CREATE EXTERNAL TABLE IF NOT EXISTS dw.factTable

(sessionNaturalKey STRING, sessionKey int, requestedDateKey int, requestedTimeKey

int, startDateKey int, startTimeKey int, endDateKey int, endTimeKey int,

tutorKey int, studentKey int, courseKey int, type STRING, status STRING,

didAttend boolean, cancellationReason STRING)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

LOCATION 'hdfs:/dimensions/factDim';

CREATE EXTERNAL TABLE IF NOT EXISTS dw.dateDimension

(date STRING, dateKey int, dayName STRING, dayNameAbbrev STRING, dayOfMonth

int, dayOfWeek int, dayOfYear int, fiscalMonthNumber int, fiscalYear int,

holidayName STRING, isHoliday boolean, isWeekday boolean, isWeekend boolean,

monthAbbrev STRING, monthEndFlag boolean, monthName STRING, monthNumberquarter int, quarterNumber int, quarterName STRING, quarterShortName STRING,

sameDayPreviousYear STRING, sameDayPreviousYearKey int, season STRING,

weekBeginDate STRING, weekBeginDateKey int, weekNumInMonth int, weekNumInYear

int, year int, yearAndMonth STRING, yearAndMonthAbbrev STRING, yearAndQuarter

STRING)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

LOCATION 'hdfs:/dimensions/dateDim';

CREATE EXTERNAL TABLE IF NOT EXISTS dw.timeDimension

(timeKey int, militaryHour int, civilianHour int, minute int, ampm STRING,

militaryTime STRING, civilianTime STRING, timeClass STRING)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

LOCATION 'hdfs:/dimensions/timeDim';

CREATE EXTERNAL TABLE IF NOT EXISTS dw.courseDimension

(courseKey int, division int, courseCode STRING, courseTitle STRING, capacity

int, beginTime STRING, endTime STRING, days STRING, faculty STRING)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

LOCATION 'hdfs:/dimensions/courseDim';

CREATE EXTERNAL TABLE IF NOT EXISTS dw.courseMap

(courseKey int, coursecode String)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

LOCATION 'hdfs:/dimensions/courseMap';

CREATE EXTERNAL TABLE IF NOT EXISTS dw.studentDimension

(studentNaturalKey STRING, studentKey int, AUID STRING, firstName STRING, lastName STRING, school STRING, major STRING, email STRING, phoneNumber STRING, isBlocked boolean, role STRING, maxHours int, isActive boolean)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

LOCATION 'hdfs:/dimensions/studentDim';

CREATE VIEW dw.tutorDimension AS

SELECT \* FROM dw.studentDimension

WHERE role=’serviceProvider’;

SHOW databases;

USE dw;

SHOW Tables;

Mongodump --db kudos --drop C://Documents/kudosBackUp

Mongorestore --db kudos --drop

SELECT ((SELECT COUNT(\*) WHERE endDateKey = NULL) \* 100 / (SELECT COUNT (\*) FROM dw.factTable)) AS AttendanceRate FROM dw.factTable

SBA: **938 => 46.34%**

SSE: **796 => 39.32%**

SHSS: **290 => 14.32%**

Query used:

select COUNT(\*) from studentDimension

where school==<SCHOOL>

AND

        select (select count(\*)

          from studentDimension where school='SSE')/count(\*)\*100

          from studentDimension

Session Missed Ratio: **19.07% missed vs. 80.92% Attended**

SELECT sum(case when didAttend = 'FALSE' then 1 else 0 end)/count(\*) as missed\_ratio,

       sum(case when didAttend` = 'true' then 1 else 0 end)/count(\*) as attended)ratior

FROM factTable

Ratio of use by SBA students: **44.73%**

SELECT (SELECT COUNT(\*) from factTable as FT JOIN studentKey as ST on FT.studentKey = ST.studentKey

        where school=’SBA’)/count(\*) as SSE Use,

FROM factTable

Ratio of use by SSE students: **44.68%**

SELECT (SELECT COUNT(\*) from factTable as FT JOIN studentKey as ST on FT.studentKey = ST.studentKey

        where school='SSE')/count(\*) as SSE Use,

FROM factTable

Ratio of use by SHSS students: **10.58%**

SELECT (SELECT COUNT(\*) from factTable as FT JOIN studentKey as ST on FT.studentKey = ST.studentKey

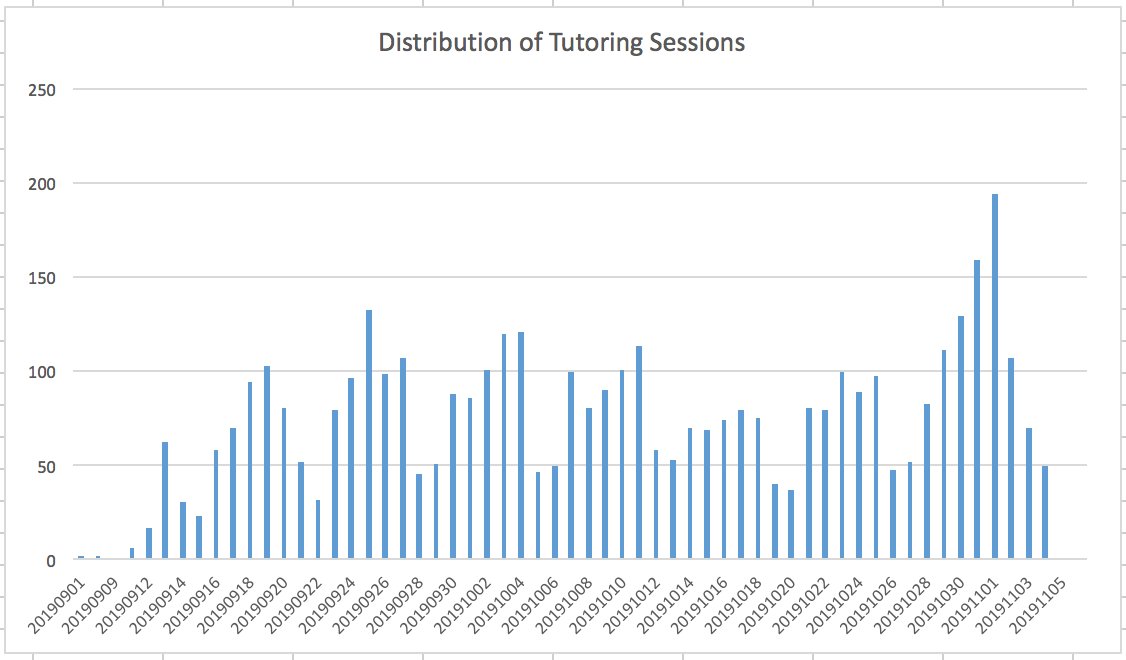
        where school=’SHSS’)/count(\*) as SSE Use,

FROM factTable

Number of sessions per day:

SELECT startdatekey, count(\*) as NumberOfSession FROM factTable GROUP BY startDateKey

Curve:



Similar to previous query but by school:

Number of blocked users in SBA: **2**

select count(\*) from studentDimension

where isBlocked=='true' and school==’SBA’

Number of blocked users in SHSS: **0**

select count(\*) from studentDimension

where isBlocked=='true' and school==’SHSS’

Number of blocked users in SSE: **0**

select count(\*) from studentDimension

where isBlocked=='true' and school=='SSE'

export HIVE\_HOME=/usr/local/hive

export PATH=$PATH:$HIVE\_HOME/bin

export CLASSPATH=$CLASSPATH:/usr/local/Hadoop/lib/\*:.

export CLASSPATH=$CLASSPATH:/usr/local/hive/lib/\*:.

Wget <http://archive.apache.org/dist/db/derby/db-derby-10.4.2.0/db-derby-10.4.2.0-bin.tar.gz>

My ETL Key Code Snippets:

const studentDimensionCSV = createCsvWriter({

path: './studentDimension.csv',

header: [

{ id: 'studentNaturalKey', title: 'studentNaturalKey', default: null },

{ id: 'studentKey', title: 'studentKey', default: null },

{ id: 'AUID', title: 'AUID', default: null },

{ id: 'firstName', title: 'firstName', default: null },

{ id: 'lastName', title: 'lastName', default: null },

{ id: 'school', title: 'school', default: null },

{ id: 'major', title: 'major', default: null },

{ id: 'email', title: 'email', default: null },

{ id: 'phoneNumber', title: 'phoneNumber', default: null },

{ id: 'blocked', title: 'isBlocked', default: null },

{ id: 'role', title: 'role', default: null },

{ id: 'maxHours', title: 'maxHours', default: null },

{ id: 'active', title: 'active', default: null }

]

});

const factTableCSV = createCsvWriter({

path: './factTable.csv',

header: [

{ id: 'sessionNaturalKey', title: 'sessionNaturalKey', default: null },

{ id: 'sessionKey', title: 'sessionKey', default: null },

{ id: 'requestedDateKey', title: 'requestedDateKey', default: null },

{ id: 'requestedTimeKey', title: 'requestedTimeKey', default: null },

{ id: 'startDateKey', title: 'startDateKey', default: null },

{ id: 'startTimeKey', title: 'startTimeKey', default: null },

{ id: 'endDateKey', title: 'endDateKey', default: null },

{ id: 'endTimeKey', title: 'endTimeKey', default: null },

{ id: 'tutorKey', title: 'tutorKey', default: null },

{ id: 'studentKey', title: 'studentKey', default: null },

{ id: 'courseKey', title: 'courseKey', default: null },

{ id: 'type', title: 'type', default: null },

{ id: 'status', title: 'status', default: null },

{ id: 'didAttend', title: 'didAttend', default: null },

{ id: 'cancellationReason', title: 'cancellationReason', default: null },

{ id: 'topic', title: 'topic', default: null },

{ id: 'professor', title: 'professor', default: null },

{ id: 'location', title: 'location', default: null },

{ id: 'duration', title: 'duration', default: 0 }

]

});

exports.generateStudentDimenssion = async function(req, res, next) {

try {

let inc = 0;

let records = [];

let users = await db.User.find({});

users.forEach(user => {

inc++;

records.push({

studentNaturalKey: user.\_id,

studentKey: inc,

AUID: user.AUID,

firstName: user.profile.firstname,

lastName: user.profile.lastname,

school: user.profile.school,

major: user.profile.major,

email: user.profile.email,

phoneNumber: user.profile.phoneNumber,

blocked: user.blocked.toString(),

role: user.role,

active: user.authentication.active,

maxHours: user.maxHours

});

});

return studentDimensionCSV

.writeRecords(records)

.then(() => {

console.log('...Done');

})

.then(() => {

return res.status(200).json({

status: '...Conversion Completed'

});

});

} catch (e) {

console.log(e);

return next({

status: 401,

message: 'Wrong credentials'

});

}

};

async function asyncForEach(array, callback) {

for (let index = 0; index < array.length; index++) {

await callback(array[index], index, array);

}

}

exports.generateFactTable = async function(req, res, next) {

try {

let inc = 1;

let records = [];

let sessions = await db.Session.find({});

asyncForEach(sessions, async session => {

let sessionNaturalKey = session.\_id;

let sessionKey = inc++;

let type = session.sessionType;

let status = session.sessionStatus;

let didAttend = session.attendance.length ? true : false;

let topic = session.requestInfo.topic;

let professor = session.requestInfo.professor;

let requestedDateKey =

(

parseInt(session.requestInfo.requestedAt.getMonth(), 10) + 1

).toString() +

'/' +

session.requestInfo.requestedAt.getDate() +

'/' +

(

parseInt(session.requestInfo.requestedAt.getYear(), 10) % 100

).toString();

let minutes = session.requestInfo.requestedAt.getMinutes().toString();

if (minutes.length === 1) minutes = '0' + minutes;

let requestedTimeKey =

(

parseInt(session.requestInfo.requestedAt.getHours(), 10) - 1

).toString() +

':' +

minutes;

let startDateKey =

(parseInt(session.startTime.getMonth(), 10) + 1).toString() +

'/' +

session.startTime.getDate() +

'/' +

(parseInt(session.startTime.getYear(), 10) % 100).toString();

minutes = session.startTime.getMinutes().toString();

if (minutes.length === 1) minutes = '0' + minutes;

let startTimeKey =

(parseInt(session.startTime.getHours(), 10) - 1).toString() +

':' +

minutes;

let endDateKey =

(parseInt(session.endTime.getMonth(), 10) + 1).toString() +

'/' +

session.endTime.getDate() +

'/' +

(parseInt(session.endTime.getYear(), 10) % 100).toString();

minutes = session.endTime.getMinutes().toString();

if (minutes.length === 1) minutes = '0' + minutes;

let endTimeKey =

(parseInt(session.endTime.getHours(), 10)).toString() +

':' +

minutes;

let duration = Math.abs(session.startTime - session.endTime) / 60000; //in minutes

let cancellationReason = session.reason;

let location = session.location;

let tutorKey = '';

await db.User.findOne({

\_id: mongoose.Types.ObjectId(session.serviceProvider)

}).then(tutor => {

if (tutor) {

tutorKey = tutor.AUID;

}

});

let studentKey = '';

await db.User.findOne({

\_id: mongoose.Types.ObjectId(session.requestInfo.requester)

}).then(student => {

if (student) {

studentKey = student.AUID;

}

});

let courseKey = '';

await db.Unit.findOne({

\_id: mongoose.Types.ObjectId(session.unit)

}).then(unit => {

if (unit) {

courseKey = unit.unitCode.replace(/\s/g, '');

}

});

records.push({

sessionNaturalKey,

sessionKey,

tutorKey,

studentKey,

courseKey,

requestedDateKey,

requestedTimeKey,

startDateKey,

startTimeKey,

endDateKey,

endTimeKey,

type,

status,

didAttend,

topic,

professor,

duration,

cancellationReason,

location

});

}).then(() => {

return factTableCSV

.writeRecords(records)

.then(() => {

console.log('...Done');

})

.then(() => {

return res.status(200).json({

status: '...Conversion Completed'

});

});

});

} catch (e) {

console.log(e);

return next({

status: 401,

message: 'Wrong credentials'

});

}

};

import csv

import numpy as np

import datetime

courses = list(csv.reader(open('courseDimension.csv', encoding='utf-8', errors='ignore')))

res = [["key", "courseCode"]]

oldCourse = ""

for i in range(0,len(courses)):

if(courses[i][2] != oldCourse):

res.append([i,courses[i][2]])

oldCourse=courses[i][2]

with open('courseMap.csv', mode='w') as file:

file\_writer = csv.writer(file, delimiter=',', quotechar='"', quoting=csv.QUOTE\_MINIMAL)

for row in res:

file\_writer.writerow(row)

import csv

import numpy as np

import datetime

fileName = input("File Name> ")

column = input("Column> ")

originFileName = input("Origin File Name> ")

originColumnName = input("Origin Column> ")

keyColumnInOriginFile = input("Key Column In Origin File> ")

isDate = input("Is Date (0:false,1:true)> ")

# fileName = "factTableTest.csv"

# column = "tutorKey"

# originFileName = "tutorDimensionTest.csv"

# originColumnName = "AUID"

# keyColumnInOriginFile = "studentKey"

factTable = None

originTable = None

def findCell(inputTable,columnName):

for i in range(0,len(inputTable[0])):

if(inputTable[0][i] == columnName):

return i

return -1

def readCsvFile(fileName):

return list(csv.reader(open(fileName, encoding='utf-8', errors='ignore')))

def writeCsvFile(fName, array2D):

with open(fName, mode='w') as file:

file\_writer = csv.writer(file, delimiter=',', quotechar='"', quoting=csv.QUOTE\_MINIMAL)

for row in array2D:

file\_writer.writerow(row)

factTable = readCsvFile(fileName)

originTable = readCsvFile(originFileName)

print("Finding Indexes..........", end='')

columnIndex = findCell(factTable, column)

columnOriginIndex = findCell(originTable, originColumnName)

keyColumnInOriginFileIndex = findCell(originTable, keyColumnInOriginFile)

print("Done")

print("\n\n'",column,"' Position: ",columnIndex)

print("'",originColumnName,"' Position: ",columnOriginIndex)

print("'",keyColumnInOriginFile,"' Position: ",keyColumnInOriginFileIndex ,"\n\n")

if(columnIndex == -1 or columnOriginIndex == -1 or keyColumnInOriginFileIndex == -1):

print("Column with label -1 couldn't be found")

else:

print("Processing...............", end='')

for i in range(1,len(factTable)):

key = factTable[i][columnIndex]

if(isDate == '1'):

key = datetime.datetime.strptime(key, "%m/%d/%y").strftime("%Y-%m-%d")

flag = False

for j in range(1,len(originTable)):

if(originTable[j][columnOriginIndex] == key):

factTable[i][columnIndex] = originTable[j][keyColumnInOriginFileIndex]

flag = True

break

if(not flag):

factTable[i][columnIndex] = ""

writeCsvFile(fileName,factTable)

print("Done")

print("[Success]: I'm done, thank you for using me <3")

"""

@author: Hamza Touhs

"""

import sys

import argparse

import csv

import datetime

class TimeDimensionBuilder(object):

"""

Build a TimeDimension CSV file for use in a data warehouse/data mart.

"""

def \_\_init\_\_(self):

start = datetime.datetime(2018, 1, 1, 0, 0, 0)

self.\_times = [start + datetime.timedelta(minutes=x) for x in range(0, 86400)]

def write\_to(self, file):

"""

Write all time records to the csv file specified in the argument.

"""

writer = csv.writer(file, delimiter=',', quoting=csv.QUOTE\_NONNUMERIC)

writer.writerow(TimeRecord.columns())

for time\_record in self.time\_records():

writer.writerow(time\_record.to\_list())

def time\_records(self):

"""

Return an iterator of TimeRecords

"""

for time in self.\_times:

yield TimeRecord(time)

class TimeRecord(object):

"""

Construct a record of a time dimension table representing a unique second.

"""

def \_\_init\_\_(self, time):

self.time\_key = int(datetime.datetime.strftime(time, '%H%M%S'))/100

self.military\_hour = int(datetime.datetime.strftime(time, '%H'))

self.civilian\_hour = int(datetime.datetime.strftime(time, '%I'))

self.minute = time.minute

self.am\_pm = datetime.datetime.strftime(time, '%p')

self.military\_time = datetime.datetime.strftime(time, '%H:%M')

self.civilian\_time = datetime.datetime.strftime(time, '%I:%M:%S %p')

self.time\_class = self.time\_class\_for(time.hour)

def time\_class\_for(self, hour):

"""

Answer the standard time window, e.g. morning, noon, afternoon,

evening, or night for the given hour

"""

if hour >= 0 and hour < 6:

result = "Night"

elif hour >= 6 and hour < 12:

result = "Morning"

elif hour >= 12 and hour < 13:

result = "Noon"

elif hour >= 13 and hour < 17:

result = "Afternoon"

elif hour >= 17 and hour < 20:

result = "Evening"

elif hour >= 20 and hour <= 24:

result = "Night"

return result

def to\_list(self):

"""

Return the TimeRecord as a list.

"""

return [self.time\_key,

self.military\_hour,

self.civilian\_hour,

self.minute,

self.am\_pm,

self.military\_time,

self.civilian\_time,

self.time\_class]

@staticmethod

def columns():

"""

Return the column headings for all TimeRecords

"""

return ['time\_key', 'military\_hour', 'civilian\_hour', 'minute', 'am\_pm', 'military\_time', 'civilian\_time', 'time\_class']

def main():

"""

Build the time dimension sending contents either to stdout or a file if specified in stdin.

"""

parser = argparse.ArgumentParser(

description="Build a Time Dimension for use in a Data Warehouse or Data Mart. "

+ "The output will be written to stdout unless a CSV file "

+ "is specified with the -f/--file option.")

parser.add\_argument("-f",

"--file",

dest="filename",

help="write output to a CSV file",

metavar="FILE")

args = parser.parse\_args()

builder = TimeDimensionBuilder()

if args.filename is None:

builder.write\_to(sys.stdout)

else:

with open(args.filename, 'w', newline='') as dim\_time\_file:

builder.write\_to(dim\_time\_file)

print("Time Dimension written to " + args.filename)

if \_\_name\_\_ == '\_\_main\_\_':

main()

import sys

import argparse

import datetime

import calendar

import math

import csv

import numpy

import holidays

# Set Calendar's first day of the week to Sunday

calendar.setfirstweekday(6)

class DateDimensionBuilder(object):

"""

Builds a date dimension, AKA a calendar dimension, for use in data warehousing.

"""

def \_\_init\_\_(self, start\_date, end\_date):

"""

Constructs a new DateDimensionBuilder starting at start\_date and ending at end\_date.

"""

self.start\_date = start\_date

self.end\_date = end\_date

def date\_records(self):

"""

Returns a generator to iterate over each date record of the calendar.

"""

for ordinal in range(self.start\_date.toordinal(), self.end\_date.toordinal()+1):

yield DateRecord(datetime.date.fromordinal(ordinal))

def write\_to(self, file, columnsonly=False):

"""

Write all date records to a csv file.

"""

writer = csv.writer(file, delimiter=',', quoting=csv.QUOTE\_NONNUMERIC, escapechar='\\')

writer.writerow(DateRecord.columns())

if columnsonly is False:

for date\_record in self.date\_records():

writer.writerow(date\_record.to\_list())

class DateRecord(object):

"""

Represents a date record in a date dimension.

"""

def \_\_init\_\_(self, date):

"""

Construct a new date record for the given date.

"""

self.\_current\_date = date

@property

def date\_key(self):

"""

Retruns an int representation of the date suitable

for use as a primary key in a date dimension table

"""

return int(self.\_current\_date.strftime("%Y%m%d"))

@property

def date(self):

"""

Returns a character representation of the date in

YYYY-MM-DD format for importing into a date dimension

table as a date data type

"""

return self.\_current\_date.strftime("%Y-%m-%d")

###

# Seasons

@property

def season(self):

"""

Returns the season, i.e. Spring, Summer, Fall, or Winter.

"""

if self.day\_of\_year in range(80, 172):

season = 'Spring'

elif self.day\_of\_year in range(172, 264):

season = 'Summer'

elif self.day\_of\_year in range(264, 355):

season = 'Fall'

else:

season = 'Winter'

return season

###

# Quarters

@property

def quarter(self):

"""

Returns the current quarter as an int, 1, 2, 3, or 4.

"""

return math.ceil(self.\_current\_date.month/3)

@property

def quarter\_name(self):

"""

Returns the quarter name, e.g. First, Second, Third, or Fourth

"""

quarters = {1: "First", 2: "Second", 3: "Third", 4: "Fourth"}

return quarters[self.quarter]

@property

def quarter\_short\_name(self):

"""

Returns the short name of the quarter, i.e. Q1, Q2, Q3, or Q4

"""

quarters = {1: "Q1", 2: "Q2", 3: "Q3", 4: "Q4"}

return quarters[self.quarter]

###

# Year Methods

@property

def year(self):

"""

Returns the year as an int.

"""

return int(self.\_current\_date.strftime("%Y"))

@property

def fiscal\_year(self):

"""

Returns the fiscal year if different from the calendary year. If

not different, it returns the calendar year.

"""

#TODO: Implement this.

return None

@property

def year\_and\_quarter(self):

"""

Returns the YYYY/Qn

"""

return str(self.year) + "/" + self.quarter\_short\_name

@property

def year\_and\_month(self):

"""

Returns the year and month as YYYY/MM.

"""

return self.\_current\_date.strftime("%Y/%m")

@property

def year\_and\_month\_abbrev(self):

"""

Returns the year and month short name, e.g. YYYY/Oct.

"""

return str(self.year) + "/" + self.month\_abbrev

###

# Month Methods

@property

def month\_number(self):

"""

Returns an int representing the calendar month.

"""

return int(self.\_current\_date.strftime("%m"))

@property

def month\_name(self):

"""

Returns the long month name, e.g. January, December.

"""

return self.\_current\_date.strftime("%B")

@property

def month\_abbrev(self):

"""

Returns an abbreviated month name, e.g. Jan, Dec.

"""

return self.\_current\_date.strftime("%b")

@property

def month\_end\_flag(self):

"""

Answers true if the day is the last day of the month.

"""

mrange = calendar.monthrange(self.year, self.month\_number)

if mrange[1] == self.day\_of\_month:

return True

else:

return False

@property

def fiscal\_month\_number(self):

"""

Answers the month number for the fiscal calendar. If no fiscal start

month is set, this answers with month\_number().

"""

#TODO: Implement this.

return None

###

# Week Methods

@property

def week\_num\_in\_year(self):

"""

Return the week's number in the calendar year.

"""

week = int(self.\_current\_date.strftime("%U"))

if week < 1:

week = 1

return week

@property

def week\_num\_in\_month(self):

"""

Returns the week number in the month.

"""

mcalendar = calendar.monthcalendar(self.\_current\_date.year, self.\_current\_date.month)

matrix = numpy.array(mcalendar)

match = numpy.where(matrix == self.day\_of\_month)

week\_of\_month = match[0][0] + 1

return week\_of\_month

@property

def week\_begin\_date\_key(self):

"""

Returns the Sunday of the week for the current date as an int for use as a key

"""

week\_begin\_date = datetime.datetime.strptime(self.week\_begin\_date, "%Y-%m-%d")

return week\_begin\_date.strftime("%Y%m%d")

@property

def week\_begin\_date(self):

"""

Returns the Sunday of the week for the current date as a YYYY-MM-DD formatted string

"""

if self.day\_name == 'Sunday':

return self.date

else:

return (self.\_current\_date

- datetime.timedelta(days=self.day\_of\_week)).strftime("%Y-%m-%d")

###

# Day Methods

@property

def day\_of\_year(self):

"""

Returns the day of the year, 1..365.

"""

return self.\_current\_date.timetuple().tm\_yday

@property

def day\_of\_month(self):

"""

Returns an int representing the day of the month.

"""

return int(self.\_current\_date.strftime("%d"))

@property

def day\_of\_week(self):

"""

Returns the day's number, with 1 being Sunday and 7 being Saturday.

"""

return ((self.\_current\_date.weekday() + 1) % 7) + 1

@property

def day\_name(self):

"""

Answers the day's name in long form, e.g. Monday

"""

return self.\_current\_date.strftime("%A")

@property

def day\_name\_abbrev(self):

"""

Answers the day's name abbreveiated, e.g. Mon

"""

return self.\_current\_date.strftime("%a")

@property

def same\_day\_previous\_year\_key(self):

"""

Returns an int representation of this date one year ago, e.g. YYYYMMDD.

If the present day is a leap day, the prior day of the previous year

will be returned.

"""

return datetime.datetime.strptime(self.same\_day\_previous\_year,

"%Y-%m-%d").strftime("%Y%m%d")

@property

def same\_day\_previous\_year(self):

"""

YYYY-MM-DD formatted version of same\_day\_previous\_year\_key

"""

year = self.\_current\_date.year -1

month = self.\_current\_date.month

day = self.\_current\_date.day

try:

one\_year\_ago = datetime.date(year, month, day)

except ValueError:

# Error due to leap year. Use the date for the

# previous year less a day.

one\_year\_ago = datetime.date(year, month, day-1)

return one\_year\_ago.strftime("%Y-%m-%d")

@property

def is\_weekday(self):

"""

Answers True if not Saturday or Sunday.

"""

if self.day\_name\_abbrev == 'Sat' or self.day\_name\_abbrev == 'Sun':

return False

else:

return True

@property

def is\_weekend(self):

"""

Answers True if Saturday or Sunday.

"""

return not self.is\_weekday

###

# Holidays

@property

def is\_holiday(self):

"""

True if date is a standard US holiday.

"""

us\_holidays = holidays.UnitedStates()

return self.\_current\_date in us\_holidays

@property

def holiday\_name(self):

"""

Holiday name if there is one, None if not.

"""

us\_holidays = holidays.UnitedStates()

return us\_holidays.get(self.\_current\_date.strftime("%Y-%m-%d"))

def to\_list(self):

"""

Returns a list representation of the record.

"""

return [getattr(self, value) for value in dir(self)

if value not in ("to\_list", "columns") and value[0] is not "\_"]

@staticmethod

def columns():

"""

Returns the column names corresponding to to\_list().

"""

return [value for value in dir(DateRecord)

if value not in ("to\_list", "columns") and value[0] is not "\_"]

def main():

"""

Build the date dimension sending contents either to stdout or a file if specified in stdin.

"""

parser = argparse.ArgumentParser(

description="Build a Date Dimension for use in a Data Warehouse or Data Mart. "

+ "The output will be written to stdout unless a CSV file "

+ "is specified with the -f/--file option.")

parser.add\_argument("-f",

"--file",

dest="filename",

help="write output to a CSV file.",

metavar="FILE")

parser.add\_argument("-s",

"--startdate",

dest="startdate",

#type="string",

default="1/1/1850",

help="starting date of the date dimension. Default is 1/1/1850.",

metavar="DATE")

parser.add\_argument("-e",

"--enddate",

dest="enddate",

#type="string",

default="12/31/2050",

help="ending date for the date date dimesnion. Default is 12/31/2050.",

metavar="DATE")

parser.add\_argument("-c",

"--columnnamesonly",

dest="columnsonly",

action='store\_true',

help="output column names only.")

args = parser.parse\_args()

try:

start\_date = datetime.datetime.strptime(args.startdate, "%m/%d/%Y")

end\_date = datetime.datetime.strptime(args.enddate, "%m/%d/%Y")

builder = DateDimensionBuilder(start\_date, end\_date)

if args.columnsonly is True:

builder.write\_to(sys.stdout, columnsonly=True)

elif args.filename is None:

builder.write\_to(sys.stdout)

else:

with open(args.filename, 'w', newline='') as dim\_date\_file:

builder.write\_to(dim\_date\_file)

print("Date Dimension written to " + args.filename)

except ValueError as err:

print("Unable to parse supplied dates "

+ args.startdate

+ ", and "

+ args.enddate

+ ". Error detail: "

+ str(err))

parser.print\_help()

if \_\_name\_\_ == '\_\_main\_\_':

main()

Not Important:

Sudo sustemctl enable ssh

Sudo systemctl start ssh

Wget <https://archive.apache.org/dist/hadoop/common/hadoop-2.7.3/>

Start-dfs.sh jps

hadoop fs -put ~/Documents/studentDimension.csv hdfs:/dimensions