In [1]:
```python
# Import librairies python
import pandas as pd
import numpy as np
import os
```

# Téléchargement des fichiers

In [2]:
```python
# Téléchargement des fichiers
df_communes = pd.read_excel('donnees_communes.xlsx')
df_valeurs = pd.read_excel('Valeurs-foncières.xlsx')
df_ref_geo = pd.read_excel('fr-esr-referentiel-geographique.xlsx')
```

# Nombre d'enregistrements par fichiers

In [3]:
```python
# Calcul des volumes sources
source_metrics = {
    'commune': len(df_communes),
    'vente': len(df_valeurs),
    'bien': len(df_valeurs), # Puisque 1 ligne de vente = 1 bien dans ta logique
    'departement': df_ref_geo['dep_code'].nunique(),
    'region': df_ref_geo['reg_code'].nunique()
}

print(f"Nombre d'enregistrements des régions : {source_metrics['region']} ")
print(f"Nombre d'enregistrements des départements : {source_metrics['departement']}
print(f"Nombre d'enregistrements des communes : {source_metrics['commune']} ")
print(f"Nombre d'enregistrements des bien : {source_metrics['bien']} ")
print(f"Nombre d'enregistrements des ventes : {source_metrics['vente']} ")
```

```
Nombre d'enregistrements des régions : 19
Nombre d'enregistrements des départements : 109
Nombre d'enregistrements des communes : 34991
Nombre d'enregistrements des bien : 34169
Nombre d'enregistrements des ventes : 34169
```

# Extraction des données pour alimentation table SQL

## 1) Infos fichier region

In [4]:
```python
print("Colonne fichier ref_géo")
print(f"Nombre de lignes et de colonnes du fichier source {df_ref_geo.shape}")
print(df_ref_geo.info())
```

```
Colonne fichier ref_géo
Nombre de lignes et de colonnes du fichier source (38916, 37)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38916 entries, 0 to 38915
Data columns (total 37 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   regrgp_nom          38916 non-null  object
 1   reg_nom             38916 non-null  object
 2   reg_nom_old         38916 non-null  object
 3   aca_nom             38916 non-null  object
 4   dep_nom             38916 non-null  object
 5   com_code            38916 non-null  object
 6   com_code1           38916 non-null  object
 7   com_code2           38916 non-null  object
 8   com_id              38916 non-null  object
 9   com_nom_maj_court   38916 non-null  object
 10  com_nom_maj         38916 non-null  object
 11  com_nom             38916 non-null  object
 12  uu_code             7625 non-null   object
 13  uu_id               38916 non-null  object
 14  uucr_id             38916 non-null  object
 15  uucr_nom            38916 non-null  object
 16  ze_id               38916 non-null  object
 17  dep_code            38916 non-null  object
 18  dep_id              38916 non-null  object
 19  dep_nom_num         38916 non-null  object
 20  dep_num_nom         38916 non-null  object
 21  aca_code            38916 non-null  int64
 22  aca_id              38916 non-null  object
 23  reg_code            38916 non-null  int64
 24  reg_id              38916 non-null  object
 25  reg_code_old        38916 non-null  int64
 26  reg_id_old          38916 non-null  object
 27  fd_id               38916 non-null  object
 28  fr_id               38916 non-null  object
 29  fe_id               38916 non-null  object
 30  uu_id_99            38916 non-null  object
 31  au_code             21444 non-null  object
 32  au_id               38916 non-null  object
 33  auc_id              38916 non-null  object
 34  auc_nom             38916 non-null  object
 35  uu_id_10            38916 non-null  object
 36  geolocalisation     36745 non-null  object
dtypes: int64(3), object(34)
memory usage: 11.0+ MB
None
```

## Génération fichier region pour la bdd

In [5]:
```python
# Supprimer les espaces inutiles autour des noms
df_ref_geo['reg_nom'] = df_ref_geo['reg_nom'].str.strip()

# Conversion du reg_code en string
df_ref_geo['reg_code'] = df_ref_geo['reg_code'].astype(str)
```

```
print(df_ref_geo['reg_code'].dtypes)

# Ajout du zéro pour avoir 2 caractères (ex: '1' devient '01')
df_ref_geo['reg_code'] = df_ref_geo['reg_code'].str.zfill(2)

# Choix des colonnes et suppression des dublons pour avoir des valeurs uniques
region = df_ref_geo[['reg_code','reg_nom','regrgp_nom']].drop_duplicates()

# Tri du dataframe, génération nouveaux index après suppression des anciens
region = region.sort_values('reg_code').reset_index(drop=True)

# Afficher le résultat
print(region.head(3))
print(f"Nombre de lignes et de colonnes de l'extraction region {region.shape}")

# Export des données dans un fichier csv avec le bon encordage
region.to_csv("Export_donnees/region.csv", index=False, encoding='utf-8', sep=';')
```

```
object
  reg_code                    reg_nom regrgp_nom
0       00  Collectivités d'outre-mer   DROM-COM
1       01                 Guadeloupe   DROM-COM
2       02                 Martinique   DROM-COM
Nombre de lignes et de colonnes de l'extraction region (19, 3)
```

## Génération fichier departement pour la bdd

In [6]:
```
# 1. Nettoyage des noms (supp espaces)
df_ref_geo['dep_nom'] = df_ref_geo['dep_nom'].str.strip()

# 2. Conversion et formatage : On traite dep_code ET reg_code pour que la jointure
for col in ['dep_code', 'reg_code']:
    df_ref_geo[col] = df_ref_geo[col].astype(str).str.strip().str.zfill(2)

# 3. Extraction des données département et suppression des doublons
departement = df_ref_geo[['dep_code', 'dep_nom', 'reg_code']].drop_duplicates()

# 4. Tri et réindexation
departement = departement.sort_values('dep_code').reset_index(drop=True)

# 5. Affichage
print(departement.head(3))
print(f"Nombre de lignes et de colonnes de l'extraction departement {departement.sh
# Export des données dans un fichier csv avec le bon encordage
departement .to_csv("Export_donnees/departement.csv", index=False, encoding='utf-8'
```

```
  dep_code dep_nom reg_code
0       01     Ain       84
1       02   Aisne       32
2       03  Allier       84
Nombre de lignes et de colonnes de l'extraction departement (109, 3)
```

## 3) Infos fichier communes

In [7]:
```
print("Colonne fichier communes")
print(f"nombre de lignes et de colonnes du fichier source: {df_communes.shape} ")
```

```
#print(df_communes.info())
```

```
Colonne fichier communes
nombre de lignes et de colonnes du fichier source: (34991, 9)
```

## 4) Infos fichier biens

In [8]:
```
print("Colonne fichier valeurs")
print(f"Nombre de lignes et de colonnes du fichier sources {df_valeurs.shape} ")
#print(df_valeurs.info())
```

```
Colonne fichier valeurs
Nombre de lignes et de colonnes du fichier sources (34169, 47)
```

In [9]:
```
# 1) Supression des colonnes totalement vides
df_valeurs = df_valeurs.dropna(axis=1, how="all")
#print(df_valeurs.info())
```

## 3) Génération fichiers commune/ bien/ vente pour la bdd

In [10]:
```
# --- 1. FONCTION DE NETTOYAGE INSEE  ---
def format_insee(df, col_dep, col_com):
    # Nettoyage des types et suppression des espaces/.0
    dep = df[col_dep].astype(str).str.replace('.0', '', regex=False).str.strip()
    com = df[col_com].astype(str).str.replace('.0', '', regex=False).str.strip()

    # Logique :
    # Si dep=3 car (DROM) -> dep(3) + 2 derniers chiffres de commune
    # Sinon (Métropole) -> dep(2) + 3 derniers chiffres de commune
    return np.where(
        dep.str.len() == 3,
        dep + com.str.zfill(2).str[-2:],
        dep.str.zfill(2).str[-2:] + com.str.zfill(3).str[-3:]
    )

# --- 2. TRAITEMENT DES DONNÉES ---

# Application du formatage INSEE
df_communes['code_dep_com'] = format_insee(df_communes, 'CODDEP', 'CODCOM')
df_valeurs['code_dep_com'] = format_insee(df_valeurs, 'Code departement', 'Code com

# --- 3. GESTION DES CODES POSTAUX (Mapping depuis df_valeurs) ---
df_valeurs['Code postal'] = pd.to_numeric(df_valeurs['Code postal'], errors='coerce
cp_mapping = df_valeurs[['code_dep_com', 'Code postal']].dropna().drop_duplicates(s
cp_mapping['Code postal'] = cp_mapping['Code postal'].astype(int).astype(str).str.z

# --- 4. CRÉATION DES TABLES FINALISÉES ---

# Table COMMUNE
commune = pd.merge(df_communes, cp_mapping, on='code_dep_com', how='left')
commune = commune[['code_dep_com', 'COM', 'Code postal', 'PTOT', 'CODDEP']].drop_du
commune['Code postal'] = commune['Code postal'].fillna('00000') # Sécurité SQL
commune = commune.sort_values('code_dep_com')

# Table BIEN
```

```
cols_bien = ['No voie', 'B/T/Q', 'Type de voie', 'Voie', 'Nombre pieces principales
             'Surface Carrez du 1er lot', 'Surface reelle bati', 'Type local', 'cod
bien = df_valeurs[cols_bien].copy()
bien['No voie'] = bien['No voie'].fillna(0).astype(int)
bien['Nombre pieces principales'] = bien['Nombre pieces principales'].fillna(0).ast
bien['Surface reelle bati'] = bien['Surface reelle bati'].fillna(0).astype(int)
bien.insert(0, 'id_bien', range(1, len(bien) + 1))

# Table VENTE
vente = df_valeurs[['Date mutation', 'Valeur fonciere', 'Nature mutation']].copy()
vente['id_bien'] = bien['id_bien']
vente.insert(0, 'id_vente', range(1, len(vente) + 1))
vente['Date mutation'] = pd.to_datetime(vente['Date mutation']).dt.strftime('%Y-%m-
vente['Valeur fonciere'] = vente['Valeur fonciere'].fillna(0).astype(int)

# --- 5. TESTS DE VALIDATION (Anti-erreur SQL) ---

print("--- VALIDATION DES LONGUEURS ---")
max_len_com = commune['code_dep_com'].str.len().max()
max_len_bien = bien['code_dep_com'].str.len().max()

print(f"Longueur max INSEE (Commune) : {max_len_com}")
print(f"Longueur max INSEE (Bien)    : {max_len_bien}")

# Blocage du script si ce n'est pas parfait
assert max_len_com == 5, f"ERREUR : Longueur max {max_len_com} au lieu de 5"
assert max_len_bien == 5, f"ERREUR : Longueur max {max_len_bien} au lieu de 5"
print("✅ Validation réussie : Tous les codes font 5 caractères.")

# --- 6. EXPORTATION ---

import os
if not os.path.exists('Export_donnees'): os.makedirs('Export_donnees')

commune.to_csv("Export_donnees/commune.csv", index=False, sep=';', encoding='utf-8'
bien.to_csv("Export_donnees/bien.csv", index=False, sep=';', encoding='utf-8')
vente.to_csv("Export_donnees/vente.csv", index=False, sep=';', encoding='utf-8')

print("🚀 Fichiers exportés avec succès !")
```

```
--- VALIDATION DES LONGUEURS ---
Longueur max INSEE (Commune) : 5
Longueur max INSEE (Bien)    : 5
✅ Validation réussie : Tous les codes font 5 caractères.
🚀 Fichiers exportés avec succès !
```

## Vérification des données Exportées

In [11]:

```
# 1. Chargement des fichiers exportés

try:
    c_check = pd.read_csv("Export_donnees/commune.csv", sep=';')
    b_check = pd.read_csv( "Export_donnees/bien.csv", sep=';')
    v_check = pd.read_csv("Export_donnees/vente.csv", sep=';')
    print("✅ Fichiers chargés avec succès.")
```

```
except Exception as e:
    print(f"❌ Erreur lors du chargement : {e}")

# 2. VÉRIFICATION DES CODES INSEE (VARCHAR 5)
print("\n--- TEST CODES INSEE ---")
insee_errors_c = c_check[c_check['code_dep_com'].astype(str).str.len() != 5]
insee_errors_b = b_check[b_check['code_dep_com'].astype(str).str.len() != 5]

if insee_errors_c.empty and insee_errors_b.empty:
    print("✅ OK : Tous les codes INSEE font exactement 5 caractères.")
else:
    print(f"⚠️ ATTENTION : {len(insee_errors_c)} erreurs dans communes, {len(insee_
    if not insee_errors_c.empty: print(insee_errors_c['code_dep_com'].unique())

# 3. VÉRIFICATION DES TYPES NUMÉRIQUES (Pas de .0)
print("\n--- TEST TYPES NUMÉRIQUES ---")
# On regarde si les colonnes censées être INT contiennent des flottants
sample_bien = b_check[['No voie', 'Nombre pieces principales', 'Surface reelle bati
print("Aperçu des colonnes numériques (doivent être sans .0) :")
print(sample_bien)

# 4. APERÇU OUTRE-MER (Le point critique)
print("\n--- APERÇU GUADELOUPE (971) ---")
print(c_check[c_check['code_dep_com'].astype(str).str.startswith('971')].head(5))

# 5. VÉRIFICATION DES DATES
print("\n--- TEST FORMAT DATE ---")
print(f"Format de la première date de vente : {v_check['Date mutation'].iloc[0]}")
# Doit être YYYY-MM-DD
```

```
✅ Fichiers chargés avec succès.

--- TEST CODES INSEE ---
✅ OK : Tous les codes INSEE font exactement 5 caractères.

--- TEST TYPES NUMÉRIQUES ---
Aperçu des colonnes numériques (doivent être sans .0) :
   No voie  Nombre pieces principales  Surface reelle bati
0      347                         3                  48

--- APERÇU GUADELOUPE (971) ---
      code_dep_com            COM  Code postal   PTOT CODDEP
34878        97101     Les Abymes            0  54027    971
34879        97102  Anse-Bertrand            0   4065    971
34880        97103   Baie-Mahault            0  31335    971
34881        97104        Baillif            0   5295    971
34882        97105    Basse-Terre            0  10105    971

--- TEST FORMAT DATE ---
Format de la première date de vente : 2020-01-02
```

In [12]:
```
print("--- VÉRIFICATION DE L'EXHAUSTIVITÉ ---")

# 1. Vérification des Ventes
# On compare le nombre de lignes initiales vs le nombre de lignes dans la table Ven
initial_ventes = len(df_valeurs)
```

```python
final_ventes = len(vente)
ecart_ventes = initial_ventes - final_ventes

print(f"Ventes initiales : {initial_ventes}")
print(f"Ventes exportées  : {final_ventes}")
if ecart_ventes == 0:
    print("✅ Succès : Aucune vente perdue.")
else:
    print(f"⚠️ Attention : Écart de {ecart_ventes} lignes sur les ventes.")

# 2. Vérification des Communes
# On vérifie si toutes les communes du référentiel sont présentes
initial_communes = df_communes['CODDEP'].count() # ou une autre colonne de référenc
final_communes = len(commune)
ecart_com = initial_communes - final_communes

print(f"\nCommunes initiales : {initial_communes}")
print(f"Communes exportées  : {final_communes}")
if ecart_com == 0:
    print("✅ Succès : Toutes les communes sont présentes.")
else:
    print(f"ℹ️ Note : Écart de {ecart_com} communes (souvent dû aux drop_duplicates

# 3. Vérification des liens (Clés étrangères)
# Est-ce que tous les codes INSEE de la table BIEN existent dans la table COMMUNE ?
insee_bien = set(bien['code_dep_com'])
insee_commune = set(commune['code_dep_com'])
orphelins = insee_bien - insee_commune

if not orphelins:
    print("\n✅ Intégrité : Tous les biens sont rattachés à une commune existante."
else:
    print(f"\n⚠️ Alerte : {len(orphelins)} codes INSEE dans BIEN n'existent pas dar
    print(f"Exemple d'orphelins : {list(orphelins)[:5]}")
```

```
--- VÉRIFICATION DE L'EXHAUSTIVITÉ ---
Ventes initiales : 34169
Ventes exportées  : 34169
✅ Succès : Aucune vente perdue.

Communes initiales : 34991
Communes exportées  : 34991
✅ Succès : Toutes les communes sont présentes.

✅ Intégrité : Tous les biens sont rattachés à une commune existante.
```

In [ ]: