github.com/yboryshchak

SalesForce data ETL with Simple SalesForce

SalesForce SOQL

Salesforce offers several customer relationship management (CRM) services, such as Sales Cloud, Service Cloud, Marketing Cloud, Commerce Cloud and Platform. Salesforce Object Query Language (SOQL) query is the equivalent of a SELECT SQL statement.

More details at https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_soql_sosl_intro.htm

Simple SalesForce

Simple Salesforce is a basic Salesforce.com REST API client built for Python 3.6, 3.7 3.8, 3.9, 3.10, and 3.11. The goal is to provide a very low-level interface to the REST Resource and APEX API, returning a dictionary of the API JSON response.

Find more at https://github.com/simple-salesforce/simple-salesforce

You can install Simple Salesforce package either through **PyPI:**

pip install simple_salesforce

or **Anaconda:**

conda install -c conda-forge simple_salesforce

API Security Token

To obtain or reset SalesForce security token, login to SalesForce, navigate to **View Profile/Settings/Reset My Security Token**, and click Reset Security Token. The security token will be sent to your registered email within a few minutes.

Python code from now on!

```
# 0. List all available SalesForce tables.

""" Turn VPN Off !!! """

import os, os.path, simple_salesforce

from simple_salesforce import Salesforce

import numpy as np

import pandas as pd

from pandas import DataFrame

from pathlib import Path

import pyarrow as pa

import pyarrow.parquet as pq


# Useful Functions

def cred_sf():

    """ SalesForce Credential Function """

    SFC  = Salesforce(username='YourUserName', password='YourPassword', security_token= 'YourSecurityToken')

    return SFC

def SaveTable(Table_df, FolderPath, FileName, FileType = 2):
```

```
    """ Saves Table_df as 0-parquet, 1-pickle, 2-csv """
    t_shape = Table_df.shape
    if FileType == 0:
        Table_padf = pa.Table.from_pandas(Table_df)
        Fname = FileName + '.parquet'
        FilePath = os.path.join(FolderPath, Fname)
        pq.write_table(Table_padf, FilePath)
    elif FileType == 1:
        Fname = FileName + '.pickle'
        FilePath = os.path.join(FolderPath, Fname)
        Table_df.to_pickle(FilePath)
    else:
        Fname = FileName + '.csv'
        FilePath = os.path.join(FolderPath, Fname)
        Table_df.to_csv(FilePath, index = False)
    return print("...", t_shape, " saved ", FilePath)
```

 # List Available SF tables

```
sf = cred_sf()
tables_df = pd.DataFrame(sf.query("SELECT QualifiedApiName, Label, IsQueryable, IsCustomSetting  FROM EntityDefinition")['records'])
sfo_list = tables_df['QualifiedApiName'].to_list()
tables_df.drop(columns = ['attributes'] , inplace = True)
tables_df.rename(columns = {'QualifiedApiName': 'TableName', 'Label': 'TableLabel'}, inplace = True)
obj_num = len(sfo_list)
print('... Found ', obj_num, ' Sales Force Tables')
```

table.describe() keys

```
TableDescribeKeys = ['actionOverrides', 'activateable', 'childRelationships', 'compactLayoutable', 'createable', 'custom', 'customSetting', 'deletable',
'deprecatedAndHidden', 'feedEnabled', 'fields', 'hasSubtypes', 'isSubtype', 'keyPrefix', 'label', 'labelPlural', 'layoutable', 'listviewable',
'lookupLayoutable', 'mergeable', 'mruEnabled', 'name', 'namedLayoutInfos', 'networkScopeFieldName', 'queryable', 'recordTypeInfos',
'replicateable', 'retrieveable', 'searchLayoutable', 'searchable', 'sobjectDescribeOption', 'supportedScopes', 'triggerable', 'undeletable',
'updateable', 'urls']
```

fields Keys

```
TableFieldsKeys = ['aggregatable', 'autoNumber', 'byteLength', 'calculated', 'calculatedFormula', 'cascadeDelete', 'caseSensitive',
'compoundFieldName', 'controllerName', 'createable', 'custom', 'defaultValue', 'defaultValueFormula', 'defaultedOnCreate', 'dependentPicklist',
'deprecatedAndHidden', 'digits', 'displayLocationInDecimal', 'encrypted', 'externalId', 'extraTypeInfo', 'filterable', 'filteredLookupInfo', 'groupable',
'highScaleNumber', 'htmlFormatted', 'idLookup', 'inlineHelpText', 'label', 'length', 'mask', 'maskType', 'name', 'nameField', 'namePointing', 'nillable',
'permissionable', 'picklistValues', 'polymorphicForeignKey', 'precision', 'queryByDistance', 'referenceTargetField', 'referenceTo', 'relationshipName',
'relationshipOrder', 'restrictedDelete', 'restrictedPicklist', 'scale', 'searchPrefilterable', 'soapType', 'sortable', 'type', 'unique', 'updateable',
'writeRequiresMasterRead']
```

```
#1. Table Cols Describe
tnum = 0
for table in  sfo_list:
    t_desc = """sf.{}.describe()""".format(table)
    table_desc = eval(t_desc)
    pc_arr = table_desc['fields']
    f_col = pc_arr[0]
    lkeys = list(f_col.keys())
    table_name = [table for x in range(len(pc_arr))]
    colsnum = [int(len(pc_arr)) for x in range(len(pc_arr))]
    pc_df = pd.DataFrame({'Table': table_name, 'Cols_Num': colsnum})
    tcols = pc_df['name'].to_list()
    tlabels = pc_df['label'].to_list()
    print('... ', table, ' Names -> Labels: ')
    m_query = M_SF_RenameCols(tcols, tlabels)
    for ckey in lkeys:
        keycol = [str(x[ckey]) for x in pc_arr]
        pc_df[ckey] = keycol
    if tnum == 0:
        comb_df = pc_df.copy(deep=True)
    else:
        comb_df = pd.concat([comb_df, pc_df], ignore_index=True)
    tnum = tnum + 1
    print(tnum, ' ... out of ', len(sfo_list))

# Print SalesForce Col Types
col_types = set(comb_df['type'].to_list())
print('... SF_ColTypes = ', sorted(list(col_types)))
SF_ColTypes = ['address', 'anyType', 'base64', 'boolean', 'combobox', 'complexvalue', 'currency', 'date', 'datetime', 'double', 'email',
'encryptedstring', 'id', 'int', 'location', 'multipicklist', 'percent', 'phone', 'picklist', 'reference', 'string', 'textarea', 'time', 'url']

# Save Table-Cols
comb_df.sort_values(by = ['Table', 'type', 'name'], inplace = True)
out_path = Path(r"C:\Users\YourUserName\Desktop")
SaveTable(Table_df = comb_df, FolderPath = out_path, FileName = 'SF_Table-Cols', FileType = 2)
comb_df = pd.DataFrame()
```

```
# 2. Compile Parent-Child Relations Map
tnum = 0
for table in sfo_list:
    t_desc = """sf.{}.describe()""".format(table)
    table_desc = eval(t_desc)
    pc_arr = table_desc['childRelationships']
    print('... PricingRequest__c childObject ... ', pc_arr)
    child_list = [x['childSObject'] for x in pc_arr]
    field_list = [x['field'] for x in pc_arr]
    relation_list = [x['relationshipName'] for x in pc_arr]
    cdel_list = [str(x['cascadeDelete']) for x in pc_arr]
    rdel_list = [str(x['restrictedDelete']) for x in pc_arr]
    jrt_list = [str(x['junctionReferenceTo']) for x in pc_arr]
    jrid_list = [str(x['junctionIdListNames']) for x in pc_arr]
    table_name = [table for x in range(len(child_list))]
    pc_df = pd.DataFrame({'ParentTable': table_name, 'ChildTable': child_list, 'ChildColName': field_list, 'RelationName': relation_list,
'cascadeDelete': cdel_list, 'restrictedDelete': rdel_list, 'junctionReferenceTo': jrt_list, 'junctionIdListNames': jrid_list})
    if tnum == 0:
        comb_df = pc_df.copy(deep=True)
    else:
        comb_df = pd.concat([comb_df, pc_df], ignore_index=True)
    tnum = tnum + 1
    print(tnum)
    field_names = [field['name'] for field in table_desc['fields']]


# Save SalesForce Parent-Child
comb_df = comb_df[comb_df['ParentTable'] != comb_df['ChildTable']] # filter out dumb relations
comb_df.sort_values(by = ['ParentTable', 'ChildTable'], inplace = True)
SaveTable(Table_df = comb_df, FolderPath = out_path, FileName = 'SF_Parent-Child', FileType = 2)
comb_df = pd.DataFrame()


# 3. Load of  'queryable' SF tables
load_list = []
for table in load_list:
    t_desc = """sf.{}.describe()""".format(table)
    table_desc = eval(t_desc)
```

```
    field_names = [field['name'] for field in table_desc['fields']]
    t_soql = "SELECT {} FROM".format(','.join(field_names)) + " {}".format(table)
    query_sf = sf.query_all(t_soql)
    table_df = pd.DataFrame(query_sf['records']).drop(columns='attributes')
    SaveTable(Table_df = table_df, FolderPath = out_path, FileName = table, FileType = 2)

# 4. Load of 'non-queryable' SF tables
""" Non-queryable objects can only be queried within merged SOQL statement with a quachild.
Example: non-queryable table ProcessInstance, (must be) parent to StepAndWorkitem"""

SOQL = "SELECT Id, TargetObjectId, (SELECT Id, StepStatus, Comments FROM StepsAndWorkitems), Status, ProcessDefinitionId FROM ProcessInstance"
"""Inner (child) object, StepsWorkitems, called in SOQL above in plural form, which is found in 'labelPlural' of table.describe() function """
query_sf = sf.query_all(SOQL)
ch_df = pd.DataFrame(query_sf['records']).drop(columns='attributes') # pivoted merge table
print('... ch_df cols: ', list(ch_df.columns))
Id_list = ch_df.shape[0]
nrows = len(Id_list)
TOId_list = ch_df['TargetObjectId'].to_list()
SAWI_list = ch_df['StepsAndWorkitems'].to_list()
prdf_list = ch_df['ProcessDefinitionId'].to_list()

# Unpivot Step History from StepsAndWorkitems
id_array = []
toid_array = []
stepstatus_array = []
stepid_array = []
stepstatus_array = []
stepcomments_array = []
step_types = []
procdef_array = []

for rn in range(0, nrows):
    sawi_dict = SAWI_list[rn]
    cid = Id_list[rn]
    toid = TOId_list[rn]
    prdf = prdf_list[rn]
```

```
    sawi_steps = sawi_dict['records']
    for step in sawi_steps:
        step_attr_dict = step['attributes']
        step_type = step_attr_dict['type']
        step_id = step['Id']
        step_status = step['StepStatus']
        step_comments = step['Comments']
        if step_type == 'ProcessInstanceHistory':
            id_array.append(cid)
            toid_array.append(toid)
            procdef_array.append(prdf)
            stepid_array.append(step_id)
            stepstatus_array.append(step_status)
            stepcomments_array.append(step_comments)
        else:
            step_types.append(step_type)

# Compile and Save StepsAndWorkItems Table

SAWI_df = pd.DataFrame({'Id': id_array, 'TargetObjectId': toid_array, 'ProcessDefinitionId': procdef_array, 'StepId': stepid_array, 'StepStatus': stepstatus_array, 'StepComments': stepcomments_array})

SaveTable(Table_df = SAWI_df, FolderPath = out_path, FileName = 'StepsAndWorkitemsHistory', FileType = 2)
```