

Mongo Through ETL to OLAP

May 1, 2018

0.0.1 Prepared By Praveen Hariharasubramanian, Rushan Shakya, Tahrifa Mustafa, Berihun Mekonnen

1 ETL to OLAP

1.1 Project Purpose :

- OLAP Data model designed using SQL Workbench
- Extract SalesTrx (sales transaction information) collection from Mongo DB and apply date and store number filters on it
- Look up date from sales transactions and create datedim and timedim
- Extract Mongo DB collections to filter data for OLAP model
- Through the sales data merge data for customer, sales junk, store junk, store location, item list, item hierarchy, item junk, store service, item attributes to form Customer dimension, sales junk dimension, store junk dimension, store location dimension, item list dimension, item hierarchy dimension, item junk dimension, store service dimension and item attributes dimension which is a outrigger dimension linked through bridge table know as itembridge
- Building Star schema model for OLAP by merging the data from tables with dimension keys and the sales data. This way dimension keys were extracted to place them on the transaction fact table
- Filter the data from the sales transactions to dimensions
- Load data from customer file to respective dims
- Load data from item list file to respective dims
- Load data from store file to respective dims

1.1.1 Below is the code for our project with comments:

```
In [ ]: # Libraries Used for the Project
import pandas as pd
from pymongo import MongoClient
from datetime import datetime
from sqlalchemy import create_engine
from bson import json_util, ObjectId
from pandas.io.json import json_normalize
from datetime import timedelta
import calendar
import json
```

```

from bson import json_util, ObjectId
from pandas.io.json import json_normalize

```

connectToMongo - is a function to establish connection with mongoDB to extract data into python

```

def connectToMongo(hostname,port): """ This function establishes connection to Mongo DB
and returns connection object """ mc = MongoClient(host='127.0.0.1',port=27017) return mc

```

Getcollection is a function to establish to apply find on mongodb to read data from the collections in mongodb

```

In [ ]: def getCollection(database, collection, mongo_conn):
        """
        This function gets data for Store Location
        """
        db = mongo_conn.get_database(database)
        lists = db.get_collection(collection).find({},{'_id':0})
        return lists

```

GetSalesTrx function to apply filters on the transaction date time and store number to filter specific set of sales records and extract data from mongodb collections and store it in python data frames

```

In [ ]: def getSalesTrx(sDate,eDate,store,database, collection, mongo_conn):
        """
        This function gets data for Store Location
        """
        db = mongo_conn.get_database(database)
        lists = db.get_collection(collection).find({'StoreNum':store,
                                                    'TransDatetime(Local)'
                                                    :{'$lt': eDate, '$gte': sDate}},
                                                    {'_id':0})
        return lists

```

Convert to DF function to convert the data extracted from mongo to data frames

```

In [ ]: def convertToDF(lists):
        """
        This function converts lists of data extracted from mongo to Data Frames.
        """
        df= pd.DataFrame(list(lists))
        print("Number of rows : " + str(len(df)))
        return df

```

Connect to MySQL function to establish a connection with MySQL database

```

In [ ]: def connectMySQL(username, password, hostname, db):
        """
        This function establishes connection to MySQL and returns
        connection object username and password to be passed as

```

```

parameters to access the database of chocie with
hostname and databasename(db)
"""

#mysql+mysqlconnector://[username]:[password]@localhost/[database]
link = 'mysql+mysqlconnector://' + username + ':' + password + '@' + hostname + '/' + db
engine = create_engine(link)
return engine

```

Housekeeping function to flush out old data in the data frames and tables and create a fresh set of inserts into the MySQL database

```

In [ ]: def performHousekeeping(engine):
        """
        This function deletes all data records from the database table
        """
        list1 = ['DateDim', 'TimeDim', 'ItemListDim', 'ItemJunkDim', 'ItemHierarchyDim',
                  'StoreJunkDim', 'StoreLocationDim', 'SalesJunkDim', 'CustomerDim',
                  'ItemAttributesDim', 'StoreServiceDim', 'trans_fact']
        for i in list1:
            engine.execute('delete from ' + i)

```

Function to access the embedded documents within a document in mongoDB

```

In [ ]: def getEmbedded(mongo_data):
        """
        This function is to convert nested documents from mongoDB to DataFrames
        """
        sanitized = json.loads(json_util.dumps(mongo_data))
        normalized = json_normalize(sanitized)
        df = pd.DataFrame(normalized)
        return df

```

Creating a unknown customer to keep a record of the customers that do not have a loyalty card number

Creating dataframes for all the docuements from mongoDB

```

In [ ]: if __name__ == '__main__':
        #Creating a mongoDB connection object
        conn_obj = connectToMongo(hostname='127.0.0.1',port=27017)

        unknowncustrecord = dict({"LoyaltyCardNum":-999,"HouseholdNum":-999,
                                   "MemberFavStore":-999,"City":'-999',
                                   "State":'-999',"ZipCode":'-999'})
        unknowncustomer = conn_obj.get_database('BIPProject').get_collection('Customer')
        .insert_one(unkowncustrecord)

        """
        Get data from Mongo

```

```

"""
#Collect data from StoreLocation collection at mongoDB
storeDF = convertToDF(getCollection('BIPProject','StoreLocation',conn_obj))
print(storeDF.dtypes)
print(storeDF.isna().sum())

#Collect data from ItemAttribute collection at mongoDB
itemAttrDF = convertToDF(getCollection('BIPProject','ItemAttribute',conn_obj))
print(itemAttrDF.dtypes)
print(itemAttrDF.isna().sum())

#Collect data from ItemList collection at mongoDB
itemListDF = convertToDF(getCollection('BIPProject','ItemList',conn_obj))
print(itemListDF.dtypes)
print(itemListDF.isna().sum())

#Collect data from Customer collection at mongoDB
customerDF = convertToDF(getCollection('BIPProject','Customer',conn_obj))
print(customerDF.dtypes)
print(customerDF.isna().sum())

"""
Collect data from SalesTrx collection at mongoDB between specific transaction
dates at a particular store
"""
start = '2014-02-22 00:00:00'
end = '2014-02-23 00:00:00'
store = 562
saleDF = convertToDF(getSalesTrx(start, end,store,'BIPProject','SalesTrx',
                                conn_obj))
salejunkDF = saleDF[['StoreNum','Register','DeptNum','CashierNum','PriceType',
                    'ServiceType','TenderType']]
print(saleDF.dtypes)
print(saleDF.isna().sum())

#Extracting scraped data at mongoDB
DF = getEmbedded(getCollection('BIPProject','StoreScraped', conn_obj))

scrapedDF = DF[['Service.Alcohol','Service.Amarillo National Bank',
                'Service.Angus Beef','Service.Bakery','Service.Bill Pay',
                'Service.Boars Head','Service.Bulk Foods',
                'Service.Check Cashing','Service.City Bank',
                'Service.Clear Talk','Service.Coffee Shop','Service.Concierge',
                'Service.DMV Registration','Service.Deli',
                'Service.Dish Gift Center','Service.First Financial Bank',
                'Service.Floral','Service.Full Service Seafood',
                'Service.Herring National Bank','Service.Hot Deli',
                'Service.Keva Juice','Service.Living Well Dept',

```

```
'Service.Lottery','Service.Meals For Two','Service.Meat Market',
'Service.Red Box','Service.Restaurant','Service.Rug Doctor',
'Service.Salad Bar','Service.Sushi','Service.Team Spirit Shop',
'Service.Ticket Sales','Service.Walk-in Clinic',
'Service.Wells Fargo Bank','Service.Western Union',
'StoreId','StoreName']]
```

#renaming the scraped columns

```
scrapedDF.rename(columns={'Service.Alcohol':'Alcohol',
                          'Service.Amarillo National Bank':
                          'AmarilloNationalBank',
                          'Service.Angus Beef':'AngusBeef',
                          'Service.Bakery':'Bakery',
                          'Service.Bill Pay':'BillPay',
                          'Service.Boars Head':'BoarsHead',
                          'Service.Bulk Foods':'BulkFoods',
                          'Service.Check Cashing':'CheckCashing',
                          'Service.City Bank':'CityBank',
                          'Service.Clear Talk':'ClearTalk',
                          'Service.Coffee Shop':'CoffeeShop',
                          'Service.Concierge':'Concierge',
                          'Service.DMV Registration':'DMVregistration',
                          'Service.Deli':'Deli',
                          'Service.Dish Gift Center':'DishGiftCenter',
                          'Service.First Financial Bank':
                          'FirstFinancialBank',
                          'Service.Floral':'Floral',
                          'Service.Full Service Seafood':'FullServiceSeafood',
                          'Service.Herring National Bank':'HerringNationalBank',
                          'Service.Hot Deli':'HotDeli',
                          'Service.Keva Juice':'KevaJuice',
                          'Service.Living Well Dept':'LivingWellDept',
                          'Service.Lottery':'Lottery',
                          'Service.Meals For Two':'MealsForTwo',
                          'Service.Meat Market':'MeatMarket',
                          'Service.Red Box':'RedBox',
                          'Service.Restaurant':'Restaurant',
                          'Service.Rug Doctor':'RugDoctor',
                          'Service.Salad Bar':'SaladBar',
                          'Service.Sushi':'Sushi',
                          'Service.Team Spirit Shop':'TeamSpiritShop',
                          'Service.Ticket Sales':
                          'TicketSales','Service.Walk-in Clinic':'WalkInClinic',
                          'Service.Wells Fargo Bank':'WellsFargoBank',
                          'Service.Western Union':'WesternUnion',
                          'StoreId':'StoreNum','StoreName':'StoreType'},
                  inplace=True)
```

```
scrapedDF.drop_duplicates(keep='first')
```

Create a mysql connection and insert the sales fact data as base for merging dimensional data with it

Create all dimension tables with auto increment so that dimension keys get autogenerated through auto increment column in the table

Merge sales and customer data to categorise and link customers to the each sales transaction

```
In [ ]: """
        Create a mysql database connection
        """
        engine = connectMySQL('[username]', '[password]', '[hostname]',
                               '[database name]')

        #Delete all data from all tables in the database
        performHousekeeping(engine)

        """
        Inserting data values into Dimensions of the mysql database
        """

        #Inserting values into sales junk dimension
        #Merging customerDF with saleDF to get only those customers who are in our sales
        #transaction file
        cust_sale = pd.merge(customerDF, saleDF, left_on='LoyaltyCardNum',
                              right_on='LoyaltyCardNumber', how =
                              'inner').drop_duplicates(keep='first')
```

Inserting data into customerdim, storejunkdim and storelocationdim from the dataframes cust_Sale and storedf

```
In [ ]: #Inserting data into CustomerDim
        CustomerDim = cust_sale[['LoyaltyCardNum', 'HouseholdNum', 'MemberFavStore',
                                   'City', 'State', 'ZipCode']]
        .drop_duplicates(keep='first')
        .drop_duplicates(keep='first')

        CustomerDim.to_sql('CustomerDim', engine, if_exists='append', index=False)

        #Inserting data into StoreJunkDim
        storeDF[['StoreNum', 'StoreName', 'ActiveFlag', 'SqFoot', 'ClusterName']]
        .to_sql('StoreJunkDim', engine, if_exists='append', index=False)

        #Inserting data into StoreLocationDim
        StoreLocationDim = storeDF[['Region', 'StateCode', 'City', 'ZipCode',
                                      'AddressLine1']]
        .drop_duplicates(keep='first')
```

```
StoreLocationDim.to_sql('StoreLocationDim', engine, if_exists='append',
                        index=False)
```

Merged item information with sales data is loaded into all item related dimensions

```
In [ ]: #Merging itemListDF and saleDF to get only those items which are in our sales
        item_sale = pd.merge(itemListDF, saleDF, left_on=['UPC', 'ItemID'],
                              right_on=['UPC', 'ItemID'], how='inner')
        .drop_duplicates(keep='first')

        #Inserting data into ItemListDim
        ItemListDim = item_sale[['UPC', 'ItemID', 'LongDes', 'ShortDes', 'ExtraDes']]
        .drop_duplicates(keep='first')

        ItemListDim.to_sql('ItemListDim', engine, if_exists='append', index=False)

        #Inserting data into item hierarchy
        ItemHierarchyDim = item_sale[['DepartmentCode', 'FamilyCode', 'FamilyDes',
                                       'CategoryCode', 'CategoryDes', 'ClassCode',
                                       'ClassDes']]
        .drop_duplicates(keep='last').astype(str).drop_duplicates(keep='first')

        ItemHierarchyDim.to_sql('ItemHierarchyDim', engine, if_exists='append',
                                index=False)

        #Inserting data into ItemJunkDim
        ItemJunkDim = item_sale[['StoreBrand', 'Status']].drop_duplicates(keep='first')
        .drop_duplicates(keep='first')

        ItemJunkDim.to_sql('ItemJunkDim', engine, if_exists='append', index=False)

        #Inserting into scraped StoreServicesDim
        scrapedDF.to_sql('StoreServiceDim', engine, if_exists='append', index=False)
```

Creating a item attribute outrigger dimension linked to item dimension through bridge table item_Bridge

```
In [ ]: #Inserting into item attributes dimension
        itemattrnitem = pd.merge(item_sale, itemAttrDF, left_on=['UPC'],
                                   right_on=['UPC'], how='inner')
        .drop_duplicates(keep='first')
        ItemattributesDim = itemattrnitem[['UPC', 'ItemAttributeValue',
                                             'ItemAttributeDes', 'AttributeStartDate',
                                             'AttributeEndDate']]
        ItemattributesDim.to_sql('ItemattributesDim', engine, if_exists='append',
                                index=False)

        #Inserting into ItemBridge table
        itemdim = pd.read_sql_table('itemlistdim', engine, columns=['ILDK', 'UPC'])
```

```

itemattrdim = pd.read_sql_table('itemattributesdim', engine,
                                columns=['IADK', 'UPC'])
itembridgedf = pd.merge(itemdim, itemattrdim, left_on=['UPC'],
                        right_on=['UPC'], how='inner')
.itemdrop_duplicates(keep='first')
Itembridge = itembridgedf[['IADK', 'ILDK']]
Itembridge.rename(index = str, columns={'IADK': 'ItemAttribute_IADK',
                                         'ILDK': 'ItemList_ILDK'})
ItemattributesDim.to_sql('ItemattributesDim', engine, if_exists='append',
                        index=False)

```

Date and time dimensions created through transaction time in sales fact table

In []: *# Date Dimension*

```

temp = pd.DatetimeIndex(saleDF['TransDatetime(GMT)'])
saleDF['Year_int'] = temp.year
saleDF['Month_int'] = temp.month
saleDF['Month_abbr'] = saleDF['Month_int'].apply(lambda x: calendar
                                                .month_abbr[x])

saleDF['Day_int'] = temp.day
saleDF['DayOfWeek_int'] = temp.dayofweek
saleDF['DayOfWeek_char'] = saleDF['DayOfWeek_int'].apply(lambda x: calendar
                                                         .day_name[x])

saleDF['DayOfYear_int'] = temp.dayofyear
saleDF['Date'] = saleDF['TransDatetime(GMT)']
datedim = saleDF[['TransDatetime(GMT)', 'Date', 'Year_int', 'Month_int',
                  'Month_abbr', 'Day_int', 'DayOfWeek_int', 'DayOfWeek_char',
                  'DayOfYear_int']]
.itemdrop_duplicates(keep='first')

datedim[['Date', 'Year_int', 'Month_int', 'Month_abbr', 'Day_int',
         'DayOfWeek_int', 'DayOfWeek_char', 'DayOfYear_int']]
.to_sql('DateDim', engine, if_exists='append', index=False)

```

#Time Dimension

```

saleDF['Time_hhmmss_char'] = temp.time.astype(str)
saleDF['Hour_24_int'] = temp.hour
saleDF['Time'] = saleDF['Time_hhmmss_char']
saleDF['Minute_int'] = temp.minute
saleDF['Second_int'] = temp.second
temp_12hour = saleDF['TransDatetime(GMT)'] + timedelta(hours=12)
saleDF['Hour_12_int'] = pd.DatetimeIndex(temp_12hour).hour
saleDF['AM_PM_char'] = saleDF['Hour_12_int']
i=0
ampm = []
for i in list(range(0,24)):
    if(i < 12):
        ampm.append('AM')

```



```

        i=i+1
    elif(i>=12):
        ampm.append('PM')
        i=i+1
mapping = dict(zip(list(range(0,24)),ampm))
saleDF.replace({'AM_PM_char': mapping})
#print(saleDF)

timedim = saleDF[['TransDatetime(GMT)', 'Time', 'Hour_24_int', 'Minute_int',
                  'Second_int', 'Hour_12_int',
                  'AM_PM_char']].drop_duplicates(keep='first')

timedim.rename(index = str, columns ={'Time': 'Time_hhmmss_char',
                                     'Hour_24_int': 'Hour_24_int',
                                     'Minute_int': 'Minute_int',
                                     'Second_int': 'Second_int',
                                     'Hour_12_int': 'Hour_12_int',
                                     'AM_PM_char': 'AM_PM_char'})

timedim[['Time', 'Hour_24_int', 'Minute_int', 'Second_int', 'Hour_12_int',
          'AM_PM_char']]
    .to_sql('TimeDim', engine, if_exists='append', index=False)

```

Sales junk dimension loaded from sale dataframe

```

In [ ]: #Sales Junk Dimension records are inserted
salejunkdimtable = salejunkDF[['Register', 'DeptNum', 'CashierNum', 'PriceType',
                               'ServiceType', 'TenderType']]
    .drop_duplicates(keep='first')
salejunkdimtable.to_sql('salesjunkdim', engine, if_exists='append',
                        index=False)

```

FACT table loaded by merging the sales DF and each dimension table to filter the dimension keys and load the dimension keys to the sales transaction fact table
 Extracting sales junk dimension keys to salesfact dataframe

```

In [ ]: #Inserting FACT records in FACT table for sales transaction data

```

```

salesfact = saleDF[['UPC', 'ItemID', 'TransDatetime(GMT)', 'StoreNum', 'WeightAmt',
                    'SalesAmt', 'BusDate', 'TransNum', 'ItemQuantity', 'CostAmt',
                    'Register', 'DeptNum', 'CashierNum', 'PriceType', 'ServiceType',
                    'TenderType']]

salesjunktable = pd.read_sql_table('salesjunkdim', engine)
salesjunkfact = pd.merge(salejunkDF[['StoreNum', 'Register', 'DeptNum', 'CashierNum',
                                     'PriceType', 'ServiceType', 'TenderType']],
                        salesjunktable,
                        left_on=['Register', 'DeptNum', 'CashierNum', 'PriceType',
                                'ServiceType', 'TenderType'],

```

```

right_on=['Register', 'DeptNum', 'CashierNum', 'PriceType',
          'ServiceType', 'TenderType'],
how = 'inner').drop_duplicates(keep='first')

salesfact = pd.merge(salesfact[['UPC', 'ItemID', 'TransDatetime(GMT)', 'WeightAmt',
                                'SalesAmt', 'BusDate', 'TransNum', 'ItemQuantity',
                                'CostAmt', 'Register', 'DeptNum', 'CashierNum',
                                'PriceType', 'ServiceType', 'TenderType',
                                'StoreNum']],
                    salesjunkfact[['SJDK', 'StoreNum', 'Register', 'DeptNum',
                                    'CashierNum', 'PriceType', 'ServiceType',
                                    'TenderType']],
                    left_on=['StoreNum', 'Register', 'DeptNum', 'CashierNum',
                              'PriceType', 'ServiceType', 'TenderType'],
                    right_on=['StoreNum', 'Register', 'DeptNum', 'CashierNum',
                              'PriceType', 'ServiceType', 'TenderType'],
                    how = 'inner').drop_duplicates(keep='first')

```

Extracting item list dimension keys to itemfact dataframe

```

In [ ]: itemfact = pd.read_sql_table('itemlistdim', engine,
                                     columns=['ILDK', 'UPC', 'ItemID'])
itemsalesfact = pd.merge(itemfact, salesfact, left_on=['UPC', 'ItemID'],
                          right_on=['UPC', 'ItemID'], how = 'inner')
                          .drop_duplicates(keep='first')

```

Date and time dimension keys to datefact and timefact dataframe

```

In [ ]: datefact = pd.read_sql_table('datedim', engine, columns=['DDK', 'Date'])
itemsalesdatefact = pd.merge(itemsalesfact, datefact, left_on=['TransDatetime(GMT)'],
                              right_on=['Date'], how = 'inner')
                              .drop_duplicates(keep='first')

itemsalesdatefact['time'] = pd.DatetimeIndex(itemsalesdatefact['Date']).time

timefact = pd.read_sql_table('timedim', engine, columns=['TDK', 'Time'])
itemsalesdatetimefact = pd.merge(itemsalesdatefact, timefact,
                                  left_on=['time'], right_on=['Time'],
                                  how = 'inner').drop_duplicates(keep='first')

itemsaledtfact = itemsalesdatetimefact

```

item junk dimension keys to itemjunkfact dataframe

```

In [ ]: itemjunktable = pd.read_sql_table('itemjunkdim', engine,
                                           columns=['IJDK', 'StoreBrand', 'Status'])
itemjunk = pd.merge(itemjunktable, item_sale[['StoreBrand', 'Status', 'UPC',
                                                'ItemID', 'ClassCode', 'CategoryCode',
                                                'DepartmentCode', 'FamilyCode']],

```

```

        left_on=['StoreBrand', 'Status'], right_on=['StoreBrand', 'Status'],
        how = 'inner').drop_duplicates(keep='first')
itemjunkfact = pd.merge(itemsaledtfact, itemjunk, left_on=['UPC', 'ItemID'],
        right_on=['UPC', 'ItemID'],
        how = 'inner').drop_duplicates(keep='first')

```

aligning the columns from dataframes and dimension tables to a common data type

```

In [ ]: allitemsalesdtfact=itemjunkfact
        itemtemp_sale=item_sale
        allitemsalesdtfact['sales_SJDK'] = allitemsalesdtfact['SJDK']

        itemtemp_sale['ClassCode']=itemtemp_sale['ClassCode'].astype(str)
        itemtemp_sale['CategoryCode']=itemtemp_sale['CategoryCode'].astype(str)
        itemtemp_sale['DepartmentCode']=itemtemp_sale['DepartmentCode'].astype(str)
        itemtemp_sale['FamilyCode']=itemtemp_sale['FamilyCode'].astype(str)
        allitemsalesdtfact['ClassCode']=allitemsalesdtfact['ClassCode'].astype(str)
        allitemsalesdtfact['CategoryCode']=allitemsalesdtfact['CategoryCode'].astype(str)
        allitemsalesdtfact['DepartmentCode']=allitemsalesdtfact['DepartmentCode'].astype(str)
        allitemsalesdtfact['FamilyCode']=allitemsalesdtfact['FamilyCode'].astype(str)

```

item hierarchy dimension key to itemhierfact dataframe

```

In [ ]: itemhiertable = pd.read_sql_table('itemhierarchydim', engine,
        columns=['IHDK', 'ClassCode', 'CategoryCode',
        'DepartmentCode', 'FamilyCode'])

        itemhiertemp = pd.merge(itemhiertable, itemtemp_sale,
        left_on=['ClassCode', 'CategoryCode', 'DepartmentCode',
        'FamilyCode'],
        right_on=['ClassCode', 'CategoryCode', 'DepartmentCode',
        'FamilyCode'],
        how = 'inner').drop_duplicates(keep='first')

        itemhierfact = pd.merge(allitemsalesdtfact[['sales_SJDK', 'ILDK', 'UPC', 'ItemID',
        'TransDatetime(GMT)', 'StoreNum',
        'WeightAmt', 'SalesAmt', 'BusDate',
        'TransNum', 'ItemQuantity', 'CostAmt',
        'DDK', 'Date', 'TDK', 'IJDK', 'ClassCode',
        'CategoryCode', 'DepartmentCode',
        'FamilyCode']],
        itemhiertemp[['IHDK', 'ClassCode', 'CategoryCode',
        'DepartmentCode', 'FamilyCode']],
        left_on=['ClassCode', 'CategoryCode', 'DepartmentCode',
        'FamilyCode'],
        right_on=['ClassCode', 'CategoryCode', 'DepartmentCode',
        'FamilyCode'],
        how = 'inner').drop_duplicates(keep='first')

```

```
halfact = itemhierfact
```

store location dimension key and store junk dimension keys to storefact and storejunkfact dataframe

```
In [ ]: storetable = pd.read_sql_table('storelocationdim', engine,
                                         columns=['SLDK', 'Region', 'StateCode', 'City',
                                                  'ZipCode', 'AddressLine1'])
storetablefact = pd.merge(storeDF[['StoreName', 'ClusterName', 'StoreNum',
                                     'Region', 'StateCode', 'City', 'ZipCode',
                                     'AddressLine1']],
                           storetable[['SLDK', 'Region', 'StateCode', 'City',
                                          'ZipCode', 'AddressLine1']],
                           left_on=['Region', 'StateCode', 'City', 'ZipCode', 'AddressLine1'],
                           right_on=['Region', 'StateCode', 'City', 'ZipCode', 'AddressLine1'],
                           how='inner').drop_duplicates(keep='first')
storefact = pd.merge(halfact[['sales_SJDK', 'IHDK', 'ILDK', 'UPC', 'ItemID',
                               'TransDatetime(GMT)', 'StoreNum', 'WeightAmt',
                               'SalesAmt', 'BusDate', 'TransNum', 'ItemQuantity',
                               'CostAmt', 'DDK', 'Date', 'TDK', 'IJDK', 'ClassCode',
                               'CategoryCode', 'DepartmentCode', 'FamilyCode']],
                      storetablefact[['StoreName', 'ClusterName', 'StoreNum', 'SLDK',
                                         'Region', 'StateCode', 'City', 'ZipCode',
                                         'AddressLine1']],
                      left_on=['StoreNum'], right_on=['StoreNum'],
                      how='inner').drop_duplicates(keep='first')
storejunktable = pd.read_sql_table('storejunkdim', engine,
                                    columns=['SJDK', 'StoreName', 'ClusterName', 'StoreNum',
                                             'ActiveFlag', 'SqFoot'])
storejunkfact = pd.merge(storefact[['sales_SJDK', 'SLDK', 'IHDK', 'ILDK', 'UPC', 'ItemID',
                                     'ClusterName', 'StoreName', 'TransDatetime(GMT)',
                                     'StoreNum', 'WeightAmt', 'SalesAmt', 'BusDate',
                                     'TransNum', 'ItemQuantity', 'CostAmt', 'DDK', 'Date',
                                     'TDK', 'IJDK', 'ClassCode', 'CategoryCode',
                                     'DepartmentCode', 'FamilyCode']],
                          storejunktable[['StoreName', 'ClusterName', 'StoreNum',
                                             'SJDK', 'ActiveFlag', 'SqFoot']],
                          left_on=['StoreName', 'ClusterName', 'StoreNum'],
                          right_on=['StoreName', 'ClusterName', 'StoreNum'],
                          how='inner')
.drop_duplicates(keep='first')
```

store service dimension keys to prefinalfact dataframe

```
In [ ]: storeservicetable = pd.read_sql_table('storeservicedim', engine)
prefinalfact = pd.merge(storejunkfact, storeservicetable,
                         left_on=['StoreNum'],
```

```

        right_on=['StoreNum'], how = 'inner')
    .drop_duplicates(keep='first')

customer dimension key to finalfact dataframe

In [ ]: prefinalfact= prefinalfact[['SSDK','SJDk','sales_SJDk','SLDK','IHDK',
        'ILDK','UPC','ItemID','ClusterName',
        'StoreName','TransDatetime(GMT)',
        'StoreNum','WeightAmt','SalesAmt','BusDate',
        'TransNum','ItemQuantity','CostAmt','DDK','Date',
        'TDK','IJDk','ClassCode',
        'CategoryCode','DepartmentCode','FamilyCode']]

customertable = pd.read_sql_table('customerdim', engine)
customerfact = pd.merge(saleDF[['LoyaltyCardNumber']], customertable,
        left_on=['LoyaltyCardNumber'],
        right_on=['LoyaltyCardNum'], how = 'inner')
    .drop_duplicates(keep='first')

prefinalfact = pd.merge(saleDF[['TransNum','LoyaltyCardNumber']],
        prefinalfact[['SSDK','sales_SJDk','SJDk','SLDK',
        'IHDK','ILDK','UPC','ItemID','ClusterName',
        'StoreName','TransDatetime(GMT)','StoreNum',
        'WeightAmt','SalesAmt','BusDate','TransNum',
        'ItemQuantity','CostAmt','DDK','Date','TDK',
        'IJDk','ClassCode','CategoryCode',
        'DepartmentCode','FamilyCode']],
        left_on=['TransNum'], right_on=['TransNum'], how = 'inner')
    .drop_duplicates(keep='first')

finalfact = pd.merge(prefinalfact[['SSDK','LoyaltyCardNumber','sales_SJDk','SJDk',
        'SLDK','IHDK','ILDK','UPC','ItemID','ClusterName',
        'StoreName','TransDatetime(GMT)','StoreNum',
        'WeightAmt','SalesAmt','BusDate','TransNum',
        'ItemQuantity','CostAmt','DDK','Date','TDK',
        'IJDk','ClassCode','CategoryCode',
        'DepartmentCode','FamilyCode']],
        customerfact[['CDK','LoyaltyCardNum']],
        left_on=['LoyaltyCardNumber'],
        right_on=['LoyaltyCardNum'], how = 'inner')
    .drop_duplicates(keep='first')

In [ ]: #loading data to the fact table

In [ ]: fact = finalfact[['SSDK','CDK','sales_SJDk','SJDk','SLDK','IHDK','ILDK','UPC',
        'ItemID','ClusterName','StoreName',
        'TransDatetime(GMT)','StoreNum','WeightAmt',
        'SalesAmt','BusDate','TransNum','ItemQuantity',
        'CostAmt','DDK','Date','TDK','IJDk',
        'ClassCode','CategoryCode','DepartmentCode',

```

```

        'FamilyCode']]

fact.rename(index = str, columns={'CustomerDim_CDK':'CDK',
                                  'SalesJunkDim_SJDK':'sales_SJDK',
                                  'StoreJunkDim_SJDK':'SJDK',
                                  'StoreServiceDim_SSDK':'SSDK',
                                  'StoreLocationDim_SLDK':'SLDK',
                                  'ItemHierarchyDim_IHDK':'IHDK',
                                  'ItemJunkDim_IJDK':'IJDK',
                                  'ItemListDim_ILDK':'ILDK',
                                  'TimeDim_TDK':'TDK',
                                  'DateDim_DDK':'DDK'})

fact[['CustomerDim_CDK']]=fact[['CDK']]
fact[['SalesJunkDim_SJDK']]=fact[['sales_SJDK']]
fact[['StoreJunkDim_SJDK']]=fact[['SJDK']]
fact[['StoreServiceDim_SSDK']]=fact[['SSDK']]
fact[['StoreLocationDim_SLDK']]=fact[['SLDK']]
fact[['ItemHierarchyDim_IHDK']]=fact[['IHDK']]
fact[['ItemJunkDim_IJDK']]=fact[['IJDK']]
fact[['ItemListDim_ILDK']]=fact[['ILDK']]
fact[['TimeDim_TDK']]=fact[['TDK']]
fact[['DateDim_DDK']]=fact[['DDK']]
fact[['WeightAmt']] = fact[['WeightAmt']].round(1)

fact = fact[['CustomerDim_CDK','SalesJunkDim_SJDK','StoreJunkDim_SJDK',
              'StoreServiceDim_SSDK','StoreLocationDim_SLDK',
              'ItemHierarchyDim_IHDK','ItemJunkDim_IJDK','ItemListDim_ILDK',
              'TimeDim_TDK','DateDim_DDK','BusDate',
              'TransNum','ItemQuantity','WeightAmt','SalesAmt','CostAmt']]

fact.to_sql('trans_fact', engine, if_exists='append', index=False)

```

2 Problems solved by the OLAP Project

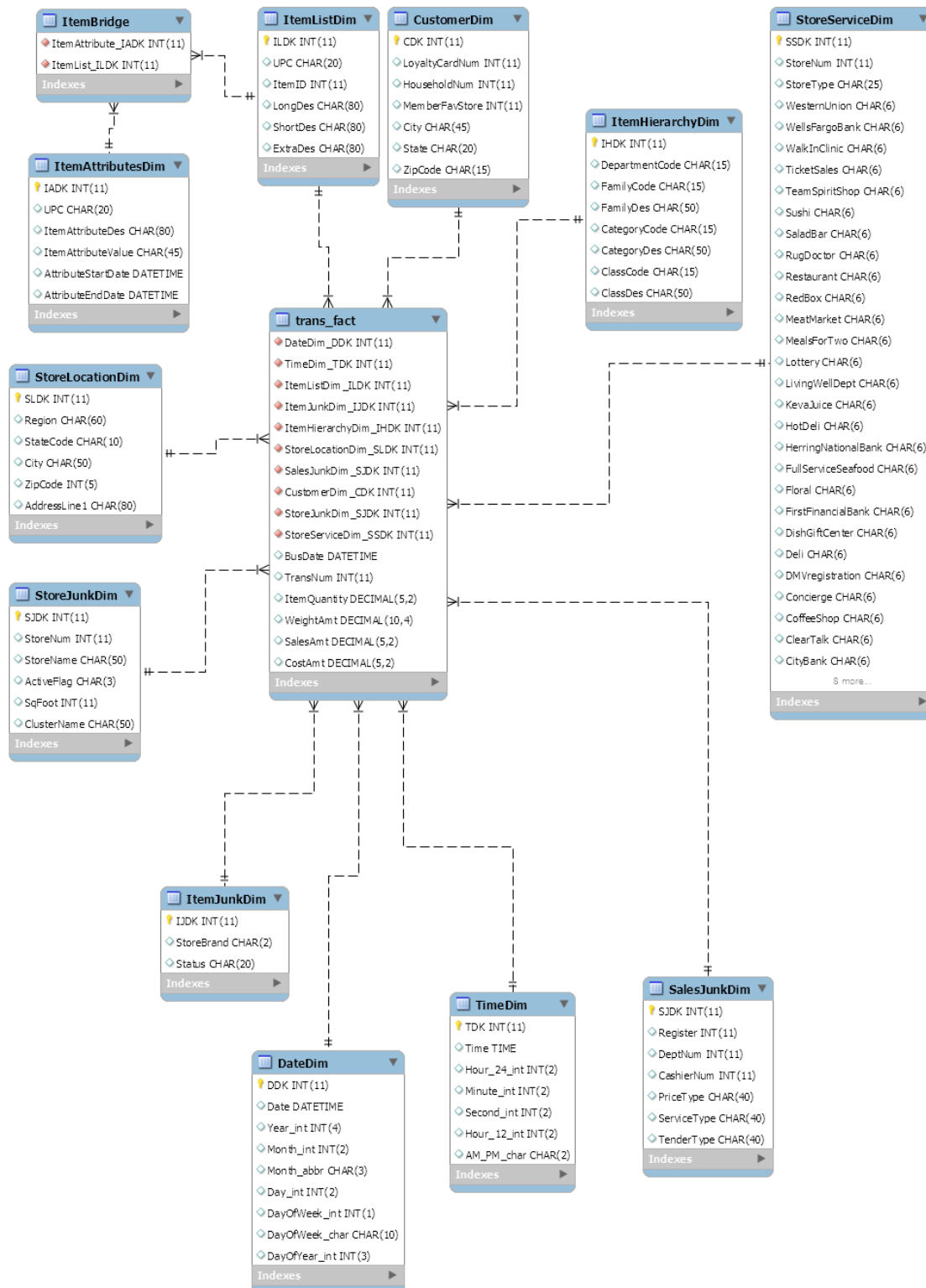
Challenges and Solutions in the Code Development Process:

1. Timeout errors and solutions
2. Single merge was causing memory error so had to apply multiple merges to filter dimension keys from dimensions for loading the fact
3. Connecting with different Developers' Teams
4. There were transactions where Customers made purchase transactions, but they did not have a LoyaltyCardNum. So, we inserted -999 as their LoyaltyCardNum in the CustomerDim to link it with transactions. The value -999 in CustomerDim is the Unknown customer at our OLAP.

3 OLAP ERD

```
In [4]: from IPython.display import Image  
        Image(filename= "C:/ttu/spring18/BI/MongoDB2OLAP-master/erd.png")
```

Out[4]:



This is the ERD diagram we created for our OLAP. We have an outrigger dimension for ItemListDim which has a Bridge table to link the ItemAttributeDim. We decided to use ItemAttribute

as an outrigger dimensions because we have many Item Attributes for a single product (which is made unique using the UPC key and ItemID Keys).

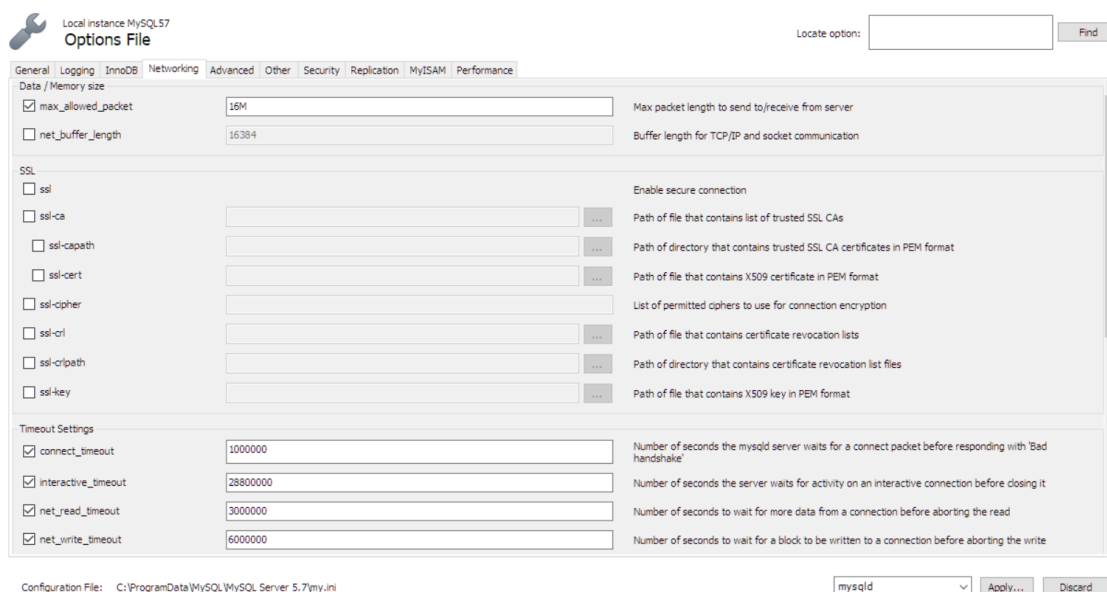
4 Problems with Bulk inserts using sqlalchemy, and how to solve it

While using SQLAlchemy package to load our dimensions, we found that we can do bulk inserts into our dimension table if we are trying to insert million rows of data. So, to enable bulk inserts into MySQL, we had to make changes to a few settings at MySQL. The settings are shown below:

1. Once you are at your Local Instance tab.
2. Go to Server >> Options File
3. Select Networking tab
4. Check Max allowed and change its value to 16M
5. Scroll down to Timeout settings and use the following settings:

In [7]: `Image(filename= "C:/ttu/spring18/BI/MongoDB20LAP-master/options file.png")`

Out [7]:



5 Problems with running out of memory while loading the data frames and then loading the data from data frames to MySQL

While we were loading data from MongoDB to data frames and then loading to MySQL, we ran out of physical memory in our local machines. We decided to work around it by only using collections that we were working with into our data frames. After we were done with the data frames, we cleared the variables once the needed data was inserted into MySQL.

In the variable explorer, use the eraser to select which variable to clear out from memory.