```python
# txt_to_db_GTFS_IDF.py

import numpy as np
import codecs
import sqlite3 as sql
import matplotlib.pyplot as pl
import json
import math as m

stops=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\IDFM_gtfs\\stops.txt', 'r', encoding='utf-8')
trips=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\IDFM_gtfs\\trips.txt', 'r', encoding='utf-8')
stop_times=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\IDFM_gtfs\\stop_times.txt', 'r', encoding='utf-8')
routes=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\IDFM_gtfs\\routes.txt', 'r', encoding='utf-8')
transfers=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\IDFM_gtfs\\transfers.txt', 'r', encoding='utf-8')
agency=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\IDFM_gtfs\\agency.txt', 'r', encoding='utf-8')
calendar=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\IDFM_gtfs\\calendar.txt', 'r', encoding='utf-8')
calendar_dates=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\IDFM_gtfs\\calendar_dates.txt', 'r', encoding='utf-8')

conn=sql.connect(r"F:\informatique\TIPE\database\produit exploitable\GTFS.db")
c = conn.cursor()
erreur=[]


## Remplire de données exploitables la DB
def modele(nom_table_string, fonction, fichier, delete):
    num_lig=-1
    print(nom_table_string)
    if delete:
        c.execute('DELETE FROM {}'.format(nom_table_string))
        conn.commit()
    for line in fichier:
        if num_lig!=-1:  #la première ligne
            try:
                fonction(num_lig, line)
            except Exception:
                erreur.append([line,nom_table_string])
        num_lig+=1
        if (num_lig%10000)==0:
            print(num_lig)
    conn.commit()
    print(len(erreur), 'erreurs \n')

#il y a parfois des virgules dans les "stop_name"
def traitement_stops(num_lig,line):
    data=line.split('"')
    stop_name=traitement_accents_tirets(data[1])
    data2=[]
    fin_de_ligne=''
    for i in range(2, len(data)):
        fin_de_ligne+=data[i]
    for subline in (data[0], fin_de_ligne):
        data2.extend(subline.split(','))
    c.execute('insert into stops(id, stop_id, stop_name, stop_lat, stop_lon) values (?,?,?,?,?)', (num_lig, data2[0], stop_name, data2[4], data2[5]))


def traitement_stop_times(num_lig,line):
```

```
    data=line.split(',')
    c.execute('insert into stop_times(trip_id, departure_time, stop_id, stop_sequence) values (?,?,?,?)', (data[0], data[1], data[3], int(data[4])))

def traitement_trips(num_lig,line):
    data=line.split(',')
    c.execute('insert into trips(route_id, trip_id, trip_short_name, service_id) values (?,?,?,?)', (data[0], data[2], traitement_guillemets(data[3]), data[1]))

def traitement_routes(num_lig,line):
    data=line.split(',')
    [name1, name2]=data[2:4]
    c.execute('insert into routes(route_id, agency_id, route_short_name, route_long_name, route_type) values (?,?,?,?,?)', (data[0], data[1],
traitement_guillemets(name1), traitement_guillemets(name2), int(data[5])))

def traitement_transfers(num_lig,line):
    data=line.split(',')
    c.execute('insert into transfers(from_stop_id, to_stop_id, transfer_time) values (?,?,?)', (data[0], data[1], data[3]))

def traitement_agency(num_lig,line):
    data=line.split(',')
    c.execute('insert into agency (agency_id, agency_name) values (?,?)', (data[0], traitement_guillemets(data[1])))

def traitement_calendar(num_lig,line):
    data=line.split(',')
    c.execute('insert into calendar (service_id, start_date, stop_date) values (?,?,?)', (data[0], data[8], data[9]))

def traitement_calendar_dates(num_lig,line):
    data=line.split(',')
    c.execute('insert into calendar_dates(service_id, date, exception_type) values (?,?,?)', (data[0], data[1], data[2]))

def traitement_guillemets(string):
    n=len(string)
    return string[1:n-1]


#pour les frequentations il faut rejoindre les gares à leur id par leur nom
def traitement_accents_tirets(string):
    n=len(string)
    suppression=0  #j'enlève les tirets et les espaces associés
    string=string.upper() #majuscules
    i=0
    while i < n-suppression:
        lettre=string[i]
        if lettre=='-':
            if string[i+1]==' ' and string[i-1]==' ':
                string=string[:i-1]+' '+string[i+2:]
                suppression+=2
            else:
                string=string[:i]+' '+string[i+1:]
        if lettre in 'ÀÄÂ':
            string=string[:i]+'A'+string[i+1:]
        elif lettre in 'ÉÈÊË':
            string=string[:i]+'E'+string[i+1:]
        elif lettre=='Ç':
            string=string[:i]+'C'+string[i+1:]
        i+=1
    return string
```

```
110|
111| ## frequentation
112| # elles sont loin d'être exactes mais permettent des estimations
113| '''les fréquentations utilisées dans TIPE_traffic sont obtenues à partir du nombre de gares proches (importance donc du pôle de population) et d'un facteur
multiplicatif
114| le facteur est calculé pour obtenir des fréquentations cohérentes avec celles du RER B qui sont obtenues ici'''
115|
116| voyageurs_ratp=open('F:\\informatique\\TIPE\database\\format brut\\trafic-annuel-entrant-par-station-RATP.json','r').read()
117|
118| # on est obligé de considérer que le nom de chaque gare d'idf est unique
119| def remplir_frequentation():
120|     c.execute('delete from flux_emis')
121|     conn.commit()
122|     print('delete done')
123|     liste=selection_reseau_RER_metro()
124|     ajout_voyageurs(liste)
125|
126| def ajout_voyageurs(liste):
127|     data=json.loads(voyageurs_ratp)
128|     print(len(data), 'gares enregistrées')
129|     compt=0
130|     for line in data:
131|             nom, freq, route_type=ratp_voyageurs(line)
132|             reponse=comparaison(liste, nom)
133|             if reponse!=None:
134|                 id_groupe,route_id=reponse
135|                 c.execute('insert into flux_emis (id_groupe, route_id, frequentation_m, frequentation_c) values (?,?,?,?)',(id_groupe, route_id, 0, freq))
136|                 compt+=1
137|             if compt%200==0:
138|                 print(compt)
139|     print('done\n', compt, ' freq_c attribuées\n')
140|     conn.commit()
141|
142| def ratp_voyageurs(line):
143|     nom=traitement_accents_tirets(line['fields']['station'])
144|     freq=line['fields']['trafic']//365
145|     res=line['fields']['reseau']
146|     if res=='Métro':
147|         route_type=1
148|     elif res=='RER':
149|         route_type=2
150|     else:
151|         route_type=None
152|     return nom, freq, route_type
153|
154| #on ne sélectionne que les stations de RER/métro où ne passe qu'une seule ligne --> pas d'ambiguité sur l'attribution du flux émis
155| def selection_reseau_RER_metro():
156|     c.execute('''
157|     select id_groupe, stop_name, route_id, route_type
158|     from (
159|         select id_groupe, stop_name, route_id, route_type, count(route_id) as c
160|         from(
161|             select DISTINCT stops_groupe.id_groupe, stops_groupe.stop_name, graphe.route_id, graphe.route_type
162|                 from graphe
163|                 join stops_groupe
164|                     on stops_groupe.id_groupe=graphe.from_id_groupe
```

```
                where (route_type=1 or route_type=2)
                order by stop_name
            )
        group by id_groupe
        )
        where c=1''')
    liste=c.fetchall()
    print('liste réseau RER_metro récupérée')
    return liste

#le coût est un peu grand mais le nombre de gares réduit aux RER donc ok
def comparaison(liste, nom):
    for id_groupe, stop_name, route_id, route_type in liste:
        if nom in stop_name or stop_name in nom:
            liste.remove((id_groupe, stop_name, route_id, route_type))
            return id_groupe, route_id


##mobilités
code_postaux=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\laposte_hexasmal.csv','r')
mobilites=codecs.open('F:\\informatique\\TIPE\\database\\format brut\\flux_mobilite_domicile_lieu_travail.csv','r')

#limites de l'idf (grossières):
latmin, latmax, lonmin, lonmax=47.9, 49.5, 1.1, 3.6
def traitement_postaux(num_lig, line):
    data=line.split(';')
    code_insee, name, code_postal = data[:3]
    coords=data[5]
    lat, lon = coords.split(',')
    lat, lon = float(lat), float(lon)
    if lat>latmin and lon>lonmin and lon<lonmax and lat<latmax and int(code_insee)!=89126:
        c.execute('insert into mobilites_pro(cle, code_commune, code_postal, name, lat, lon) values(?,?,?,?,?,?)', (num_lig, code_insee, code_postal, name, lat,
lon))

def elimination_doublons():
    c.execute('''
    delete from mobilites_pro
    where cle not in (
    select  cle
        from mobilites_pro
        group by code_commune)''')
    conn.commit()
    print('sans doublons\n')

def traitement_mobilites(num_lig, line):
    if num_lig>4:
        data=line.split(',')
        code_insee=data[0]
        flux_emis_jour=data[3]
        c.execute('update mobilites_pro set flux_emis_jour={f} where code_commune={c}'.format(f=flux_emis_jour, c=code_insee))


##fonctions à appeler
def ecriture():
    modele("stops", traitement_stops, stops, True)
```

```
    modele("routes", traitement_routes, routes, True)
    modele("trips", traitement_trips, trips, True)
    modele("stop_times", traitement_stop_times, stop_times, True)
    modele("transfers", traitement_transfers, transfers, True)
    modele("agency", traitement_agency, agency, True)
    modele("calendar", traitement_calendar, calendar, True)
    modele("calendar_dates", traitement_calendar_dates, calendar_dates, True)
    remplir_frequentation()
    modele('mobilites_pro', traitement_postaux, code_postaux, True)
    elimination_doublons()
    modele('mobilites_pro', traitement_mobilites, mobilites, False)
    conn.close()

```