

import pandas as pd

import argparse

import os

# Recursive function to find ultimate parent loan id

def find\_ult\_prnt\_loan\_id(pl\_id):

rec\_df = trgt\_df[trgt\_df['Loan\_id'] == pl\_id]

if not rec\_df.empty:

ln\_id = rec\_df['Parent\_Loan\_id'].values[0]

if not pd.isna(ln\_id):

find\_ult\_prnt\_loan\_id(ln\_id)

else:

trgt\_df.loc[counter,'Ultimate\_Parent\_Loan\_ID'] = rec\_df['Loan\_id'].values[0]

# Function to load data into the database

def load\_into\_database(target\_dataframe):

pass

# we can add code here to load data into database

# Create parser to read arguments

parser = argparse.ArgumentParser()

parser.add\_argument('--file', help='CSV file path')

args = parser.parse\_args()

options = vars(args)

file\_path = options['file']

# Check if provided file path is valid

if not os.path.isfile(file\_path):

print('{} is not a valid file.'.format(file\_path))

quit(1)

# Create the pandas DataFrame using csv file

src\_df = pd.read\_csv(file\_path, dtype={'Loan\_id':'Int64','Parent\_Loan\_id':'Int64'})

# Create target dataframe using source dataframe and adding new column

trgt\_df = src\_df.assign(Ultimate\_Parent\_Loan\_ID=None)

# Loop through all the rows in the dataframe

for i in range(0,trgt\_df.shape[0]):

prnt\_ln\_id = trgt\_df['Parent\_Loan\_id'][i]

counter = i

# if parent loan id is present find the ultimate parent loan id

# else assign loan id as ultimate customer parent loan id

if not pd.isna(prnt\_ln\_id):

find\_ult\_prnt\_loan\_id(prnt\_ln\_id)

else:

trgt\_df.loc[counter,'Ultimate\_Parent\_Loan\_ID'] = trgt\_df['Loan\_id'][counter]

# Print the source

print('\nSource:')

print(src\_df.to\_string(index=False).replace('<NA>','null'))

# Print final output

print('\nTarget:')

print(trgt\_df.to\_string(index=False).replace('<NA>','null'))

# Load data into database

load\_into\_database(trgt\_df)

# Check if provided file path is valid

if not os.path.isfile(file\_path):

print('{} is not a valid file.'.format(file\_path))

quit(1)

# Create the pandas DataFrame using csv file

src\_df = pd.read\_csv(file\_path, dtype={'Loan\_id':'Int64','Parent\_Loan\_id':'Int64'})

# Create target dataframe using source dataframe and adding new column

trgt\_df = src\_df.assign(Ultimate\_Parent\_Loan\_ID=None)

# Loop through all the rows in the dataframe

for i in range(0,trgt\_df.shape[0]):

prnt\_ln\_id = trgt\_df['Parent\_Loan\_id'][i]

counter = i

# if parent loan id is present find the ultimate parent loan id

# else assign loan id as ultimate customer parent loan id

if not pd.isna(prnt\_ln\_id):

find\_ult\_prnt\_loan\_id(prnt\_ln\_id)

else:

trgt\_df.loc[counter,'Ultimate\_Parent\_Loan\_ID'] = trgt\_df['Loan\_id'][counter]

# Print the source

print('\nSource:')

print(src\_df.to\_string(index=False).replace('<NA>','null'))

# Print final output

print('\nTarget:')

print(trgt\_df.to\_string(index=False).replace('<NA>','null'))

# Load data into database

load\_into\_database(trgt\_df)

**Output:A computer screen with white text

Description automatically generated**

Running the same script with source file containing upto 8 levels of hierarchy.

**A screenshot of a computer

Description automatically generated**

**RESOURCES:**

**Problem:**

**Source:**

**Loan\_id Parent\_Loan\_id**

**100 600**

**200 null**

**600 200**

**300 null**

**Target:**

**Loan\_id Parent\_Loan\_id Ultimate\_Parent\_Loan\_ID**

**100 600 200**

**200 null 200**

**600 200 200**

**300 null 300**

## **SQL Script:**

**/\*#######################################################################################################################**

**# Developer : Rohan Tandel**

**# Description : Find Ultimate Parent Loan ID for given Loan\_id and Parent\_Loan\_id data built in temporary table**

**#######################################################################################################################\*/**

**CREATE OR REPLACE TEMPORARY TABLE tt\_loan\_rel**

**(**

**loan\_id INT64,**

**parent\_loan\_id INT64**

**);**

**INSERT INTO tt\_loan\_rel**

**VALUES**

**(100,600),**

**(200,NULL),**

**(600,200),**

**(300,NULL)**

**/\***

**We can add this test the recursion for more levels**

**,**

**(400,600),**

**(500,400),**

**(700,500),**

**(800,300),**

**(900,800),**

**(1000,900),**

**(1001,900),**

**(1002,1001),**

**(1003,1002),**

**(1004,1003),**

**(1005,1004)\*/**

**;**

**WITH RECURSIVE parent\_child\_rel**

**AS**

**(**

**SELECT loan\_id,**

**parent\_loan\_id,**

**loan\_id AS ultimate\_parent\_loan\_id**

**FROM tt\_loan\_rel**

**WHERE parent\_loan\_id IS NULL**

**UNION ALL**

**SELECT t.loan\_id,**

**t.parent\_loan\_id,**

**ultimate\_parent\_loan\_id**

**FROM parent\_child\_rel AS p**

**JOIN tt\_loan\_rel AS t**

**ON t.parent\_loan\_id = p.loan\_id**

**)**

**SELECT \***

**FROM parent\_child\_rel p**

**ORDER BY 3,1;**

## **PYTHON Script:**

**parent\_child\_load\_csv.py**

**#######################################################################################################################**

**# Developer : Rohan Tandel**

**# Description : Find Ultimate Parent Loan ID for given Loan\_id and Parent\_Loan\_id data in file**

**# Execution : python3 parent\_child\_load\_csv.py --file 'file\_path'**

**# : ex - python3 parent\_child\_load\_csv.py --file '/Users/rohantandel/Downloads/Zions/python/Loan\_data.csv'**

**#######################################################################################################################**

**import pandas as pd**

**import argparse**

**import os**

**# Recursive function to find ultimate parent loan id**

**def find\_ult\_prnt\_loan\_id(pl\_id):**

**rec\_df = trgt\_df[trgt\_df['Loan\_id'] == pl\_id]**

**if not rec\_df.empty:**

**ln\_id = rec\_df['Parent\_Loan\_id'].values[0]**

**if not pd.isna(ln\_id):**

**find\_ult\_prnt\_loan\_id(ln\_id)**

**else:**

**trgt\_df.loc[counter,'Ultimate\_Parent\_Loan\_ID'] = rec\_df['Loan\_id'].values[0]**

**# Function to load data into the database**

**def load\_into\_database(target\_dataframe):**

**pass**

**# we can add code here to load data into database**

**# Create parser to read arguments**

**parser = argparse.ArgumentParser()**

**parser.add\_argument('--file', help='CSV file path')**

**args = parser.parse\_args()**

**options = vars(args)**

**file\_path = options['file']**

**# Check if provided file path is valid**

**if not os.path.isfile(file\_path):**

**print('{} is not a valid file.'.format(file\_path))**

**quit(1)**

**# Create the pandas DataFrame using csv file**

**src\_df = pd.read\_csv(file\_path, dtype={'Loan\_id':'Int64','Parent\_Loan\_id':'Int64'})**

**# Create target dataframe using source dataframe and adding new column**

**trgt\_df = src\_df.assign(Ultimate\_Parent\_Loan\_ID=None)**

**# Loop through all the rows in the dataframe**

**for i in range(0,trgt\_df.shape[0]):**

**prnt\_ln\_id = trgt\_df['Parent\_Loan\_id'][i]**

**counter = i**

**# if parent loan id is present find the ultimate parent loan id**

**# else assign loan id as ultimate customer parent loan id**

**if not pd.isna(prnt\_ln\_id):**

**find\_ult\_prnt\_loan\_id(prnt\_ln\_id)**

**else:**

**trgt\_df.loc[counter,'Ultimate\_Parent\_Loan\_ID'] = trgt\_df['Loan\_id'][counter]**

**# Print the source**

**print('\nSource:')**

**print(src\_df.to\_string(index=False).replace('<NA>','null'))**

**# Print final output**

**print('\nTarget:')**

**print(trgt\_df.to\_string(index=False).replace('<NA>','null'))**

**# Load data into database**

**load\_into\_database(trgt\_df)**

## **CSV:**

Loan\_data.csvLoan\_id,Parent\_Loan\_id

100,600

200,

600, 200

300,

### Loan\_data\_more\_loan\_id.csv

Loan\_id,Parent\_Loan\_id

100,600

200,

600, 200

300,

400,600

500,400

700,500

800,300

900,800

1000,900

1001,900

1002,1001

1003,1002

1004,1003

1005,1004