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

Author: Rohan Tandel

Table of Contents

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.

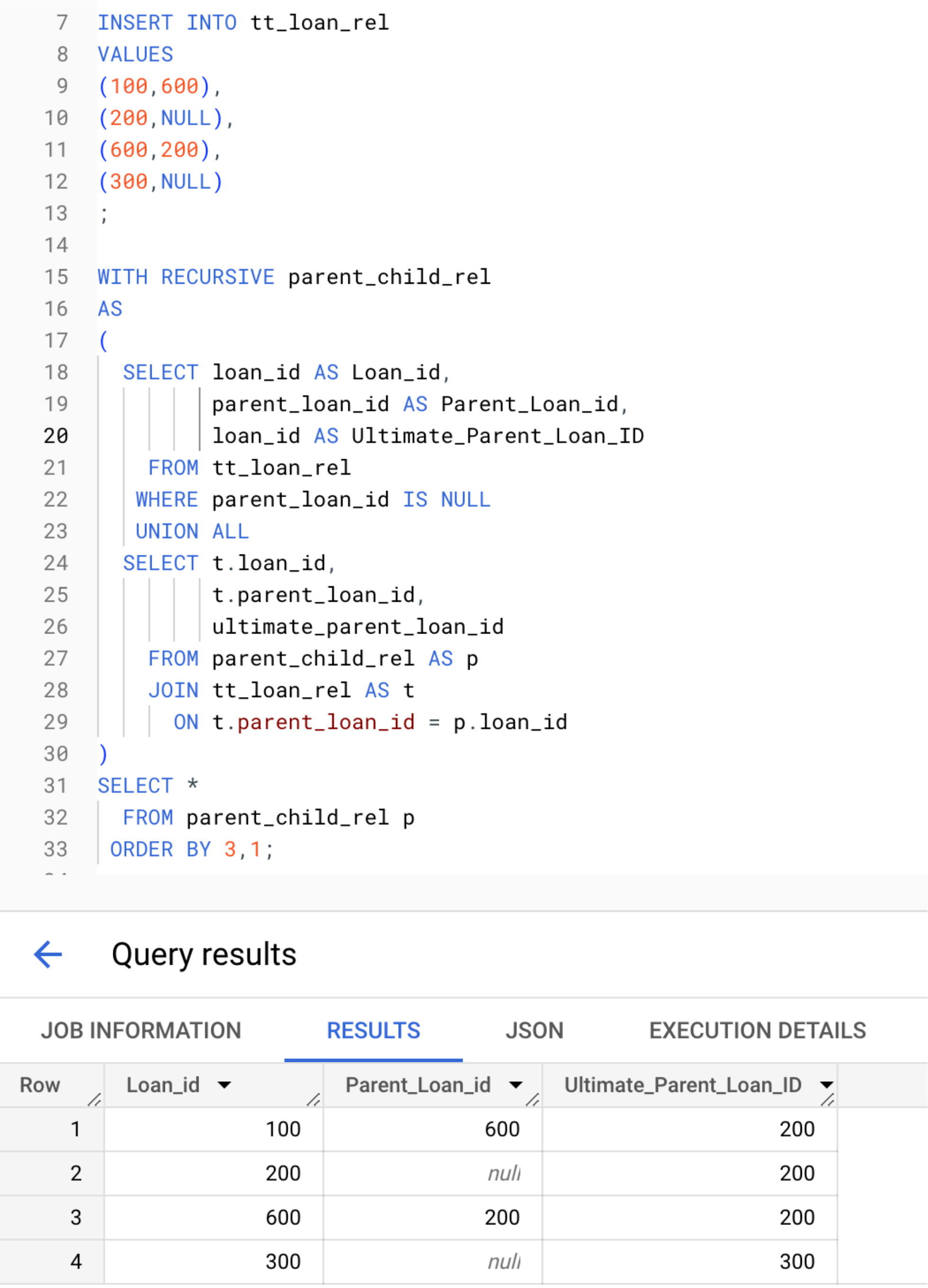
1. SQL

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:



```
INSERT INTO tt_loan_rel
VALUES
(100,600),
(200,NULL),
(600,200),
(300,NULL)
;

WITH RECURSIVE parent_child_rel
AS
(
  SELECT loan_id AS Loan_id,
         parent_loan_id AS 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;
```

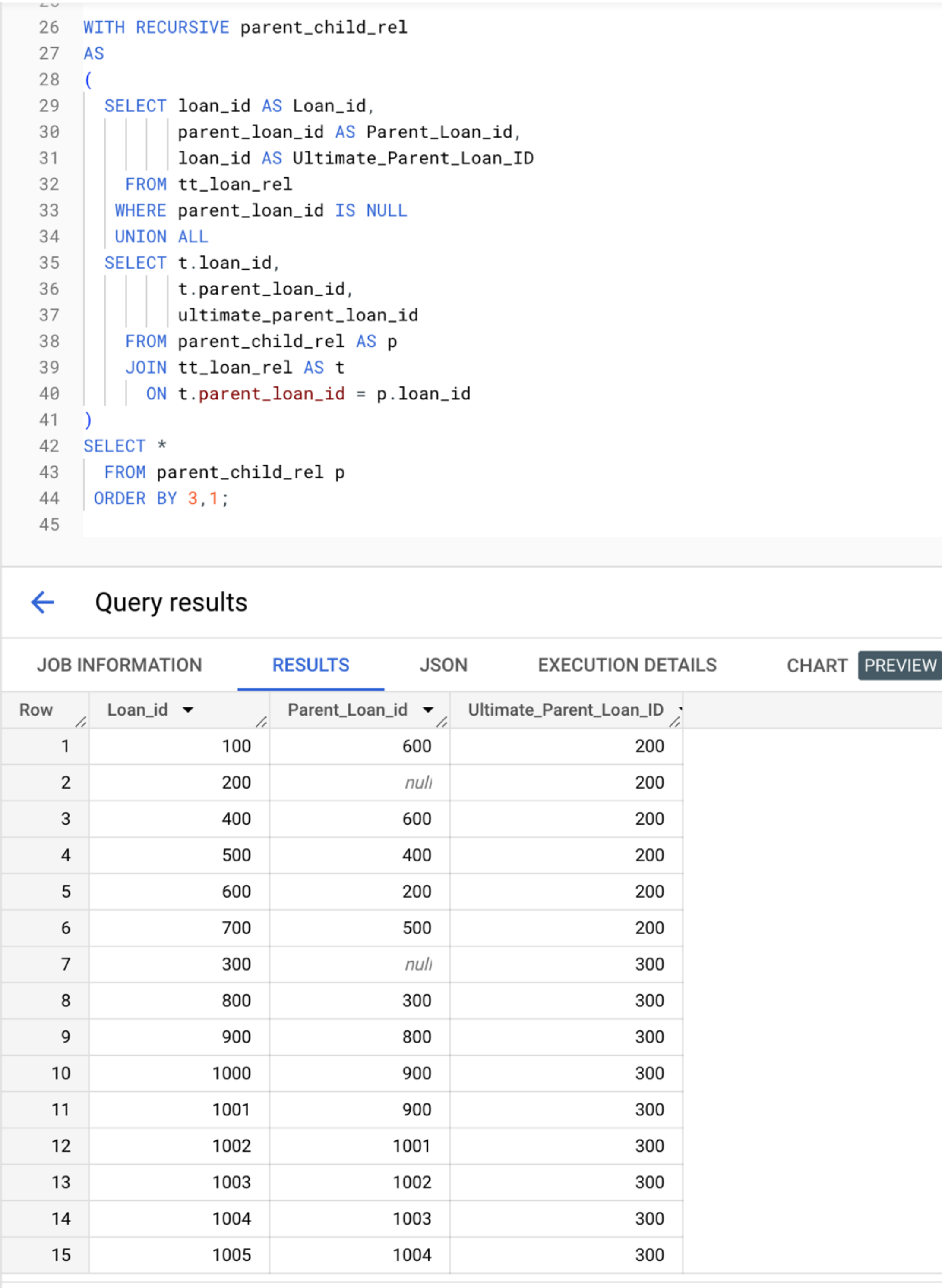
Query results

Row	Loan_id	Parent_Loan_id	Ultimate_Parent_Loan_ID
1	100	600	200
2	200	null	200
3	600	200	200
4	300	null	300

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:



```
WITH RECURSIVE parent_child_rel
AS
(
  SELECT loan_id AS Loan_id,
         parent_loan_id AS 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;
```

Row	Loan_id	Parent_Loan_id	Ultimate_Parent_Loan_ID
1	100	600	200
2	200	*null*	200
3	400	600	200
4	500	400	200
5	600	200	200
6	700	500	200
7	300	*null*	300
8	800	300	300
9	900	800	300
10	1000	900	300
11	1001	900	300
12	1002	1001	300
13	1003	1002	300
14	1004	1003	300
15	1005	1004	300

2. Python

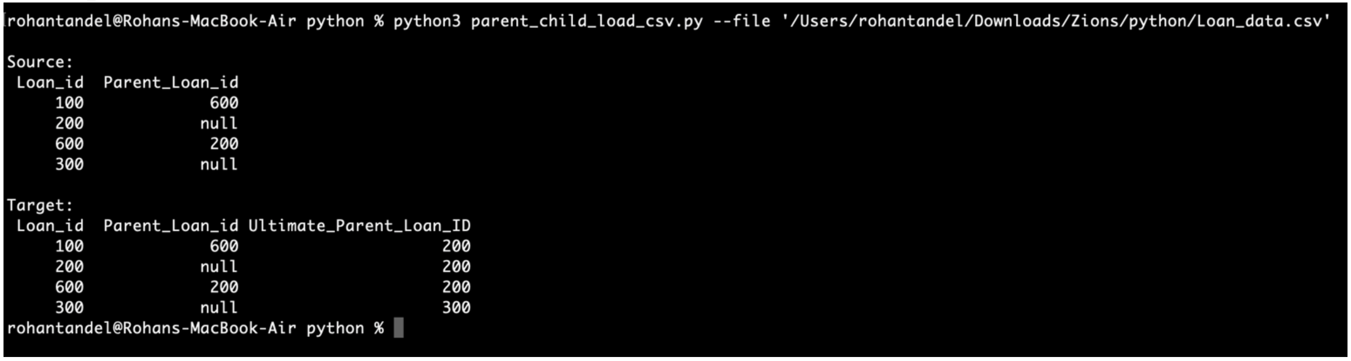
We assume the data is available in CSV file with format “Loan_id,Parent_Loan_id”. The file contains headers, and all the data is in Integer data type.
We will do recursive call to function/method to build the hierarchical structure in the format loan_id, parent_loan_id and ultimate_parent_loan_id. We 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']
```

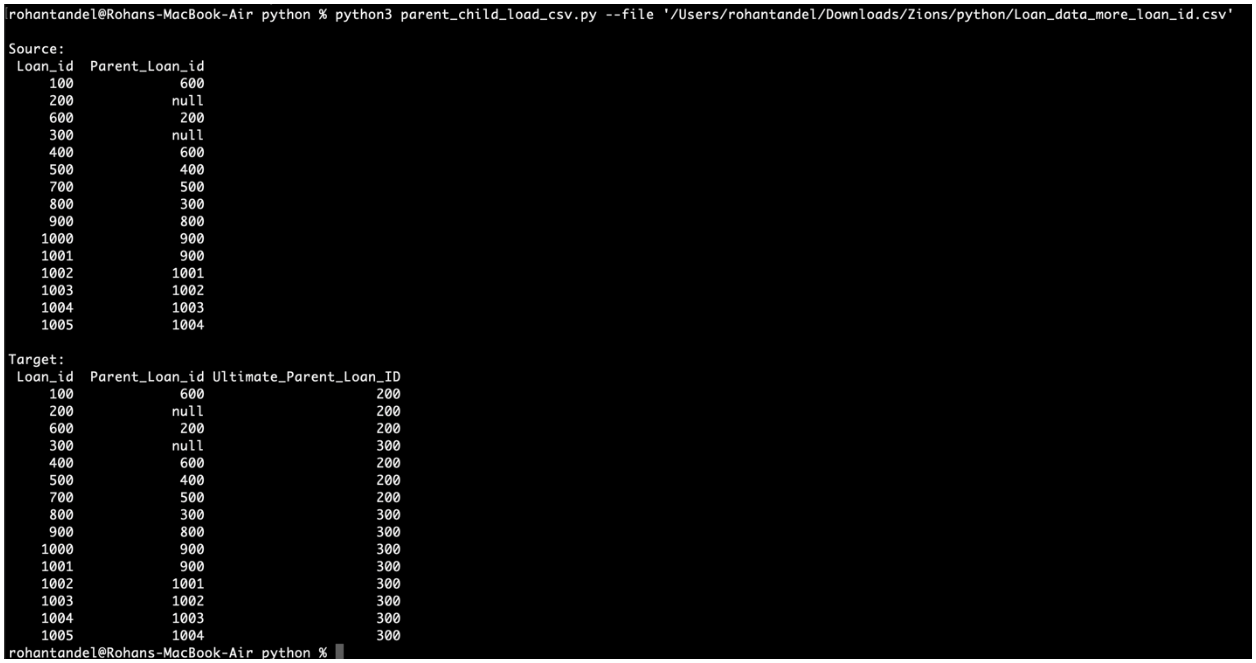
```
# Check if provided file path is valid
if not os.path.isfile(file_path):
  print('{} is not a valid file.'.format(file_path))
  quit(1)

# 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'))
```

Output:



```
rohantandel@Rohans-MacBook-Air python % python3 parent_child_load_csv.py --file '/Users/rohantandel/Downloads/Zions/python/Loan_data.csv'

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
rohantandel@Rohans-MacBook-Air python % 
```

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



```
rohantandel@Rohans-MacBook-Air python % python3 parent_child_load_csv.py --file '/Users/rohantandel/Downloads/Zions/python/Loan_data_more_loan_id.csv'

Source:
 Loan_id  Parent_Loan_id
     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

Target:
 Loan_id  Parent_Loan_id Ultimate_Parent_Loan_ID
     100             600                     200
     200            null                     200
     600             200                     200
     300            null                     300
     400             600                     200
     500             400                     200
     700             500                     200
     800             300                     300
     900             800                     300
    1000             900                     300
    1001             900                     300
    1002            1001                     300
    1003            1002                     300
    1004            1003                     300
    1005            1004                     300
rohantandel@Rohans-MacBook-Air python %
```

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.csv

```
Loan_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
```