PARTIE 2 : CRÉATION D'UNE BASE DE DONNÉES RELATIONNELLE

1. Import des librairies python et de l'ORM SQLAlchemy

Definition ORM: Object–relational mapping (ORM, O/RM, and O/R mapping tool) in computer science is a programming technique for converting data between incompatible type systems using object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language.

1. Import de deux fichiers csv contenant les travaux de la partie 1
2. Préparation des 4 tables demandées avec les colonnes demandées
3. Import de ces tables dans la base de données
4. Je les importe finalement dans ma bdd afin de pouvoir faire les requetes SQL demandées.

In [1]:

```
# Import des librairies
import pandas as pd
import numpy as np
from sqlalchemy import create_engine # permet de se connecter à la base de donnée
from IPython.display import display


# je cré l'engine avec SQLAlchemy. La bdd nommé P3_2.db sera creer automatiquement dans '/Users/ismail/Projet3'
# selon la configuration de mon ordinateur (Macbook Air M1)(ref. voir https://getdoc.wiki/Sqlalchemy-dialects)

bdd_engine = create_engine('sqlite:///P3_2.db')
```

In [2]:

```
#visualization du chemin
%pwd
```

Out[2]:

```
'/Users/ismail/Projet3'
```

In [3]:

```
# J'importe les données de la 1er partie pour creer les 3 premieres tables.
df_partie1 = pd.read_csv("data_partie_I.csv")

# Je renomme les colonnes car SQL ne gere pas les lettres avec des accents ou caracteres sepciaux
df_partie1.rename(columns={
    'Zone' : 'zone',
    'Code zone' : 'code zone',
    'Produit' : 'produit',
    'Population' : 'population',
    'Code Produit' : 'code produit',
    'Disponibilité alimentaire en quantité (kg/personne/an)' : 'dispo_alim_kg',
    'Disponibilité alimentaire (Kcal/personne/jour)' : 'dispo_alim_kcal_an',
    'Disponibilité de protéines en quantité (g/personne/jour)' : 'dispo_prot',
    'Disponibilité de matière grasse en quantité (g/personne/jour)' : 'dispo_mat_gr',
    'Disponibilité intérieure' : 'dispo_int',
    'Aliments pour animaux' : 'alim_ani',
    'Semences' : 'semences',
    'Pertes' : 'pertes',
    'Traitement' : 'transfo',
    'Nourriture' : 'nourriture',
    'Autres utilisations (non alimentaire)' : 'autres_utilisations',
    },
    inplace=True)

# Affichage du dataframe
df_partie1.head()
```

Out[3]:

	zone	code zone	code produit	produit	origine	alim_ani	autres_utilisations	dispo_alim_kcal_an	dispo_alim_kg	dispo_mat_gr	...	pertes	Production	semences	transfo
0	Afghanistan	2	2511	Blé	vegetale	NaN	NaN	1369.0	160.23	4.69	...	775000000.0	5.169000e+09	322000000.0	NaN
1	Afghanistan	2	2513	Orge	vegetale	360000000.0	NaN	26.0	2.92	0.24	...	52000000.0	5.140000e+08	22000000.0	NaN
2	Afghanistan	2	2514	Maïs	vegetale	200000000.0	NaN	21.0	2.50	0.30	...	31000000.0	3.120000e+08	5000000.0	NaN
3	Afghanistan	2	2517	Millet	vegetale	NaN	NaN	3.0	0.40	0.02	...	1000000.0	1.300000e+07	0.0	NaN
4	Afghanistan	2	2531	Pommes de Terre	vegetale	NaN	NaN	15.0	7.53	0.04	...	9000000.0	3.030000e+08	23000000.0	NaN

5 rows × 25 columns

In [4]:

```
# Je verfie les noms des colonnes
df_partie1.columns
```

Out[4]:

```
Index(['zone', 'code zone', 'code produit', 'produit', 'origine', 'alim_ani',
       'autres_utilisations', 'dispo_alim_kcal_an', 'dispo_alim_kg',
       'dispo_mat_gr', 'dispo_prot', 'dispo_int', 'Exportations - Quantité',
       'Importations - Quantité', 'nourriture', 'pertes', 'Production',
       'semences', 'transfo', 'Variation de stock', 'population',
       'DispoAlimKcal', 'DispoProtKg', 'Ratio Kcal/KgNourriture', '%proteine'],
      dtype='object')
```

In [5]:

```
# Création du dataframe df_population
df_population = df_partie1[['zone','code zone','population']]


# Affichage du dataframe
df_population.head()
```

Out[5]:

	zone	code zone	population
0	Afghanistan	2	30552000
1	Afghanistan	2	30552000
2	Afghanistan	2	30552000
3	Afghanistan	2	30552000
4	Afghanistan	2	30552000

In [6]:

```
#Pour eviter d'avoir des lignes completement identiques pour chaque pays je supprime les doublons pour avoir une
#seule ligne par pays
df_population = df_population.drop_duplicates(subset=['code zone'],inplace=False)
df_population.head()
```

Out[6]:

	zone	code zone	population
0	Afghanistan	2	30552000
51	Afrique du Sud	202	52776000
135	Albanie	3	3173000
200	Algérie	4	39208000
276	Allemagne	79	82727000

In [7]:

```
# Clef primaire proposée pour la table df_population: code zone
df_population['primary_key'] = df_population['code zone']
```

In [ ]:

In [8]:

```
# Création du dataframe df_dispo_alim
df_dispo_alim = df_partie1[['zone', 'code zone', 'produit', 'code produit', 'origine', 'dispo_prot','dispo_alim_kcal_an','dispo_mat_gr','DispoAlimKcal', '


# Affichage du dataframe
display(df_dispo_alim)
```

	zone	code zone	produit	code produit	origine	dispo_prot	dispo_alim_kcal_an	dispo_mat_gr	DispoAlimKcal	DispoProtKg
0	Afghanistan	2	Blé	2511	vegetale	36.91	1369.0	4.69	1.526638e+13	4.116011e+08
1	Afghanistan	2	Orge	2513	vegetale	0.79	26.0	0.24	2.899385e+11	8.809669e+06
2	Afghanistan	2	Maïs	2514	vegetale	0.56	21.0	0.30	2.341811e+11	6.244829e+06
3	Afghanistan	2	Millet	2517	vegetale	0.08	3.0	0.02	3.345444e+10	8.921184e+05
4	Afghanistan	2	Pommes de Terre	2531	vegetale	0.25	15.0	0.04	1.672722e+11	2.787870e+06
...	...	...	...	...	...	...	...	...	...	...
11582	Îles Salomon	25	Poissons Marins, Autres	2764	animale	5.57	36.0	1.33	7.371540e+09	1.140541e+06
11583	Îles Salomon	25	Mollusques, Autres	2767	animale	0.06	0.0	0.01	0.000000e+00	1.228590e+04
11584	Îles Salomon	25	Riz (Eq Blanchi)	2805	vegetale	10.90	623.0	1.36	1.275686e+11	2.231938e+06
11585	Îles Salomon	25	Lait - Excl Beurre	2848	animale	1.05	19.0	0.70	3.890535e+09	2.150033e+05
11586	Îles Salomon	25	Miscellanees	2899	vegetale	0.10	3.0	0.04	6.142950e+08	2.047650e+04

11587 rows × 10 columns

In [9]:

```
#Clef primaire proposée pour la table df_dispo_alim: combinaison de code zone + code produit.
with pd.option_context('mode.chained_assignment', None):
    df_dispo_alim['primary_key'] = df_dispo_alim['code zone'].astype(str) + '-' + df_dispo_alim['code produit'].astype(str)
pd.reset_option('mode.chained_assignment')
```

In [10]:

```
# Création du dataframe df_equilibre_prod
df_equilibre_prod = df_partie1[['zone', 'code zone', 'produit', 'code produit', 'dispo_int', 'alim_ani', 'semences', 'pertes', 'transfo', 'nourriture', 'a

# Affichage du dataframe
display(df_equilibre_prod)
```

	zone	code zone	produit	code produit	dispo_int	alim_ani	semences	pertes	transfo	nourriture	autres_utilisations
0	Afghanistan	2	Blé	2511	5.992000e+09	NaN	322000000.0	775000000.0	NaN	4.895000e+09	NaN
1	Afghanistan	2	Orge	2513	5.240000e+08	360000000.0	22000000.0	52000000.0	NaN	8.900000e+07	NaN
2	Afghanistan	2	Maïs	2514	3.130000e+08	200000000.0	5000000.0	31000000.0	NaN	7.600000e+07	NaN
3	Afghanistan	2	Millet	2517	1.300000e+07	NaN	0.0	1000000.0	NaN	1.200000e+07	NaN
4	Afghanistan	2	Pommes de Terre	2531	2.620000e+08	NaN	23000000.0	9000000.0	NaN	2.300000e+08	NaN
...	...	...	...	...	...	...	...	...	...	...	...
11582	Îles Salomon	25	Poissons Marins, Autres	2764	1.100000e+07	NaN	NaN	NaN	NaN	1.100000e+07	0.0
11583	Îles Salomon	25	Mollusques, Autres	2767	1.000000e+06	NaN	NaN	NaN	NaN	1.000000e+06	0.0
11584	Îles Salomon	25	Riz (Eq Blanchi)	2805	4.900000e+07	0.0	0.0	1000000.0	NaN	3.600000e+07	12000000.0
11585	Îles Salomon	25	Lait - Excl Beurre	2848	7.000000e+06	0.0	NaN	0.0	NaN	6.000000e+06	NaN
11586	Îles Salomon	25	Miscellanees	2899	NaN	NaN	NaN	NaN	NaN	NaN	NaN

11587 rows × 11 columns

In [11]:

```
#Clef primaire proposée pour la table df_equilibre_prod:  code zone + code produit.
with pd.option_context('mode.chained_assignment', None):
    df_equilibre_prod['primary_key'] = df_equilibre_prod['code zone'].astype(str) + '-' + df_equilibre_prod['code produit'].astype(str)
pd.reset_option('mode.chained_assignment')
```

In [12]:

```
df_sousnut = pd.read_csv("sousnut_partie_I.csv")

# Création du dataframe df_sous_nutrition
df_sousnut1 = df_sousnut[['Zone','Code zone','sousalimentés','Population']]

df_sousnut1.rename(columns={
    'Zone' : 'zone',
    'Code zone' : 'code zone',
    'Produit' : 'produit',
    'sousalimentés' : 'sousalimentes',
    'Population' : 'population',
    },
    inplace=True)



# Affichage du dataframe
display(df_sousnut1)
```

```
/opt/anaconda3/lib/python3.8/site-packages/pandas/core/frame.py:4296: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().rename(
```

	zone	code zone	sousalimentes	population
0	Afghanistan	2	7900000.0	30552000
1	Afghanistan	2	7900000.0	30552000
2	Afghanistan	2	7900000.0	30552000
3	Afghanistan	2	7900000.0	30552000
4	Afghanistan	2	7900000.0	30552000
...	...	...	...	...
15600	Îles Salomon	25	50000.0	561000
15601	Îles Salomon	25	50000.0	561000
15602	Îles Salomon	25	50000.0	561000
15603	Îles Salomon	25	50000.0	561000
15604	Îles Salomon	25	50000.0	561000

15605 rows × 4 columns

In [13]:

```
#Pour eviter d'avoir des lignes completement identiques pour chaque pays je supprime les doublons pour avoir une
#seule ligne par pays
df_sousnut1 = df_sousnut1.drop_duplicates(subset=['code zone'],inplace=False)
df_sousnut1.head()
```

Out[13]:

	zone	code zone	sousalimentes	population
0	Afghanistan	2	7900000.0	30552000
60	Afrique du Sud	202	2600000.0	52776000
155	Albanie	3	200000.0	3173000
247	Algérie	4	1700000.0	39208000
340	Allemagne	79	0.0	82727000

In [14]:

```
#Clef primaire proposée pour la table df_sousnut1: code zone
df_sousnut1['primary_key'] = df_sousnut1['code zone']
```

In [15]:

```
#Transfer des tables dans la bdd
df_population.to_sql(
    name='population', # nom de la table apres transfer dans la bdd
    con = bdd_engine, # connexion utilisé (ici, celle donnée par SQLAlchemy)
    index=False,  # on ne veut pas écrire les index dans la table
    if_exists='replace' # si la table existe déjà, on l'écrase
)

df_dispo_alim.to_sql(
    name='dispo_alim',
    con = bdd_engine,
    index=False,
    if_exists='replace'
)

df_equilibre_prod.to_sql(
    name='equilibre_prod',
    con = bdd_engine,
    index=False,
    if_exists='replace'
)

df_sousnut1.to_sql(
    name='sous_nutrition',
    con = bdd_engine,
    index=False,
    if_exists='replace'
)
```

In [16]:

```
#Question 19 : Écrivez les requêtes SQL permettant de connaître…
#Les 10 pays ayant le plus haut ratio disponibilité alimentaire/habitant en termes de protéines (en kg)
#par habitant, puis en termes de kcal par habitant.

#I. en kg

top10_dispo_prot = pd.read_sql_query(con=bdd_engine, sql = f"""
                    SELECT
                        zone,
                        sum(dispo_prot) as dispoprot
                    FROM dispo_alim
                    GROUP BY zone
                    ORDER BY dispoprot DESC
                    LIMIT 10""")

top10_dispo_prot.index += 1

print('Top 10 des pays avec le plus de disponibilité alimentaire par habitant et par an (en kg de protéines)')
display(top10_dispo_prot)
```

```
Top 10 des pays avec le plus de disponibilité alimentaire par habitant et par an (en kg de protéines)
```

	zone	dispoprot
1	Islande	130.91
2	Chine - RAS de Hong-Kong	129.01
3	Israël	127.98
4	Lituanie	124.17
5	Maldives	119.16
6	Finlande	117.41
7	Luxembourg	111.90
8	Pays-Bas	111.45
9	Albanie	111.14
10	Portugal	110.86

In [17]:

```
#II. en Kcal

top10_dispo_kcal = pd.read_sql_query(con=bdd_engine, sql = f"""
                    SELECT
                        zone,
                        sum(dispo_alim_kcal_an) dispoalimkcal
                    FROM dispo_alim
                    GROUP BY zone
                    ORDER BY dispoalimkcal DESC
                    LIMIT 10""")

top10_dispo_kcal.index += 1

print('Top 10 des pays avec le plus de disponibilité alimentaire par habitant et par an (en kcal)')
display(top10_dispo_kcal)
```

```
Top 10 des pays avec le plus de disponibilité alimentaire par habitant et par an (en kcal)
```

	zone	dispoalimkcal
1	Autriche	3769.0
2	Belgique	3737.0
3	Turquie	3708.0
4	États-Unis d'Amérique	3682.0
5	Israël	3610.0
6	Irlande	3596.0
7	Italie	3578.0
8	Égypte	3518.0
9	Allemagne	3503.0
10	Canada	3499.0

In [18]:

```
# les 10 pays ayant le plus faible ratio disponibilité alimentaire/habitant
#en termes de protéines (en kg) par habitant.

top10_faible_dispo_prot = pd.read_sql_query(con=bdd_engine, sql = f"""
                    SELECT
                        zone,
                        sum(dispo_prot) as dispoprot
                    FROM dispo_alim
                    GROUP BY zone
                    ORDER BY dispoprot ASC
                    LIMIT 10""")

top10_faible_dispo_prot.index += 1

print('Top 10 des pays avec le moins de disponibilité alimentaire par habitant et par an (en kg de protéines)')
display(top10_faible_dispo_prot)
```

```
Top 10 des pays avec le moins de disponibilité alimentaire par habitant et par an (en kg de protéines)
```

	zone	dispoprot
1	Libéria	37.62
2	Guinée-Bissau	43.91
3	Mozambique	45.67
4	République centrafricaine	46.03
5	Madagascar	46.68
6	Haïti	47.69
7	Zimbabwe	48.32
8	Congo	51.30
9	Sao Tomé-et-Principe	51.65
10	Ouganda	52.64

In [19]:

```
#La quantité totale (en kg) de produits perdus par pays

pertes_par_pays = pd.read_sql_query(con=bdd_engine, sql = f"""
                    SELECT
                        zone,
                        sum(pertes)
                    FROM equilibre_prod
                    GROUP BY zone
                    ORDER BY zone""")

print('Quantité des pertes par pays (en kg)')
display(pertes_par_pays)
```

```
Quantité des pertes par pays (en kg)
```

	zone	sum(pertes)
0	Afghanistan	1.135000e+09
1	Afrique du Sud	2.193000e+09
2	Albanie	2.750000e+08
3	Algérie	3.747000e+09
4	Allemagne	3.781000e+09
...	...	...
169	Émirats arabes unis	7.050000e+08
170	Équateur	7.040000e+08
171	États-Unis d'Amérique	7.162000e+09
172	Éthiopie	2.256000e+09
173	Îles Salomon	6.000000e+06

174 rows × 2 columns

In [20]:

```
#Les 10 pays pour lesquels la proportion de personnes sous-alimentées est la plus forte

top10_pays_sous_nutrition = pd.read_sql_query(con=bdd_engine, sql=f"""
                        SELECT
                            zone,
                            ROUND(AVG((sousalimentes / population) * 100),2) AS prop_p_ss_alim
                        FROM sous_nutrition
                        NATURAL JOIN sous_nutrition
                        GROUP BY zone
                        ORDER BY prop_p_ss_alim DESC
                        LIMIT 10""")

top10_pays_sous_nutrition.index += 1

print('Top 10 des pays avec la proportion de personnes sous-alimentées la plus importante (en %)')
display(top10_pays_sous_nutrition)
```

```
Top 10 des pays avec la proportion de personnes sous-alimentées la plus importante (en %)
```

	zone	prop_p_ss_alim
1	Dominique	69.44
2	Haïti	50.40
3	Kiribati	49.02
4	Zambie	48.15
5	Zimbabwe	46.64
6	Saint-Vincent-et-les Grenadines	45.87
7	République centrafricaine	43.33
8	République populaire démocratique de Corée	42.58
9	Congo	40.47
10	Tchad	38.21

In [21]:

```
#Les 10 produits pour lesquels le ratio Autres utilisations/Disponibilité intérieure est le plus élevé.

top10_produit_autres_util = pd.read_sql_query(con=bdd_engine, sql=f"""
                        SELECT
                            produit AS Produit,
                            ROUND(AVG(autres_utilisations / dispo_int)*100,2) AS Part_autres_utilisations
                        FROM equilibre_prod
                        GROUP BY Produit
                        ORDER BY Part_autres_utilisations DESC
                        LIMIT 10""")

top10_produit_autres_util.index += 1

print('Top 10 des produits pour lesquels le ratio Autres utilisations/Disponibilité intérieure est le plus élevé')
display(top10_produit_autres_util)
```

```
Top 10 des produits pour lesquels le ratio Autres utilisations/Disponibilité intérieure est le plus élevé
```

	Produit	Part_autres_utilisations
1	Alcool, non Comestible	98.25
2	Piments	73.91
3	Huile de Palmistes	72.65
4	Huile de Palme	62.17
5	Palmistes	57.56
6	Huile de Colza&Moutarde	57.36
7	Huile de Son de Riz	50.30
8	Huil Plantes Oleif Autr	49.66
9	Huile de Coco	48.99
10	Sucre non centrifugé	46.51

#Question 20 : pour quelques uns des produits identifiés dans cette dernière requête SQL, supposez quelles sont ces "autres utilisations" possibles (recherchez sur internet !) 1. Piments: Utilisé comme parfum ou insecticide: https://fr.wikipedia.org/wiki/Piment#Parfums_et_cosmétiques https://fr.wikipedia.org/wiki/Piment#Comme_insecticide 2. Alcool, non comestible: https://ec.europa.eu/taxation_customs/denatured-alcohol-not-human-consumption_fr Alcool dénaturé non destiné à la consommation humaine L'alcool peut être rendu impropre à la consommation humaine par l'ajout de produits ayant très mauvais goût et/ou très mauvaise odeur et (de préférence) d'un marqueur chimique. Le procédé permettant de rendre l'alcool impropre à la consommation humaine est appelé dénaturation (totale ou partielle) de l'alcool. Lorsqu'il a été dénaturé conformément à la réglementation de l'UE, l'alcool peut être exonéré de droits d'accise. Cette règle s'applique notamment à l'alcool utilisé à des fins industrielles, pour produire des denrées alimentaires et des médicaments ou pour fabriquer des cosmétiques, des biocarburants, des peintures, ainsi que des produits antigel et de nettoyage. 3. Huile de Palme: pour la production des cosmetiques et des agrocarburants https://fr.wikipedia.org/wiki/Huile_de_palme#Oléochimie https://fr.wikipedia.org/wiki/Huile_de_palme#Agro-carburants