Freie Universität Berlin

Prof. Dr. Agnès Voisard, Nicolas Lehmann

Datenbank Systeme, SoeSe 2017

Project 2.Iteration

TutorIn: Hoffman Christian
Tutorium 3, Gruppe 22

Ingrid Tchilibou, Emil Milanov, Boyan Hristov

2. Juni 2017

Allgemein

Link zum Projekt: https://github.com/gancia-kiss/dbs_projekt

1.Aufgabe: Datenbankschema erstellen

Link zum *.sql* Datei:
https://github.com/gancia-kiss/dbs_projekt/blob/master/DatabaseDump.sql

Die letzte Iteration hatten wir ein paar 'Count' Attributen addiert, wir haben aber uns später entschieden, dass wir sie nicht brauchen. Diese Iteration, aber wir haben bemerkt, dass wir eigenltich diese Attribute brauchen.

Zusätzlich haben wir auch ein Count Attribut, damit wir aufzählen können, welche Paar von Hashtags am häufigsten vorkommt.

Wir haben uns entschieden selber IDs von Tweets zu erstellten und zwar mit Python UUID. Das Ergebnis kann größere Integer sein, deswegen haben wir als TWEET.ID 'bigint' als Datentyp benutzt.

Bei uns ist der Inhalt der Hashtags die primäre Schlüssel. Wir haben auch als Erinerrung der Attribut 'textlowercase' genannt, um zu wissen, nur kleine Buchstaben zu verwenden.

```
pg_dump election > DatabaseDump.sql      # Dump Datei erstellen

psql election < DatabaseDump.sql         # Datenbank vom .sql Datei importieren
```

2.Aufgabe: Datenbereinigung

Link zum Datenreinigungsprogramm:
https://github.com/gancia-kiss/dbs_projekt/blob/master/programs/cleaner.py

Nach tiefere Datenanalyse haben wir zwei Hauptprobleme identifiziert:

1. Manche Tweets sind abgeschnitten und ein Teil von denen sind 'Truncated'. Wir wussten aber nicht, wie wir schnell die Tweets rekonstruiren können. Deswegen haben wir die gezählt und haben festgestellt, dass nur 2% der Tweets abgeschnitten sind. Es wäre für uns am leichtesten, alle solchen Tweets einfach zu löschen.

Wir haben bemerkt, dass alle abgeschnittenen Tweets auf '...' dann einen Link endeten. Es ist aber herausgekommen, dass '...' in solchen Tweets nur einen Unicode Zeichen war. Deswegen müssten wir im Python Program überprüfen, ob der Zeichen $u'\backslash u2026'$ im Körper des Tweets vorkommt.

2. Die vorgegebene Einstellungen der Libre Office Calculator hat versucht mit Codierung 'UTF-8' geöffnet. Dann gab es aber Probleme mit manchen Symbolen. Apostrophe, manche Sonderzeichen und vermutlich auch Emojis wurden nicht angezeigt. Nach kurze Analyse haben wir festgestellt, dass wenn wir die Tabelle mit Codierung 'Windows 1252' öffnen, werden Apostrophen und wichtige Sonderzeichen normal dargestellt, und Emojis und nicht für den Datensatz relevante Zeichen wurden gelöscht.

Die Python Program war sehr einfach. Wir haben die Standartbibliothek 'csv' benutzt um ein csv_writer und csv_reader zu erzeugen. Wir haben dann die .csv Datei gelesen, und Zeile für Zeile haben wir geprüft ob der Tweet Truncated ist, oder abgeschnitten ist. Wenn nicht, haben wir den Tweet in einer anderen Datei *test.csv* gespeichert.

3.Aufgabe: Datenimport

- Was & Wie?

  Wir haben hier versucht zu beschreiben, wie das Programm für die Datenimportierung funktioniert. Den Code kann man ganz am Ende der Dokummentation finden. Wir haben versucht eine gute Modularisierung zu schaffen und gut lesbaren Code zu schreiben, deswegen sollen die hier nicht erklärte Funktionalitäten gut im Code verfolgbar sein.

- Grobe Struktur des Programms

  Unser server schickt on StartUp ein Request für alle Hashtags zu dem DBS. Falls keine zurückgelifert werden, müssen wir die Daten importieren.

  Alles fängt bei TableParser.py an. Die schon bereinigte Datei data-cleaned.csv wird gelesen und es werden für jede Zeile in der Tabelle dynamisch die dazugehörige Tabellen als Dictionaries erstellt. Diese Module sorgt dafür, dass die dazugehörige Funktionen in Utils.py und Extractor.py aufgerufen werden und fasst dann alle Ergebnisse in Wörterbücher, die von dem DBController.py weiter bearbeitet werden. Hier wird definiert was in jede Tabelle reinkommt, es werden aber keine große Bearbeitungen gemacht.

  Für die Bearbeitung der Daten ist Extractor.py antwortlich. Diese Module macht alle Operationen, die Daten aus der Tabelle zu 'derived attributes' konvertieren. Hier werden die Hashtags aus dem Text genommen, das Datum in Woche umgewandelt und das Rating für ein Tweet kalkuliert.

  Als wir schon erkennen, gibt es viele Abhängigkeiten zwischen das Programm, die Tabellen in den Datenbank und die CSV Dateien. Deswegen haben wir die Klasse Contract.py hergestellt, wo alle Strings definiert werden. Diese Klasse haben wir als eine Art von Struct benutzt, da es in Python keine Structs gibt.

Es gibt einige Umformungen, die in der Informatik sehr bekannt sind und ganz abstrakt sind, wie z.B. die Erzeugung von allen Paaren ohne Wiederholungen (2-Sets). Solche Methoden haben wir in der Module Utils.py definiert.

Am wichtigsten ist sicherlich die Module DBController.py. Hier wird die Verbindung mit dem DBS hergestellt und die ganze Kommunikation mit DBS implementiert. Es werden hier auch alle Seiteneffekten und Datenlogiken behandelt, wie z.B. inkrementierung von Count Attributen.

- Feststellung von in einem Tweet vorhandene Hashtags

  Worttrennung ist eine schwierige Operation, als wir in Softwaretechnik gelernt haben. Deswegen sind wir davon ausgegangen, was wir bereits sicherlich kennen - nach dem Symbol '#" sind alle Buchstaben und Ziffern Teil eines Hashtags, alles anderes beendet das Hashtag. Deswegen haben wir, so lange noch '#" Symbol weiter in dem String zu finden ist, den String von '#' bis nach einem Symbol das nicht in 0-9, a-z und A-Z zu finden ist als Hashtag genommen und weiter iterativ nach '#" gesucht. Mehr kann man in Extractor::extractHashtags() erfahren

- Woche bestimmen

  Wir haben die bereits in den Bibliotheken 'time' und 'datetime' Funktionalität benutz, um erstmal ein Datenobjekt aus dem String zu erzeugen und danach davon den 'isocalender' zu erzeugen, was ein Trippel vom Jahr, Wochennummer und Tag ist. Dann haben wir den Tag entsprechend zu 1 (für 'Week::StartDate') oder zu 7 gesetzt und davon wieder ein Datum Objekt erzeugt.

- Rating bestimmen

  Wie schon in der 1. Projektiteration erleutert, haben wir die Retweets und Favourites benutzt, um ein Float für den Rating zu erzeugen. Da alle Einträge Retweets und Favourites definiert haben, hatten wir hier keine Schwierigkeiten.

- Count setzen

  Nachdem wir jede Zeile geparsed haben in TableParse.py, haben wir für den neuen Eintrag in Datenbank immer Count auf 1 gesetzt. Wir haben aber die Ausführung von dem SQL Expression in einem try-except (try-catch in Python) gewrapped. Falls eine Fehler erzeugt wird, haben wir überprüft, ob das konkrete Eintrag ein 'Count' Attribut hat und in dem Fall ein UPDATE SQL Expression ausgeführt wo 'Count' hochgesetzt wird. An der Stelle haben wir auch überprüft, ob die erzeugte Fehlermeldung nach dem Einfügung einer Woche erzeugt wurde und in dem Fall nichts gemacht (auch keine logs), da eine Woche nicht doppelt vorkommen soll und kein 'Count' hat.

  Wir haben uns entschieden, keine SQL Expressions für Überprüfung auszuführen, da wir so uns eine SQL Ausführung sparen können, falls den entsprechenden Hashtag / Woche / Relation noch nicht erzeugt wurde. Sonst können wir trotzdem ein wieteres SQL Expression ausführen um Count hochzusetzen. SQL kann schon solche Situation gut behandeln, deswegen können wir uns die Überprüfungen sparen.

- Muster Abfragen

  Wir haben schon eine Abfrage für die relevantesten Tweets implementiert, damit wir zeigen, dass die Daten erfolgreich implementiert wurden. Als schon für die 1. Projektiteration erwähnt, kann man alle da als Beispiel gegebene Abfragen gut mit unserem Datenmodell implementieren.

Wan mann in dem Browser zu http://127.0.0.1:5234/topTweets navigiert, kann man schon die 10 relevantesten Tweets sehen (noch nicht schön visualisiert, aber irgendwie).

4.Aufgabe: Webserver

Link zum Webserver:
https://github.com/gancia-kiss/dbs_projekt/tree/master/programs/server

Wir haben uns entschieden selbe einen Webserver zu entwickeln und zwar wollten wir die 'Flask' Framework von Python zu verwenden. Die Installation war sehr einfach, und es ist sehr einfach ein kurzes Programm zu entwickeln.

```
from flask import Flask

app = Flask(__name__)
port = 5234
host = '127.0.0.1'

@app.route('/')
def index():
    return 'Hello world'

if __name__ == '__main__':
    app.run(debug=True, host=host, port=port)
```

Das Programm erzeugt einen einfachen Webserver auf Port 5234. Wenn man die Seite *localhost:5234* öffnet, dann sieht man unformatiert 'Hello world'.

Es wäre sehr einfach das zu erweitern. Wir können .html Dateis im selben ordner legen, und dann neue Wege mit *@app.route()* definieren. Vielleicht werden wir die Programme für Hashtag-analyse in den Server einbauen, damit wir dynamisch den Inhalt anpassen können.

CODE

- server.py

```
from flask import Flask
from cleaner import cleanData
from DBController import DBController
from TableParser import TableParser

app = Flask(__name__)
port = 5234
host = '127.0.0.1'

@app.route('/')
def index():
 return 'Hello world'

@app.route('/topTweets')
def getTopTweets():
 topTweets = dbController.getTopTweets()
 return 'Top tweets: <br><br>' + '<br><br>'.join(str(e) for e in topTweets)

if __name__ == '__main__':

 dbController = DBController()
```

```
filled = dbController.checkFilled()

if not filled:
 cleanData()
 TableParser.parseTables()
else:
 print('Data already imported :)')

app.run(debug=True, host=host, port=port)
```

- TableParser.py

```
import csv
from DBController import DBController
from Extractor import Extractor
from Utils import Utils
from Contract import Contract

class TableParser:

 @staticmethod
 def getTweetFromEntry(entry):

  return {

   Contract.ADD_TO_TABLE_KEY: Contract.TABLE_TWEET,
   Contract.ID_COLUMN: Utils.getRandom8ByteInt(),
   Contract.AUTHOR_COLUMN: entry[Contract.HANDLE_ENTRY],
   Contract.TEXT_COLUMN: entry[Contract.TEXT_ENTRY],
   Contract.TIME_COLUMN: Extractor.extractTime(entry),
   Contract.RATING_COLUMN: Extractor.calculateRatingForTweet(entry)
  }

 @staticmethod
 def getHashTagsFromHTTexts(hashtagTexts):
  hashtags = []

  for ht in hashtagTexts:
   hashtags.append(
    {
     Contract.ADD_TO_TABLE_KEY: Contract.TABLE_HASHTAG,
     Contract.TEXT_LOWER_CASE_COLUMN: ht,
     Contract.COUNT_COLUMN: 1
    }
   )

  return hashtags

 @staticmethod
 def getWeekFromEntry(entry):
  return {
   Contract.ADD_TO_TABLE_KEY: Contract.TABLE_WEEK,
   Contract.START_DATE_COLUMN: Extractor.getWeekStart(entry),
   Contract.END_DATE_COLUMN: Extractor.getWeekEnd(entry)
  }

 @staticmethod
 def getUsedIns(week, hashtags):

  usedIns = []
  startDate = week[Contract.START_DATE_COLUMN]

  for ht in hashtags:
   usedIns.append(
    {
     Contract.ADD_TO_TABLE_KEY: Contract.TABLE_USED_IN,
```

```
      Contract.HASHTAG_TEXT_COLUMN: ht,
      Contract.WEEK_START_DATE_COLUMN: startDate,
      Contract.COUNT_COLUMN: 1
     }
    )

  return usedIns


@staticmethod
def getPostedIn(week, tweet):
  return {
   Contract.ADD_TO_TABLE_KEY: Contract.TABLE_POSTED_IN,
   Contract.TWEET_ID_COLUMN: tweet[Contract.ID_COLUMN],
   Contract.WEEK_START_DATE_COLUMN: week[Contract.START_DATE_COLUMN]
  }

@staticmethod
def getContains(tweet, hashtags):

  contains = []
  tweetid = tweet[Contract.ID_COLUMN]

  for ht in hashtags:
   contains.append(
    {
     Contract.ADD_TO_TABLE_KEY: Contract.TABLE_CONTAINS,
     Contract.HASHTAG_TEXT_COLUMN: ht,
     Contract.TWEET_ID_COLUMN: tweetid
    }
   )

  return contains

@staticmethod
def getUsedTogetherWithFromHTTexts(hashtagTexts):

  pairs = Utils.getUniquePairs(hashtagTexts)
  usedTogetherWiths = []

  for pair in pairs:
   usedTogetherWiths.append(
    {
     Contract.ADD_TO_TABLE_KEY: Contract.TABLE_USED_TOGETHER_WITH,
     Contract.PRIMARY_HASHTAG_COLUMN: pair[0],
     Contract.TOGETHER_WITH_HASHTAG_COLUMN: pair[1],
     Contract.COUNT_COLUMN: 1
    }
   )

  return usedTogetherWiths

@staticmethod
def parseRow(row, dbController):

  hashtagTexts = Extractor.extractHashtags(row)

  tweet = TableParser.getTweetFromEntry(row)
  week = TableParser.getWeekFromEntry(row)
  hashtags = TableParser.getHashTagsFromHTTexts(hashtagTexts)

  usedIns = TableParser.getUsedIns(week, hashtagTexts)
  postedIn = TableParser.getPostedIn(week, tweet)
  contains = TableParser.getContains(tweet, hashtagTexts)
  usedTogetherWiths = TableParser.getUsedTogetherWithFromHTTexts(hashtagTexts)

  dbController.addMultiple(tweet, week, hashtags, usedIns, postedIn, contains,
    usedTogetherWiths)
```

```
123  @staticmethod
124  def parseTables():
125   dbController = DBController()
127   filepath = Contract.CSV_CLEAN
128   csvfile = open(filepath, 'r', encoding='cp1252')
129   csv_reader = csv.DictReader(csvfile, delimiter=';', quotechar='"')
131   for idx, row in enumerate(csv_reader):
132       TableParser.parseRow(row, dbController)
133       print("Parsed {} rows...".format(idx))
135   dbController.close()
```

- Contract.py

```
1  class Contract:
3   CSV_INITIAL = 'american-election-tweets.csv'
4   CSV_CLEAN = 'data-cleaned.csv'
6   TABLE_WEEK = 'week'
7   TABLE_TWEET = 'tweet'
8   TABLE_HASHTAG = 'hashtag'
9   TABLE_USED_IN = 'usedin'
10  TABLE_POSTED_IN = 'postedin'
11  TABLE_USED_TOGETHER_WITH = 'usedtogetherwith'
12  TABLE_CONTAINS = 'contains'
14  ADD_TO_TABLE_KEY = 'addToTable'
16  COUNT_COLUMN = 'count'
17  ID_COLUMN = 'id'
18  TWEET_ID_COLUMN = 'tweetid'
19  AUTHOR_COLUMN = 'author'
20  TEXT_COLUMN = 'text'
21  START_DATE_COLUMN = 'startdate'
22  END_DATE_COLUMN = 'enddate'
23  WEEK_START_DATE_COLUMN = 'weekstartdate'
24  TIME_COLUMN = 'time'
25  RATING_COLUMN = 'rating'
26  PRIMARY_HASHTAG_COLUMN = 'primaryhashtag'
27  TOGETHER_WITH_HASHTAG_COLUMN = 'togetherwithhashtag'
28  TEXT_LOWER_CASE_COLUMN = 'textlowercase'
29  HASHTAG_TEXT_COLUMN = 'hashtagtext'
31  TEXT_ENTRY = TEXT_COLUMN
32  HANDLE_ENTRY = 'handle'
34  IGNORE_DUPLICATES_IN_TABLES = [TABLE_WEEK]
```

- Extractor.py

```
1  import time
2  from datetime import datetime
3  import re
5  class Extractor:
7   RATING_WEIGHT = {
8    "retweetCount": 0.8,
9    "favouriteCount": 1.2
10  }
```

```
    TIME_FORMAT = '%Y-%m-%dT%H:%M:%S'

    @staticmethod
    def calculateRatingForTweet(entry):
     retweetCount = float(entry['retweet_count'])
     favouriteCount = float(entry['retweet_count'])

     return retweetCount*Extractor.RATING_WEIGHT['retweetCount'] + retweetCount*Extractor
       .RATING_WEIGHT['favouriteCount']

    @staticmethod
    def extractTime(entry):
     time = entry['time']

     return datetime.strptime(time, Extractor.TIME_FORMAT)

    @staticmethod
    def getNthDayOfWeek(entry, n):
     time = Extractor.extractTime(entry)
     isoCalender = time.date().isocalendar()

     return datetime.strptime('{0} {1} {2}'.format(isoCalender[0], isoCalender[1], n), '%
       G %V %u').date()

    @staticmethod
    def getWeekStart(entry):
     return Extractor.getNthDayOfWeek(entry, 1)

    @staticmethod
    def getWeekEnd(entry):
     return Extractor.getNthDayOfWeek(entry, 7)

    @staticmethod
    def extractHashtags(entry):
     text = entry['text']
     searchedUntil = 0
     hashtags = []

     try:
      while (True):

       crntHashtagStart = text.index('#', searchedUntil) + 1

       try:
        crntHashtagEnd = re.search('[^a-zA-Z0-9]', text[(crntHashtagStart):]).start() +
       crntHashtagStart
       except AttributeError as atrErr:
        crntHashtagEnd = len(text) - 1

       crntHashtag = text[crntHashtagStart:crntHashtagEnd].lower()
       searchedUntil = crntHashtagEnd

       if not crntHashtag in hashtags:
        hashtags.append(crntHashtag)

     except ValueError as err:
      # no more hashtags found, we are ready
      pass

     return hashtags
```

- Utils.py

```
    import uuid
```

```
class Utils:

 @staticmethod
 def getUniquePairs(arr):

  if len(arr) < 2:
   return []

  pairs = []

  for i, itemA in enumerate(arr):

   for j, itemB in enumerate(arr[(i+1):]):

    crntPair = [itemA, itemB]
    crntPair.sort()
    pairs.append(crntPair)

  return pairs

 @staticmethod
 def getRandom8ByteInt():
  return uuid.uuid4().int & (1 << (8*8 - 1))-1

 @staticmethod
 def getValuesFromDict(entriesDict):
  sqlValuesArr = []
  for key in entriesDict:
   escapedString = str(entriesDict[key]).replace('`', '\\`').replace('\'', '\\`')
   sqlValuesArr.append('\'{0}\''.format( escapedString ))
  return ', '.join(sqlValuesArr)
```

- DBController.py

```
import psycopg2
import json
from Contract import Contract
from Utils import Utils


class DBController:

 dbUser = "testuser"
 dbName = "Election"
 host = "localhost"
 dbUserPassword = "testpass"

 connectionStringTemplate = "dbname='{0}' user='{1}' host='{2}' password='{3}'"
 connectionString = connectionStringTemplate.format(dbName, dbUser, host,
    dbUserPassword)

 def __init__(self):
  print("Connected to DB!")
  self.connection = psycopg2.connect(DBController.connectionString)
  self.cursor = self.connection.cursor()

 def addToDB(self, table, entriesDict):

  sqlInsertExpr = "INSERT INTO {0} ({1}) VALUES ({2})"
  sqlInsertCommand = sqlInsertExpr.format(table, ",".join(list(entriesDict.keys())),
    Utils.getValuesFromDict(entriesDict))

  try:
   self.cursor.execute(sqlInsertCommand)
  except Exception as err:
```

```
    if Contract.COUNT_COLUMN in entriesDict:
     entriesDictCopy = entriesDict.copy()
     del entriesDictCopy[Contract.COUNT_COLUMN]

     sqlUpdateExpr = "UPDATE {0} SET {1}={1}+1 WHERE ({2})"
     sqlUpdateCommand = sqlUpdateExpr.format(table, Contract.COUNT_COLUMN, DBController
     .getWhereConditionsForUpdate(entriesDictCopy))
     self.connection.commit()
     self.cursor.execute(sqlUpdateCommand)

    elif table in Contract.IGNORE_DUPLICATES_IN_TABLES:
     pass
    else:
     print("Something went wrong while adding new DB entry to table {0}: {1}".format(
     table, err))

  self.connection.commit()

 def close(self):
  self.connection.close
  print("DB connection closed!")

 def handleAddTable(self, data):
  table = data[Contract.ADD_TO_TABLE_KEY]
  del data[Contract.ADD_TO_TABLE_KEY]
  self.addToDB(table, data)
  # try:
  #  self.addToDB(table, data)
  # except Exception as err:
  #  print("Something went wrong: " + str(err))


 def addMultiple(self, *tableLists):

  for tableList in tableLists:

   if isinstance(tableList, list):

    for table in tableList:
     self.handleAddTable(table)
   else:
    self.handleAddTable(tableList)

 def checkFilled(self):

  self.cursor.execute("SELECT * FROM {0}".format(Contract.TABLE_WEEK))
  weeks = self.cursor.fetchall()

  return len(weeks) > 0

 def getTopTweets(self):
  self.cursor.execute("SELECT * FROM {0} ORDER BY {1} LIMIT 10".format(Contract.
    TABLE_TWEET, Contract.RATING_COLUMN))
  return self.cursor.fetchall()

 @staticmethod
 def getWhereConditionsForUpdate(columnsDict):

  conditions = []

  for key in columnsDict:
   conditions.append(key + '=' + "'{0}'".format(columnsDict[key]))

  return ' AND '.join(conditions)
```