import argparse  
import logging  
import os  
import sys  
from contextlib import contextmanager  
from datetime import datetime  
import pandas as pd  
from sqlalchemy import text  
  
from da\_common import da\_sp\_wrapper  
from etl.core.base.agent import AgentBase  
from etl.core.da\_log import logger as log  
from etl.repo.fnd\_cfdw import EtlFileRepo  
from etl.repo.pim\_da.dats\_series import DatsSeriesRepo  
from etl.repo.pim\_da.stgp\_dats\_series\_value import StgpDatsSeriesValueRepo  
from etl.repo.pim\_da.ups\_dats\_series\_value import UpsDatsSeriesValueRepo  
  
  
class LoaderAgentRepo:  
 def \_init\_(self):  
 self.\_etl\_file\_repo = None  
 self.\_dats\_series\_repo = None  
 self.\_stgp\_dats\_series\_value\_repo = None  
 self.\_ups\_dats\_series\_value\_repo = None  
  
 @property  
 def dats\_series\_repo\_(self):  
 if self.\_dats\_series\_repo is None:  
 self.\_dats\_series\_repo = DatsSeriesRepo()  
 return self.\_dats\_series\_repo  
  
 @property  
 def etl\_file\_repo\_(self):  
 if self.\_etl\_file\_repo is None:  
 self.\_etl\_file\_repo = EtlFileRepo()  
 return self.\_etl\_file\_repo  
  
 @property  
 def stgp\_dats\_series\_value\_repo\_(self):  
 if self.\_stgp\_dats\_series\_value\_repo is None:  
 self.\_stgp\_dats\_series\_value\_repo = StgpDatsSeriesValueRepo()  
 return self.\_stgp\_dats\_series\_value\_repo  
  
 @property  
 def ups\_dats\_series\_value\_repo\_(self):  
 if self.\_ups\_dats\_series\_value\_repo is None:  
 self.\_ups\_dats\_series\_value\_repo = UpsDatsSeriesValueRepo()  
 return self.\_ups\_dats\_series\_value\_repo  
  
  
class LoaderAgent(AgentBase):  
 def \_init\_(self):  
 super(LoaderAgent, self).\_init\_(app\_code='etl-dats-bbg\_agent\_ld')  
 self.repo = None  
 self.ret\_code = self.result\_code.SUCCESS\_NO\_ITEM\_PROCESSED.value  
 self.audit\_id = self.config.etl\_audit\_id  
  
 def validate(self, \*args, \*\*kwargs):  
 pass  
  
 @staticmethod  
 def get\_files():  
 """  
 Gets files from ETL\_FILE table.  
 :return: List of Db Objects  
 """  
 try:  
 log.info('Fetching files from ETL\_FILE\_TABLE..')  
 repo = EtlFileRepo()  
 model = repo.model  
 data = repo.query.filter(model.file\_source == 'DATS\_BBG\_IN',  
 model.is\_etl\_done == 0,  
 model.is\_ftp\_done == 1).all()  
 return data  
 except Exception as ex:  
 logging.exception('Error occured while fetching files from ETL\_FILE\_TABLE' + str(ex))  
 raise  
  
 @staticmethod  
 def \_get\_unique\_rows(df):  
 return df.drop\_duplicates(subset='REQUESTOR\_TAG', keep="first")  
  
 def get\_rows(self, csv\_file):  
 """  
 :param csv\_file: String - csv file location  
 :return: List  
 """  
 df = pd.read\_csv(csv\_file)  
 df = df.loc[df['ROW\_STATUS'] == 0]  
 df = self.\_get\_unique\_rows(df)  
 return [row for index, row in df.iterrows()]  
  
 def get\_dataframe(self, csv\_file):  
 """  
 :param csv\_file: String - csv file location  
 :return: List  
 """  
 df = pd.read\_csv(csv\_file)  
 df = df.loc[df['ROW\_STATUS'] == 0]  
 df = self.\_get\_unique\_rows(df)  
 return df  
  
 @staticmethod  
 def get\_csv\_loc(obj):  
 """  
 :param obj: DB Object  
 :return: String  
 """  
 log.info('Getting csv file location...')  
 csv\_file = os.path.join(obj.local\_file\_folder, obj.local\_file\_name)  
 return csv\_file  
  
 @staticmethod  
 def get\_series\_attr(dats\_code):  
 """  
 :param dats\_code: String  
 :return: List of Objects  
 """  
 try:  
 log.info('Fetching data from the DATS\_SERIES Table')  
 repo = DatsSeriesRepo()  
 model = repo.model  
 data = repo.query.filter(model.dats\_code.in\_(dats\_code)).all()  
 return data  
 except Exception as ex:  
 logging.exception('Error occured while fetching data from DATS\_SERIES TABLE' + str(ex))  
 raise  
  
 def \_insert\_stgp\_dats\_series\_value(self, csv\_row, dats\_series, obj):  
 try:  
 log.info('Update or Insert STGP\_DATS\_SERIES\_VALUE table..')  
 self.repo.stgp\_dats\_series\_value\_repo\_.instance.model(etl\_file\_id=obj.etl\_file\_id,  
 etl\_audit\_job\_id=self.audit\_id,  
 etl\_source\_code=dats\_series.etl\_source\_code,  
 source\_provider\_code=dats\_series.source\_provider\_code,  
 dats\_code=dats\_series.dats\_code,  
 asof\_date=datetime.strptime(csv\_row['ASOF\_DATE'],  
 "%m/%d/%Y"),  
 dats\_value=csv\_row['VALUE'])  
 self.repo.stgp\_dats\_series\_value\_repo\_.save()  
 except Exception as ex:  
 logging.exception(str(ex))  
 raise  
  
 def \_insert\_ups\_dats\_series\_value(self, csv\_row, dats\_series):  
 try:  
 log.info('Update or Insert UPS\_DATS\_SERIES\_VALUE table..')  
 self.repo.ups\_dats\_series\_value\_repo\_.instance.model(etl\_audit\_job\_id=self.audit\_id,  
 dats\_id=dats\_series.dats\_id,  
 asof\_date\_key=int(  
 datetime.strptime(csv\_row['ASOF\_DATE'],  
 "%m/%d/%Y").strftime("%Y%m%d")),  
 asof\_time\_id=0,  
 dats\_value\_str=str(csv\_row['VALUE']),  
 dats\_value\_str\_hash=text(  
 "ORA\_HASH({val})".format(  
 val=str(csv\_row['VALUE']))),  
 dats\_value\_num=int(  
 csv\_row['VALUE']),  
 row\_is\_active=0  
 )  
 self.repo.ups\_dats\_series\_value\_repo\_.save()  
 except Exception as ex:  
 logging.exception(str(ex))  
 raise  
  
 def insert\_value(self, csv\_row, dats\_series, obj):  
 """  
 :param csv\_row: Dataframe Object  
 :param dats\_series: DB Object  
 :param obj: DB Object  
 """  
 self.\_insert\_stgp\_dats\_series\_value(csv\_row, dats\_series, obj)  
 self.\_insert\_ups\_dats\_series\_value(csv\_row, dats\_series)  
  
 def \_get\_args(self):  
 args = argparse.Namespace(database='ORAPIM', etl\_audit\_id=self.audit\_id,  
 in\_param=['i\_audit\_id:{0}:Numeric'.format(self.audit\_id),  
 'i\_target\_table\_name:DATS\_SERIES\_VALUE:String',  
 'i\_source\_table\_name:UPS\_DATS\_SERIES\_VALUE:String'],  
 out\_param=['o\_insert\_count:0:Integer', 'o\_update\_count:0:Integer',  
 'o\_delete\_count:0:Integer'], scalar=False, schema='da\_own',  
 stored\_proc='da\_own.sp\_etl\_ups\_merge', timeout=300, vendor='oracle')  
 return args  
  
 def \_call\_sp\_etl\_ups\_merge(self):  
 da\_sp\_wrapper.get\_arguments = self.\_get\_args  
 da\_sp\_wrapper.main()  
 return  
  
 @contextmanager  
 def run(self):  
 file\_objs = self.get\_files()  
 if file\_objs:  
 for obj in file\_objs:  
 csv\_file = self.get\_csv\_loc(obj)  
 rows = self.get\_rows(csv\_file)  
 for row in rows:  
 if row['ROW\_STATUS'] != 0:  
 continue  
 data = self.get\_series\_attr(row['REQUESTOR\_TAG'])  
 self.insert\_value(row, data, obj)  
 self.\_call\_sp\_etl\_ups\_merge()  
 self.ret\_code = self.result\_code.SUCCESS\_ITEM\_PROCESSED  
 yield self.ret\_code  
  
 @contextmanager  
 def run\_1(self):  
 vals = pd.Dataframe()  
 file\_objs = self.get\_files()  
 if file\_objs:  
 for obj in file\_objs:  
 csv\_file = self.get\_csv\_loc(obj)  
 df = self.get\_dataframe(csv\_file)  
 data = self.get\_series\_attr(list(df['REQUESTOR\_TAG']))  
 for index, row in df.iterrows():  
 vals.append(pd.Series([row,data,obj],index=['row','data','obj']),ignore\_index = True)  
   
   
 # self.insert\_value(row, data, obj)  
 self.\_call\_sp\_etl\_ups\_merge()  
 self.ret\_code = self.result\_code.SUCCESS\_ITEM\_PROCESSED  
 yield self.ret\_code  
  
  
if \_name\_ == '\_main\_':  
 with LoaderAgent().run() as x:  
 logging.info('Agent execution complete.')  
 sys.exit(x)  
  
  
  
  
### bulk insert   
 def save\_bulk(self, data):  
 self.db.session.bulk\_insert\_mappings(self.model, data)  
 self.db.session.commit()  
   
 data here is list of dictionaries.