Mélanie Donne

# Rapport

## Analyse de l'immobilier à Paris

Jems WE MAKE DATA INGENIOUS × DataScientest

# Sommaire

## Introduction

Ce rapport détaille les progrès réalisés dans le cadre du projet de l’analyse de l'immobilier à Paris.

Le premier objectif est de collecter des données brutes à partir de différentes sources d'API, puis de les nettoyer, de les prétraiter et de les préparer pour une analyse ultérieure.

Dans cette première étape, nous avons concentré nos efforts sur trois principaux aspects : la préparation des données foncières, l'extraction des données via des requêtes API et le nettoyage des résultats obtenus. À cette fin, nous avons développé trois scripts Python distincts : data-cleaning.py pour les données foncières, api-requests.py pour l'extraction des données et api-cleaning-results.py pour leur nettoyage et leur préparation.

Le deuxième objectif est de proposer une exploration détaillée et méthodique des aspects fondamentaux de la gestion et de l'analyse des données géospatiales. Nous aborderons successivement l'optimisation et le prétraitement des données géospatiales à l'aide de Mapshaper, la conception de diagrammes entité-association, la création de bases de données, et enfin, nous illustrerons l'utilisation de différentes requêtes.

Le dernier objectif est de fournir une vue d'ensemble du processus, en mettant en évidence les étapes clés et les défis rencontrés. Le processus commence par l'extraction des données à partir de diverses sources. Ensuite, les données sont transformées pour les rendre cohérentes et prêtes à l'analyse, avec des opérations. Enfin, les données transformées sont chargées dans une base de données cible. Chaque étape du processus ETL est détaillée, avec les méthodes, les outils et les résultats.

Ce rapport souligne l’engagement qui a été pris afin de mener à bien le projet d'analyse immobilière à Paris. Les différentes étapes, de la collecte des données à leur traitement, témoignent de l’approche pragmatique et de la capacité à relever les défis pratiques.

# 1. Exploration de données non structurées

## A. Le nettoyage des données

Pour commencer, j'ai codé le script `data-cleaning.py`
qui effectue plusieurs opérations de nettoyage et de prétraitement sur un jeu de données relatives aux valeurs foncières de l'année 2022.

### 1. Chargement des Données :

Le script utilise la librairie Pandas pour charger les données du fichier `valeursfoncieres-2022.txt` dans un DataFrame, en spécifiant que les colonnes sont séparées par le caractère "|".

```
# Charger les données de 2022
vf2022 = pd.read_csv("./valeursfoncieres-2022.txt", sep="|", low_memory=False)
```

### 2. Nettoyage des Colonnes avec des Données Manquantes :

Les colonnes contenant des valeurs manquantes sont identifiées, puis celles dont le pourcentage de valeurs manquantes dépasse 50% sont supprimées du DataFrame. Cette opération vise à éliminer les colonnes avec des données insuffisantes pour une analyse significative.

```
# Nettoyage des colonnes avec des données manquantes
# Identifier les colonnes avec des valeurs manquantes
columns_with_nan = vf2022.columns[vf2022.isna().any()]

# Parcourir les colonnes avec des valeurs manquantes
for column in columns_with_nan:
    # Calculer le pourcentage de valeurs manquantes dans la colonne
    percentage_missing = vf2022[column].isna().sum() * 100.0 / vf2022.shape[0]
      # Supprimer la colonne si le pourcentage de valeurs manquantes est
supérieur à 50%
    if percentage_missing > 50:
        vf2022.drop(column, axis=1, inplace=True)
```

### 3. Suppression des Colonnes Inutiles :

Certaines colonnes jugées inutiles pour l'analyse sont supprimées du DataFrame. Ces colonnes comprennent des informations telles que la section, le numéro de plan, le nombre de lots, etc.

```
# Supprimer les colonnes inutiles
useless_columns = ['Section', 'No plan', 'Nombre de lots', 'Code type local',
'Nature culture', 'No disposition']
vf2022.drop(useless_columns, axis=1, inplace=True)
```

### 4. Nettoyage des Lignes avec des Données Manquantes :

Les lignes contenant au moins 14 valeurs manquantes sont supprimées du DataFrame. Cette opération vise à éliminer les lignes ayant un nombre significatif de valeurs manquantes.

```
# Nettoyage des lignes avec des données manquantes
# Supprimer les lignes avec au moins 14 valeurs manquantes
vf2022 = vf2022.dropna(axis=0, thresh=14)
```

### 5. Conversion du Format de Données :

La colonne "Valeur fonciere" est nettoyée en remplaçant les virgules par des points pour assurer un format numérique approprié.

```
# Remplacer les virgules par des points dans la colonne "Valeur fonciere"
vf2022["Valeur fonciere"] = vf2022["Valeur fonciere"].str.replace(',', '.')
```

### 6. Conversion des Types de Données :

Certaines colonnes sont converties dans des types de données appropriés. Par exemple, la colonne "Valeur fonciere" est convertie en type numérique, la colonne "Surface reelle bati" est également convertie en type numérique, et la colonne "Code postal" est convertie en type entier (int32). De plus, la colonne "Date mutation" est convertie en type de données datetime.

```
# Convertir certaines colonnes en types appropriés
vf2022["Valeur fonciere"] = pd.to_numeric(vf2022["Valeur fonciere"])
vf2022["Surface reelle bati"] = pd.to_numeric(vf2022["Surface reelle bati"])
vf2022["Code postal"] = vf2022["Code postal"].astype({'Code postal': 'int32'})
vf2022["Date mutation"] = pd.to_datetime(vf2022["Date mutation"])
```

### 7. Filtrage des Données pour Paris :

Les données sont filtrées pour inclure uniquement les codes postaux de Paris, définis dans la liste paris_postal_codes.

```
# Déterminer les codes postaux de Paris
paris_postal_codes = [75001, 75002, 75003, 75004, 75005, 75006, 75007, 75008,
75009, 75010,
                      75011, 75012, 75013, 75014, 75015, 75016, 75017, 75018,
75019, 75020]

# Filtrer les données pour n'inclure que les codes postaux de Paris
vf_paris2022 = vf2022[vf2022['Code postal'].isin(paris_postal_codes)]
```

### 8. Enregistrement des Données dans un Fichier CSV :

Les données nettoyées et filtrées sont enregistrées dans un fichier CSV nommé "valeursfoncieres_paris_2022.csv", avec l'index des lignes omis et le format numérique des valeurs défini pour une précision décimale de 1, et en utilisant une virgule comme séparateur de champ et un retour à la ligne comme terminateur de ligne.

```
# Enregistrer les données organisées dans un fichier CSV
vf_paris2022.to_csv("valeursfoncieres_paris_2022.csv",index=False,float_format
='%.1f', sep=',', line_terminator='\n')
```

Ce script permet donc de nettoyer et de préparer les données relatives aux valeurs foncières de l'année 2022 pour une analyse ultérieure, en particulier en se concentrant sur les propriétés localisées à Paris.

## B. La récupération et la sauvegarde de données

Dans le cadre de notre analyse de l'immobilier à Paris, nous avons exploité diverses sources de données, comprenant notamment des données gouvernementales et municipales, ainsi que des API fournissant des informations pertinentes sur les biens immobiliers et leur environnement.
Ce script présente une solution pratique pour automatiser le processus de téléchargement de données CSV à partir de l'API Open Data de Paris. Il offre une manière efficace et rapide d'extraire des ensembles de données spécifiques, prêts à être utilisés pour des analyses approfondies, des visualisations ou d'autres applications dans le domaine de la science des données.

### 1. Fonction download_csv_from_url(url, destination) :

Cette fonction gère le processus de téléchargement des fichiers CSV. Elle prend deux paramètres en entrée : url (l'URL du fichier CSV à télécharger) et destination (le nom du fichier de destination où enregistrer le fichier téléchargé). La fonction effectue une requête HTTP GET vers l'URL spécifiée, vérifie si la réponse est réussie (code de statut HTTP 200), puis crée le répertoire 'api_data' s'il n'existe pas déjà. Enfin, elle enregistre le contenu de la réponse dans un fichier binaire avec le nom de fichier de destination spécifié.

```
def download_csv_from_url(url, destination):
    try:
        response = requests.get(url)
        if response.status_code == 200:
            # Obtenir le chemin complet du répertoire 'api_data' relativement
au dossier du script Python
            api_data_dir = os.path.join(os.path.dirname(__file__), 'api_data')
            # Créer le répertoire 'api_data' s'il n'existe pas
            if not os.path.exists(api_data_dir):
                os.makedirs(api_data_dir)
            # Obtenir le chemin complet du fichier destination dans le dossier
'api_data'
            destination_path = os.path.join(api_data_dir, destination)
            with open(destination_path, 'wb') as f:
                f.write(response.content)
            print(f"Le fichier CSV '{destination}' a été téléchargé avec
succès dans le dossier 'api_data'.")
        else:
            print(f"Erreur lors du téléchargement du fichier '{destination}':
{response.status_code}")
    except Exception as e:
        print(f"Une erreur s'est produite lors du téléchargement du fichier
'{destination}': {str(e)}")
```

2. Liste des fichiers à télécharger (files) :

Dans cette partie du script, une liste `files` est définie, contenant des tuples où chaque tuple représente une paire d'URL de téléchargement et de nom de fichier de destination. Ensuite, une boucle itère à travers cette liste et appelle la fonction download_csv_from_url pour télécharger chaque fichier.

```
# Les URL des différents fichiers CSV avec leurs destinations dans le dossier 'api_data'
files = [

("https://opendata.paris.fr/api/explore/v2.1/catalog/datasets/arrondissements/exports/csv?lang=fr&timezone=Europe%2FBerlin&use_labels=true&delimiter=%3B", "arrondissements.csv"),

("https://opendata.paris.fr/api/explore/v2.1/catalog/datasets/logement-encadrement-des-loyers/exports/csv?lang=fr&timezone=Europe%2FBerlin&use_labels=true&delimiter=%3B", "logement_encadrement_des_loyers.csv"),

]

# Télécharger chaque fichier dans 'api_data'
for file in files:
    download_csv_from_url(file[0], file[1])
```

## C. Les APIs

Les API sélectionnées ont été choisies en fonction de leur pertinence par rapport aux objectifs du projet.

En effet, ces dernières fournissent des données essentielles liées à l'urbanisme, à l'aménagement du territoire, au logement et à d'autres aspects de la vie urbaine qui sont au cœur de l'objet du projet.

De plus, ces informations sont mises à disposition par le gouvernement et sont accessibles publiquement, ce qui garantit la fiabilité et l'authenticité des données.

En résumé, ces API ont été choisies en raison de leur pertinence thématique, de la disponibilité de leurs données, de leur compatibilité avec les objectifs de l'analyse et de leur potentiel pour générer des insights significatifs pour le projet.

### 1. Arrondissements

Cette source fournit des informations sur les arrondissements municipaux de Paris.

### 2. Quartiers Administratifs

Cette source fournit des informations sur les quartiers administratifs de Paris.

### 3. Logement - Encadrement des Loyers

Cette source fournit des informations sur les loyers de référence par quartier à Paris depuis la mise en place de l'encadrement des loyers en 2019.

# 2. Modélisation des données

## A. Optimisation et Prétraitement des Données Géospatiales avec Mapshaper

J'ai décidé d'utiliser Mapshaper avant d'importer des données géospatiales dans ma base de données pour plusieurs raisons.

En utilisant Mapshaper, j'ai pu simplifier les géométries et supprimer les détails inutiles, ce qui a considérablement réduit la taille des fichiers et économiser de l'espace de stockage dans ma base de données.

Ensuite, j'ai cherché à optimiser les performances de ma base de données. Je savais que des données géospatiales complexes pourraient ralentir les opérations de traitement et les requêtes spatiales. En simplifiant les géométries avec Mapshaper, j'ai pu améliorer la vitesse de traitement et l'efficacité des requêtes géospatiales dans ma base de données.

Enfin, j'ai trouvé pratique de pouvoir prévisualiser et valider mes données avant de les importer. L'interface conviviale de Mapshaper m'a permis d'examiner mes données, d'identifier tout problème potentiel et de m'assurer qu'elles étaient prêtes à être intégrées dans ma base de données.

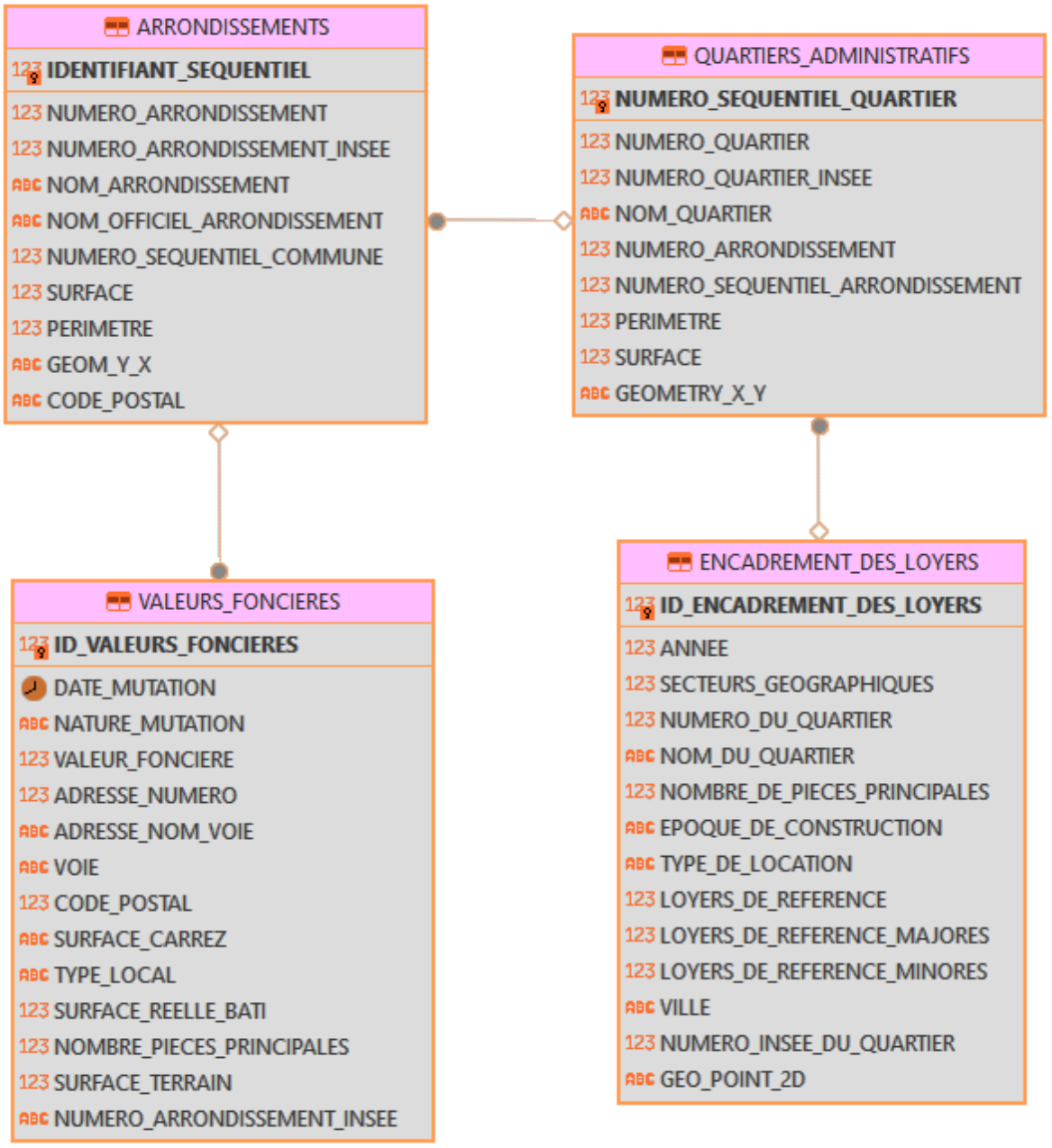
Exemple avec la table arrondissements:


```
ARRONDISSEMENTS AVANT = {
    "IDENTIFIANT_SEQUENTIEL": "750000001",
    "NUMERO_ARRONDISSEMENT": "1",
    "NUMERO_ARRONDISSEMENT_INSEE": "75101",
    "NOM_ARRONDISSEMENT": "1er Ardt",
    "NOM_OFFICIEL_ARRONDISSEMENT": "Louvre",
    "N_SQ_CO": "750001537",
    "SURFACE": "1824612.86048666",
    "PERIMETRE": "6054.93686218",
    "GEOMETRY_X_Y": "48.862562701836005, 2.3364433620533878",
    "GEOMETRY": '{"coordinates": [[[2.328007329038849, 48.86991742140714],
[2.329965588686571, 48.868514169174276], [2.3303067953208765,
48.86835619167468], ... [2.328007329038849, 48.86991742140714]]], "type":
"Polygon"}'
}
```

```
ARRONDISSEMENTS APRES = {
    "IDENTIFIANT_SEQUENTIEL_ARRONDISSEMENT": "750000001",
    "NUMERO_ARRONDISSEMENT": "1",
    "NUMERO_ARRONDISSEMENT_INSEE": "75101",
    "NOM_ARRONDISSEMENT": "1er Ardt",
    "NOM_OFFICIEL_ARRONDISSEMENT": "Louvre",
    "N_SQ_CO": "750001537",
    "SURFACE": "1824612.8604866600",
    "PERIMETRE": "6054.9368621800",
    "GEOMETRY_X_Y": '{"lon": 2.3364433620533878, "lat": 48.862562701836005}'
}
```

Les données après traitement ont été simplifiées, avec un changement de format pour les coordonnées géographiques. Les champs de géométrie ont également été simplifiés pour ne conserver que les coordonnées géographiques au format JSON. Cela rend les données plus compatibles avec les bases de données géospatiales et plus faciles à interpréter pour les applications.

## B. Création de la base de données




Ce schéma représente les différentes entités et de leurs relations dans une base de données pour la gestion des données immobilières et urbaines à Paris.

Dans les étapes à venir, nous nous concentrerons sur l'optimisation des données pour en extraire leur signification la plus pertinente possible. À ce stade, il est important de noter que notre base de données actuelle ne satisfait pas encore pleinement nos besoins. Pour remédier à cela, nous prévoyons d'intégrer les exemples de cardinalités dans notre future base de données cible. Cette démarche permettra de structurer efficacement nos données et de les rendre exploitables pour nos analyses ultérieures. En optimisant la qualité et la cohérence de nos données, nous nous assurons de tirer le meilleur parti de nos ressources disponibles pour atteindre nos objectifs analytiques.

### 1. valeurs_foncieres_paris_2022 et arrondissements:

- Chaque arrondissement peut avoir plusieurs valeurs foncières
- Chaque valeur foncière a un arrondissement

Cardinalité : Une valeur foncière peut avoir plusieurs arrondissements (N:1)

### 2. arrondissements et quartiers_administratifs :

- Chaque arrondissement peut avoir plusieurs quartiers administratifs.
- Chaque quartier administratif appartient à un seul arrondissement.

Cardilatité : Un arrondissement peut avoir plusieurs quartiers administratifs (1:N)

### 3. quartiers_ardinistratifs et logement_encadrement_des_loyers :

- Chaque entrée dans la table logement_encadrement_des_loyers est liée à un seul quartier administratif de Paris, identifié par son numéro de quartier. Donc, chaque enregistrement de logement_encadrement_des_loyers est associé à un seul quartier.
- Chaque quartier administratif peut avoir plusieurs enregistrements dans la table logement_encadrement_des_loyers, car plusieurs logements peuvent être situés dans le même quartier.

Cardinalité : Un quartier administratif peut avoir plusieurs logements avec des loyers encadrés (1:N).

## C. Création de la table et chargement de données

Nous avons pris la décision d'utiliser Snowflake comme solution pour stocker notre base de données. Snowflake est une plateforme de data warehousing moderne et hautement évolutive, qui offre de bonnes performances et une flexibilité non négligeable pour gérer et analyser des volumes massifs de données. En choisissant Snowflake, nous nous assurons d'avoir une infrastructure solide et fiable pour héberger nos données. Cette décision nous permettra de tirer pleinement parti des fonctionnalités de Snowflake pour optimiser la gestion et l'analyse de nos données, tout en garantissant leur disponibilité, leur sécurité et leur évolutivité à long terme.

```
CREATE TABLE "ANALYSE_IMMO_PARIS_2022"."PUBLIC"."VALEURS_FONCIERES" (
DATE_MUTATION DATE , NATURE_MUTATION VARCHAR , VALEUR_FONCIERE NUMBER(38, 1) ,
ADRESSE_NUMERO NUMBER(38, 1) , ADRESSE_NOM_VOIE VARCHAR , Voie VARCHAR ,
CODE_POSTAL NUMBER(38, 1) , COMMUNE VARCHAR , Code_departement NUMBER(38, 0) ,
Code_commune NUMBER(38, 0) , SURFACE_CARREZ VARCHAR , TYPE_LOCAL VARCHAR ,
SURFACE_REELLE_BATI NUMBER(38, 1) , NOMBRE_PIECES_PRINCIPALES NUMBER(38, 1) ,
SURFACE_TERRAIN NUMBER(38, 1) );

CREATE TEMP FILE FORMAT
"ANALYSE_IMMO_PARIS_2022"."PUBLIC"."temp_file_format_2024-04-29T09:53:02.537Z"
    TYPE=CSV
    SKIP_HEADER=1
    FIELD_DELIMITER=','
    TRIM_SPACE=TRUE
    FIELD_OPTIONALLY_ENCLOSED_BY='"'
    REPLACE_INVALID_CHARACTERS=TRUE
    DATE_FORMAT=AUTO
    TIME_FORMAT=AUTO
    TIMESTAMP_FORMAT=AUTO;

COPY INTO "ANALYSE_IMMO_PARIS_2022"."PUBLIC"."VALEURS_FONCIERES"
FROM (SELECT $1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11, $12, $13, $14, $15
                                                                    FROM
'@"ANALYSE_IMMO_PARIS_2022"."PUBLIC"."__snowflake_temp_import_files__"')
FILES = ('2024-04-29T09:52:52.767Z/valeurs_foncieres_paris_2022.csv')
FILE_FORMAT                                                          =
'"ANALYSE_IMMO_PARIS_2022"."PUBLIC"."temp_file_format_2024-04-29T09:53:02.537Z
"'
ON_ERROR=ABORT_STATEMENT
```

Ce script SQL a pour objectif de créer une table dans une base de données nommée "ANALYSE_IMMO_PARIS_2022", plus précisément dans le schéma "PUBLIC". Cette table, nommée "VALEURS_FONCIERES", est destinée à contenir des données sur les transactions immobilières à Paris en 2022.

La structure de la table est définie avec différentes colonnes représentant les attributs des transactions immobilières, tels que la date de mutation, la nature de la mutation, la valeur foncière, l'adresse, les caractéristiques du bien (surface, type de local, nombre de pièces, etc.).

Ensuite, un format de fichier temporaire est défini pour spécifier la manière dont les données CSV seront traitées lors de leur importation dans la table. Cela inclut des détails tels que le délimiteur de champ, le format des dates et heures, et la manière de gérer les valeurs invalides.

Enfin, la commande COPY INTO est utilisée pour charger les données depuis un fichier CSV spécifique (probablement extrait d'une source externe) dans la table "VALEURS_FONCIERES". Cette commande sélectionne les données du fichier CSV en utilisant le format de fichier temporaire défini précédemment, et spécifie également comment gérer les erreurs éventuelles lors du chargement des données. Cette approche permet d'automatiser le processus d'importation des données dans la base de données, assurant ainsi leur intégrité et leur cohérence.

## D. Différentes requêtes utilisées

### 1. Ajout d'une colonne ID

```
-- Créer une séquence
CREATE SEQUENCE encadrement_des_loyers_seq;
-- Ajouter une colonne à la table
ALTER TABLE ENCADREMENT_DES_LOYERS
ADD COLUMN ID_ENCADREMENT_DES_LOYERS INT;
-- Mettre à jour les valeurs de la colonne avec les valeurs de la séquence
UPDATE ENCADREMENT_DES_LOYERS
SET ID_ENCADREMENT_DES_LOYERS = encadrement_des_loyers_seq.NEXTVAL;
-- Définir la colonne comme clé primaire si nécessaire
ALTER TABLE ENCADREMENT_DES_LOYERS
ADD CONSTRAINT PK_ENCADREMENT_DES_LOYERS PRIMARY KEY
(ID_ENCADREMENT_DES_LOYERS);
```

Cette instruction SQL crée une séquence pour générer des valeurs séquentielles, ajoute une nouvelle colonne à une table existante et met à jour cette colonne avec les valeurs générées par la séquence. Enfin, elle définit la colonne nouvellement ajoutée comme clé primaire si nécessaire.

### 2. Changement de nom de colonne

```
-- Changement de nom de colonne
ALTER TABLE `QUARTIERS_ADMINISTRATIFS`
CHANGE`IDENTIFIANT_SEQUENTIEL_DU_QUARTIERQU`
`IDENTIFIANT_SEQUENTIEL_DU_QUARTIER` INT(11) NOT NULL,
CHANGE `C_QU` `NUMERO_QUARTIER` INT(11) NULL DEFAULT NULL,
CHANGE `C_QUINSEE` `NUMERO_QUARTIER_INSEE` INT(11) NULL DEFAULT NULL,
CHANGE `L_QU` `NOM_QUARTIER` VARCHAR(100) CHARACTER SET utf8 COLLATE
utf8_unicode_ci NULL DEFAULT NULL,
CHANGE `C_AR` `NUMERO_ARRONDISSEMENT` INT(11) NULL DEFAULT NULL,
CHANGE `N_SQ_AR` `IDENTIFIANT_SEQUENTIEL_ARRONDISSEMENT` INT(11) NULL DEFAULT
NULL;
```

Cette instruction SQL effectue plusieurs changements de nom de colonne dans la table QUARTIERS_ADMINISTRATIFS. Ces changements de nom de colonne peuvent être effectués pour clarifier la structure de la base de données, rendre les noms de colonnes plus significatifs et cohérents, ou pour suivre des conventions de dénomination spécifiques.

```sql
ALTER TABLE ANALYSE_IMMO_PARIS_2022.PUBLIC.ARRONDISSEMENTS
ADD CODE_POSTAL VARCHAR(5);
UPDATE ANALYSE_IMMO_PARIS_2022.PUBLIC.ARRONDISSEMENTS
SET CODE_POSTAL = CONCAT('750', LPAD(NUMERO_ARRONDISSEMENT, 2, '0'));

ALTER TABLE ENCADREMENT_DES_LOYERS
ADD (NUMERO_ARRONDISSEMENT NUMBER(2));
UPDATE ENCADREMENT_DES_LOYERS
SET NUMERO_ARRONDISSEMENT = TO_NUMBER(SUBSTR(NUMERO_INSEE_DU_QUARTIER, 3, 2));
```

Le premier code SQL ajoute une nouvelle colonne nommée "CODE_POSTAL" à la table "ARRONDISSEMENTS" du schéma "PUBLIC" de la base de données "ANALYSE_IMMO_PARIS_2022". Ensuite, il met à jour les valeurs de la colonne "CODE_POSTAL" en concaténant le préfixe '750' avec le numéro d'arrondissement, en s'assurant qu'il est formaté avec deux chiffres grâce à la fonction LPAD pour ajouter des zéros à gauche si nécessaire.

Le deuxième code SQL ajoute une nouvelle colonne appelée "NUMERO_ARRONDISSEMENT" à la table "ENCADREMENT_DES_LOYERS", de type NUMBER(2). Ensuite, il met à jour les valeurs de la colonne "NUMERO_ARRONDISSEMENT" en extrayant les deux derniers chiffres du numéro INSEE du quartier et les convertissant en un nombre.

Ces opérations visent à enrichir les tables existantes avec de nouvelles informations, telles que les codes postaux et les numéros d'arrondissement, en utilisant des valeurs existantes dans les tables ou des transformations sur ces valeurs.

# 3. Construction du pipeline ETL / ELT

## A. Chargement et dénormalisation

Pour notre projet, il a été choisi de dénormaliser les données en utilisant un schéma en constellation pour faciliter l'analyse et le reporting. Dans un schéma en constellation, nous avons toujours une table centrale de faits qui contient les mesures principales, mais au lieu d'avoir un seul ensemble de tables de dimensions directement liées à la table de faits, nous avons plusieurs ensembles de tables de dimensions qui peuvent se chevaucher ou être partagées entre différentes tables de faits.

### 1. Dénormalisation

Nous créons une première table centrale de faits contenant les mesures principales, et nous joignons les tables de dimensions appropriées pour enrichir les données. Par exemple, pour une analyse des ventes immobilières, nous pouvons avoir une table de faits contenant les ventes avec des colonnes telles que la valeur de la propriété, la date de vente, etc. Nous joignons ensuite les tables de dimensions telles que les arrondissements, les quartiers, etc., pour ajouter des informations contextuelles à ces ventes.

### 2. Chargement dans le Système Cible

Une fois que les données sont dénormalisées, nous les chargeons dans le système cible. Nous utilisons des requêtes SQL pour insérer les données dans les tables appropriées du schéma en étoile.

Voici un exemple simplifié du chargement des données dénormalisées dans une base de données SQL, en utilisant des instructions SQL d'insertion dans les tables correspondantes.

```sql
CREATE SCHEMA IDENTIFIER('"ANALYSE_IMMO_PARIS_2022"."SCHEMA"') COMMENT = ''

CREATE TABLE ANALYSE_IMMO_PARIS_2022.SCHEMA.FAIT_VALEURS_FONCIERES AS
SELECT * FROM ANALYSE_IMMO_PARIS_2022.PUBLIC.VALEURS_FONCIERES;

ALTER TABLE ANALYSE_IMMO_PARIS_2022.SCHEMA.FAIT_VALEURS_FONCIERES
ADD ID_ARRONDISSEMENT NUMBER(38,0);

UPDATE ANALYSE_IMMO_PARIS_2022.SCHEMA.FAIT_VALEURS_FONCIERES F
SET F.ID_ARRONDISSEMENT = A.IDENTIFIANT_SEQUENTIEL_ARRONDISSEMENT
FROM ANALYSE_IMMO_PARIS_2022.SCHEMA.DIM_ARRONDISSEMENTS A
WHERE F.NUMERO_ARRONDISSEMENT_INSEE = A.NUMERO_ARRONDISSEMENT_INSEE;
```

Ici, on crée un nouveau schéma, duplique une table existante dans ce schéma, ajoute une nouvelle colonne à cette table, puis met à jour les valeurs de cette colonne en fonction des données d'une autre table présente dans le même schéma. Cela permet de structurer les données et de les enrichir avec des informations supplémentaires pour faciliter les analyses ultérieures.

```sql
-- Création de la table de dimension DIM_TYPE_LOCAL
CREATE TABLE ANALYSE_IMMO_PARIS_2022.SCHEMA.DIM_TYPE_LOCAL AS
SELECT DISTINCT TYPE_LOCAL, ID_VALEURS_FONCIERES
FROM ANALYSE_IMMO_PARIS_2022.SCHEMA.FAIT_VALEURS_FONCIERES;

-- Renommage ID_DATE côté DIM_TYPE_LOCAL
ALTER TABLE ANALYSE_IMMO_PARIS_2022.SCHEMA.DIM_TYPE_LOCAL
RENAME COLUMN ID_VALEURS_FONCIERES TO ID_TYPE_LOCAL;

-- Création de la colonne ID_TYPE_LOCAL
ALTER TABLE ANALYSE_IMMO_PARIS_2022.SCHEMA.FAIT_VALEURS_FONCIERES
ADD COLUMN ID_TYPE_LOCAL NUMBER(38,0);

UPDATE ANALYSE_IMMO_PARIS_2022.SCHEMA.FAIT_VALEURS_FONCIERES
SET ID_TYPE_LOCAL = ID_VALEURS_FONCIERES;
```

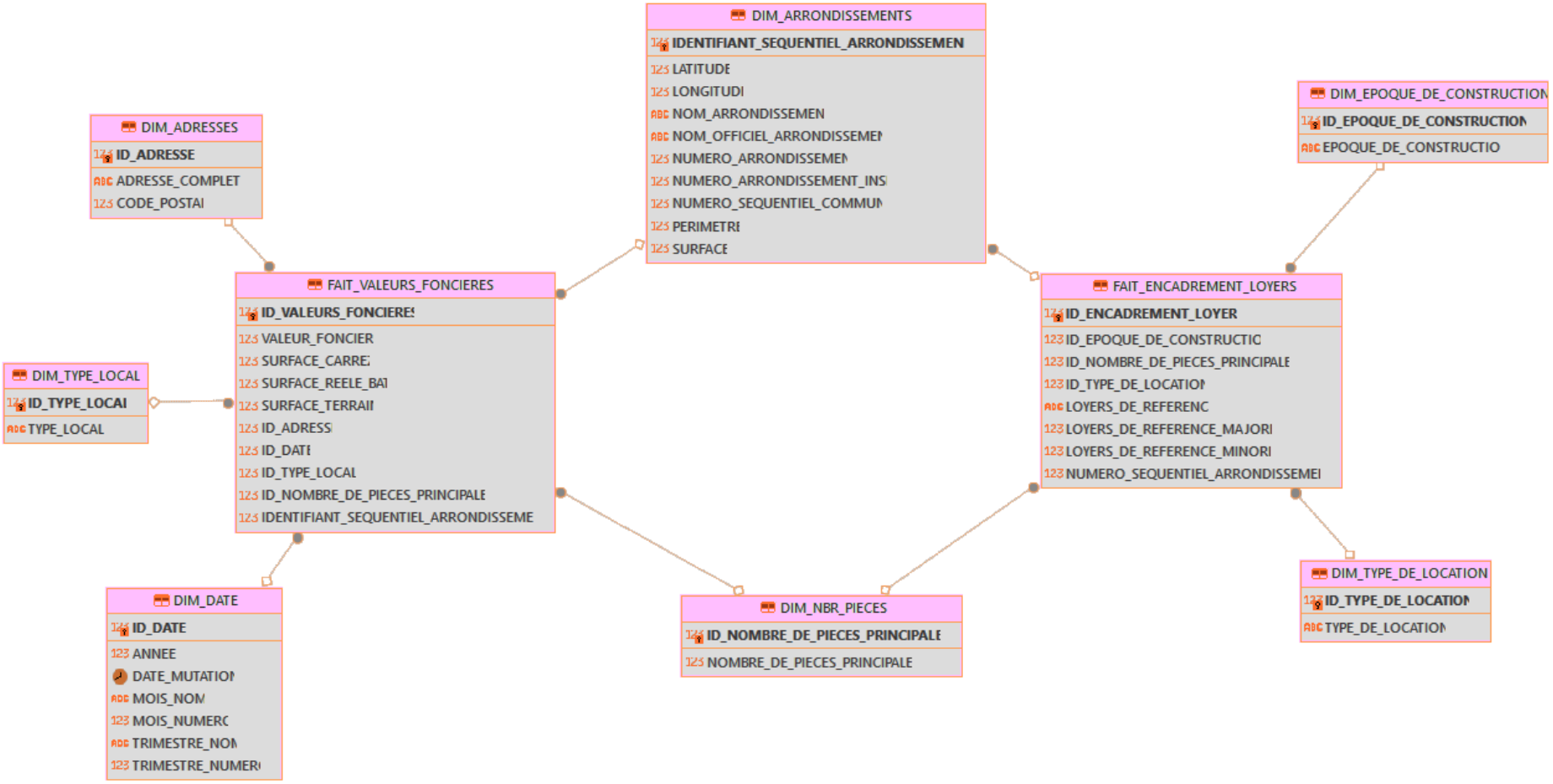
Une nouvelle table de dimension nommée "DIM_TYPE_LOCAL" est créée dans le schéma "SCHEMA" de la base de données "ANALYSE_IMMO_PARIS_2022". Les données de cette table sont obtenues en sélectionnant de manière distincte les valeurs de la colonne "TYPE_LOCAL" de la table de faits "FAIT_VALEURS_FONCIERES", ainsi que les identifiants des valeurs foncières associées.

**Renommage de la colonne ID_VALEURS_FONCIERES** : La colonne "ID_VALEURS_FONCIERES" dans la table "DIM_TYPE_LOCAL" est renommée en "ID_TYPE_LOCAL". Cela harmonise la structure avec la convention de dénomination des colonnes dans les tables de dimensions.

**Ajout de la colonne ID_TYPE_LOCAL** : Une nouvelle colonne "ID_TYPE_LOCAL" de type NUMBER(38,0) est ajoutée à la table de faits "FAIT_VALEURS_FONCIERES". Cette colonne servira à stocker les identifiants des types locaux associés aux valeurs foncières.

**Mise à jour des valeurs** : Les valeurs de la colonne "ID_TYPE_LOCAL" dans la table de faits "FAIT_VALEURS_FONCIERES" sont mises à jour en copiant simplement les valeurs de la colonne "ID_VALEURS_FONCIERES". Cela suppose que les identifiants des valeurs foncières sont utilisés pour identifier les types locaux dans la table de dimension "DIM_TYPE_LOCAL".

Nous répétons ce processus pour chaque table de faits et de dimensions, en ajustant les requêtes en fonction de la structure de chaque table et des données dénormalisées.


DIM_ARRONDISSEMENTS
IDENTIFIANT_SEQUENTIEL_ARRONDISSEMEN
LATITUDE
LONGITUDI
NOM_ARRONDISSEMEN
NOM_OFFICIEL_ARRONDISSEMEI
NUMERO_ARRONDISSEMEI
NUMERO_ARRONDISSEMENT_INS
NUMERO_SEQUENTIEL_COMMUI
PERIMETRI
SURFACE
DIM_EPOQUE_DE_CONSTRUCTION
ID_EPOQUE_DE_CONSTRUCTION
EPOQUE_DE_CONSTRUCTIO
DIM_ADRESSES
ID_ADRESSE
ADRESSE_COMPLET
CODE_POSTAI
FAIT_VALEURS_FONCIERES
ID_VALEURS_FONCIERE
VALEUR_FONCIER
SURFACE_CARRE
SURFACE_REELE_BA
SURFACE_TERRAI
ID_ADRESS
ID_DATE
ID_TYPE_LOCAL
ID_NOMBRE_DE_PIECES_PRINCIPALE
IDENTIFIANT_SEQUENTIEL_ARRONDISSEME
FAIT_ENCADREMENT_LOYERS
ID_ENCADREMENT_LOYER
ID_EPOQUE_DE_CONSTRUCTIC
ID_NOMBRE_DE_PIECES_PRINCIPALE
ID_TYPE_DE_LOCATIO
LOYERS_DE_REFERENC
LOYERS_DE_REFERENCE_MAJORI
LOYERS_DE_REFERENCE_MINORI
NUMERO_SEQUENTIEL_ARRONDISSEMEI
DIM_TYPE_LOCAL
ID_TYPE_LOCAI
TYPE_LOCAL
DIM_DATE
ID_DATE
ANNEE
DATE_MUTATIO
MOIS_NOM
MOIS_NUMERC
TRIMESTRE_NO
TRIMESTRE_NUMER
DIM_NBR_PIECES
ID_NOMBRE_DE_PIECES_PRINCIPALE
NOMBRE_DE_PIECES_PRINCIPALE
DIM_TYPE_DE_LOCATION
ID_TYPE_DE_LOCATIO
TYPE_DE_LOCATIO