CY Université

Base de Donnée
Projet

Etudes et statistiques sur la population française entre 1968 et 2020

Élèves :
Florian Grolleau
Léandre Testart
William Smith
Romain Jaffuel

Enseignant :
Renaud Verin

16 décembre 2024

Table des matières

1 Introduction

La population française évolue au cours des décennies, et significativement ces dernières décennies, que ce soit en termes de répartition géographique, de croissance ou même de densité. Entre 1968 et 2020, de nombreux recensements de la population avec leur répartition géographique ont eu lieu, nous donnant une source précieuse pour analyser et étudier ces changements et ces dynamiques.

Ce projet vise à exploiter ces données pour générer une base de données robuste et structurée. L'objectif est de pouvoir créer des représentations graphiques issues de cette base de données, afin de pouvoir mettre en évidence l'évolution de la population, les croissances des villes et leurs densités. Grâce à des requêtes sous SQL et des graphiques sous Python, plus précisément avec Matplotlib.

Dans ce rapport, nous verrons les étapes de la méthodologie : du MCD et MLD au traitement de la base de données. Puis nous aborderons les résultats obtenus et les limites rencontrées. Et nous terminerons sur notre ressenti sur ce projet et notre avis.

2 Méthodologie

Dans cette partie, nous évoquerons dans un premier temps le MCD et le MLD, les limites rencontrées et leurs difficultés. Puis, nous ferons de même avec le traitement de la base de données.

2.1 MCD et MLD

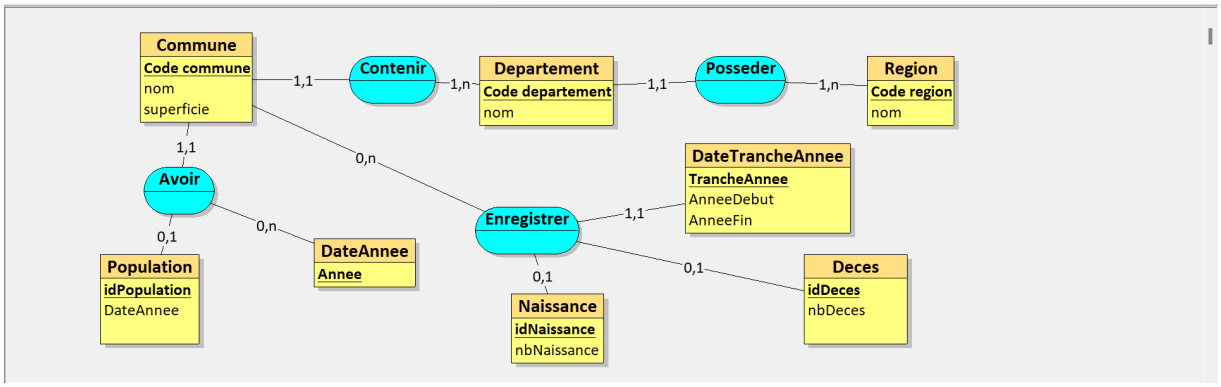

FIGURE 1 – MCD du projet

FIGURE 2 – MLD du projet

2.1.1 Explication du MCD et MLD

Le MCD comporte plusieurs entités principales avec leurs attributs clés : -Commune, avec Code commune comme clé primaire, nom et superficie. -Departement avec Code departement comme clé primaire et nom. -Region avec Code region comme clé primaire et nom. -Population avec idPopulation comme clé primaire et DateAnnee. -DateAnnee avec Annee comme clé primaire. -Naissance avec idNaissance comme clé primaire et nbNaissance. -Deces avec idDeces comme clé primaire et nbDeces. -DateTrancheAnnee avec TrancheAnnee comme clé primaire, AnneeDebut et AnneeFin.

Le MCD met aussi en évidence ses relations : -Contenir, entre Commune et Departement où une commune appartient à un département et où un département contient une ou plusieurs communes (cardinalité 1,1 côté Commune et 1,n côté Departement). -Posseder, entre Departement et Region où un département appartient à une région et où une région contient un ou plusieurs départements (cardinalité 1,n côté Region et 1,1 côté Departement). -Avoir, entre Commune et Population où une commune a une population (cardinalité 1,1 côté Commune et 0,1 côté Population). -Enregistrer, entre DateAnnee, Naissance, Deces, Population et DateTrancheAnnee, et où une Commune possède obligatoirement une seule "Enregistrer" qui regroupe les autres.

Le MLD a été fait à partir du MCD et regroupe tout ceci sous forme de liste pour une meilleure compréhension avec les clés étrangéres et le type des attributs (ici INT ou VARCHAR(50)).

2.1.2 Limite et contrainte du MCD et MLD

Durant la conception du MCD et MLD, nous avons été confrontés à plusieurs limites et contraintes. L'une des premières questions que nous avons pu avoir à partir du MCD et MLD est d'avoir des entités naissance et décès individuels pour pouvoir affiner les données. Nous avons dû aussi réfléchir à faire le MCD d'une manière à être utilisable pour SQL. Mais l'une des réelles difficultés était sur comment relier Commune et Population tout en prenant en compte les Naissances, Décès et l'Année afin d'assurer une certaine cohérence, c'est là où l'idée de créer la relation Enregistrer est apparue pour tout combiner. Le MLD, quant à lui, n'a pas été difficile, car il a été généré automatiquement à partir du MCD. Nous avons effectué une relecture et une analyse pour savoir si le MLD était cohérent.

2.2 Traitement de la base de données

2.2.1 Explication certains passage du code

```python
def fix_and_load_csv(csv_path, table_name):
    """Corrige un fichier CSV s'il est mal formé et l'importe dans une table SQL."""
    try:
        # Essayer de lire le fichier CSV avec un délimiteur ';'
        try:
            df = pd.read_csv(csv_path, delimiter=';', encoding='utf-8')
            print(f"Fichier chargé correctement avec ';' : {csv_path}")
        except Exception as e:
            print(f"Erreur lors de la lecture avec ';' pour {csv_path}: {e}")
            # Si ';' échoue, essayer avec une virgule ','
            df = pd.read_csv(csv_path, delimiter=',', encoding='utf-8')
            print(f"Fichier chargé correctement avec ',' : {csv_path}")

        # Vérifiez si le DataFrame est bien formé
        if df.shape[1] <= 1:
            raise ValueError(f"Le fichier {csv_path} semble mal formé. Vérifiez les délimiteurs ou la structure.")

        # Raccourcir les noms des colonnes si nécessaire
        max_column_length = 64
        df.columns = [col[:max_column_length] for col in df.columns]

        # Réenregistrer le fichier corrigé (facultatif)
        corrected_path = csv_path.replace(".csv", "_corrected.csv")
        df.to_csv(corrected_path, index=False, sep=';', encoding='utf-8')
        print(f"Fichier corrigé enregistré sous : {corrected_path}")

        # Importer dans la base MySQL
        table_name = table_name.lower()
        engine = create_engine(f'mysql+mysqlconnector://{user}:{password}@{host}/{database}')
        df.to_sql(table_name, engine, if_exists='replace', index=False)
        print(f"Table '{table_name}' créée avec succès dans la base '{database}'.")
    except Exception as e:
        print(f"Erreur lors du traitement de {csv_path} : {e}")

# Étape 1 : Créer la base de données si elle n'existe pas
create_database_if_not_exists()

# Étape 2 : Charger et corriger chaque CSV
for table_name, csv_path in csv_files.items():
    fix_and_load_csv(csv_path, table_name)
```

FIGURE 3 – Code traitement de la base de données

Ce script Python permet de corriger et charger des fichiers CSV dans une base de données MySQL. Il gère les erreurs potentielles liées à la structure du fichier CSV, modifie les fichiers pour garantir leur compatibilité, et importe les données corrigées dans une table SQL.

La fonction "fix and load csv(csv path, table name)" lit le fichier CSV en utilisant le délimiteur " ;". Si cette tentative échoue, une seconde lecture est faite avec le délimiteur ",". Permettant de s'adapter à différents formats de fichiers. Si aucune des deux lectures ne fonctionne, le code ce stop et signale un problème.

Une fois le fichier lu, le script vérifie que la structure du fichier est correcte en analysant le DataFrame. Si le fichier contient une seule colonne ou moins, le code ce stop en indiquant un problème potentiel avec les délimiteurs ou la structure.

Pour éviter des erreurs liées à MySQL, les noms des colonnes sont limités à 64 caractères. Cette limite est imposée pour garantir la compatibilité avec les contraintes de la base de données.

Puis il sauvegarde une copie du fichier CSV avec un nom modifié, en ajoutant corrected au nom initial. Cette étape est facultative mais utile pour conserver une version propre du fichier.

Une fois le fichier corrigé et sauvegardé, les données sont importées dans une table MySQL. La table est créée ou remplacée si elle existe déjà. Cette opération est réalisée à l'aide de la fonction df.to sql() qui permet de charger un DataFrame dans une base de données. Une connexion MySQL est établie avec la fonction create engine() en utilisant

les paramètres d'authentification (nom d'utilisateur, mot de passe, hôte, nom de la base de données).

```sql

    -- Créer table Departement
CREATE TABLE Departement (
    CodeDepart  VARCHAR(50) NOT NULL,
    NomDepart VARCHAR(255) NOT NULL,
    CodeRegion INT NOT NULL,
    PRIMARY KEY (CodeDepart),
    FOREIGN KEY (CodeRegion) REFERENCES Region(CodeRegion)
);

    -- Insertion des données dans Departement
INSERT INTO Departement (CodeDepart, NomDepart, CodeRegion)
SELECT DISTINCT CodeDepart, NomDepart, CodeRegion
FROM populationdepartementsfrance;

    -- Vérification des données dans Departement et leurs régions associées
SELECT d.CodeDepart, d.NomDepart, r.NomRegion
FROM Departement d
JOIN Region r ON d.CodeRegion = r.CodeRegion;

```

FIGURE 4 – Code traitement de la base de données, table SQL

Ce script SQL permet de créer une table Departement, d'y insérer des données provenant d'une autre table et enfin de vérifier ces données en les liant à une table existante Region.

La première partie du script crée une nouvelle table Departement pour stocker les informations sur les départements. La colonne CodeDepart est une chaîne de caractères pouvant contenir jusqu'à 50 caractères. Elle ne peut pas être vide NOT NULL et sert de clé primaire pour garantir que chaque département a un identifiant unique. La colonne NomDepart contient le nom du département, avec une limite de 255 caractères, et ne peut pas être vide NOT NULL. La colonne CodeRegion stocke un entier représentant la région à laquelle appartient le département. Une clé étrangère est ajoutée sur CodeRegion pour que chaque valeur dans cette colonne corresponde à une région existante dans la table Region.

Ensuite, le script insère des données dans la table Departement qui proviennent de la table populationdepartementsfrance.La fonction INSERT INTO spécifie que les données doivent être insérées dans les colonnes CodeDepart, NomDepart et CodeRegion de la table Departement. La fonction SELECT DISTINCT extrait uniquement les lignes sans doublon de la table populationdepartementsfrance pour les colonnes CodeDepart, NomDepart et CodeRegion.

La dernière partie du script permet de vérifier les données dans la table Departement avec la table Region. La commande FROM Departement d, se combine avec la commande JOIN Region r relie la table en utilisant la colonne CodeRegion comme condition de jointure (ON d.CodeRegion = r.CodeRegion). Permettant de joindre le nom de la région

correspondant à chaque département.

2.2.2 Limite et contrainte du traitement

Durant le traitement de la base de données, nous avons eu quelques problématiques. L'un des problèmes majeurs a été la génération de la base SQL à cause des problèmes de connexions entre Spyder et SQL, rendant l'intégration complexe. Nous avons donc choisi de ne pas utiliser Spyder, et de nous orienter vers Virtual Studio Code. Une question s'est posée quant au fait de regrouper tous les CSV dans un seul et même CSV, mais ceci a été écarté car trop complexe et difficile par la suite à étudier. Par la suite, nous avons rencontré un problème avec le chargement des CSV avec les séparateurs ";", ce qui les rend inexploitables dans SQL. Pour contourner ce problème, nous avons préféré le traiter sous Python, langage dans lequel nous sommes plus à l'aise, avant de les importer sous forme de tables SQL. Une autre contrainte a été l'adaptation du MCD après la génération des tables SQL pour une meilleure cohérence. Enfin, nous avons dû renommer les noms de colonnes pour une meilleure clarté et une meilleure compréhension des données.

3 Résultat

Dans cette partie, nous évoquerons certains passages de code. Puis nous listerons tous les graphiques attendus et nous parlerons des limites et contraintes rencontrées.

3.1 Explication de certains passage

```sql
-- Question 5a : 10 villes ayant cru le plus de 1968 à 2020
SELECT
    c.LIB_MOD AS Ville,
    (p.P20_POP - p.D68_POP) AS Evolution
FROM Commune c
JOIN Population p ON c.CODGEO = p.CODGEO
ORDER BY Evolution DESC
LIMIT 10;
```

Figure 5 – Le SQL correspondant à la question 5a

Ce script SQL sélectionne le nom des villes, récupéré depuis la colonne LIB MOD de la table Commune et nomme en Ville, ainsi que l'évolution de la population : la différence entre la population de 2020 P20 POP et celle de 1968 (68 POP de la table Population et nommé Evolution. Ensuite, une jointure est réalisée entre les tables Commune et Population en utilisant la colonne CODGEO. Permet de relier les noms des villes avec les données de population. Les résultats sont triés par ordre décroissant ORDER BY Evolution DESC de l'évolution de la population afin de placer en tête les villes ayant connu la plus forte

croissance démographique. Enfin, la clause LIMIT 10 permet de limiter le résultat aux 10 premières villes.

```python
import pandas as pd
import matplotlib.pyplot as plt
from sqlalchemy import create_engine

# Connexion MySQL
engine = create_engine('mysql+mysqlconnector://root@localhost:3306/Population')

# Requête SQL pour les 10 villes avec la plus forte croissance
query = """
SELECT
    c.LIB_MOD AS nom,
    (p.P20_POP - p.D68_POP) AS croissance
FROM Commune c
JOIN Population p ON c.CODGEO = p.CODGEO
ORDER BY croissance DESC
LIMIT 10;
"""

# Exécuter la requête et charger les données dans pandas
df = pd.read_sql(query, engine)

# Vérification des données
print(df)

# Création du graphique
plt.figure(figsize=(12, 6))
plt.bar(df['nom'], df['croissance'], color='skyblue')

# Ajouter les labels et un titre
plt.title("Top 10 des villes ayant connu la plus forte croissance démographique (1968-2020)", fontsize=16)
plt.xlabel("Villes", fontsize=12)
plt.ylabel("Croissance de la population", fontsize=12)
plt.xticks(rotation=45, ha='right')

# Définir le chemin de sauvegarde du graphique
chemin = 'C:/Users/willi/OneDrive/Documents/SQLBdd/top_10_villes_croissance.jpeg'

# Enregistrer le graphique au format JPEG
plt.savefig(chemin, format='jpeg')
# Afficher le graphique
plt.tight_layout()
plt.show()
```

FIGURE 6 – Le graphe sous python correspondant à la question 5a

Ce code Python établie une connexion à SQL grâce à la bibliothèque ‘sqlalchemy‘. Une requête SQL est ensuite exécutée pour extraire les 10 villes ayant connu la plus forte croissance démographique entre 1968 et 2020.. Les résultats de cette requête sont chargés dans un DataFrame pandas pour faciliter leur manipulation et affichage, df = pd. read sql(query, engine). Par la suite, le code utilise matplotlib pour créer un graphique à barres qui représente les 10 villes et leur taux de croissance, plt. bar(df [’nom’], df [’croissance’], color=’skyblue’) . Le graphique est personnalisé avec un titre clair, des étiquettes pour les axes, plt.title("Top 10 des villes ayant connu la plus forte croissance démographique (1968-2020)", fontsize=16) plt.xlabel("Villes", fontsize=12), plt-ylabel("Croissance de la population", fontsize=12). Enfin, le graphique est enregistré au format JPEG ce qui permet de conserverles résultats, plt. savefig (chemin, format=’jpeg’).

3.2 Limite et contrainte

Les contraintes du projet étaient la matière d’origine des tables SQL, les csv avec les populations, les régions, départements et villes. Les csv sont impropres à une utilisation directe sur MySQL. Il faut regarder un par un les trois csv voire leurs défauts, leurs contenues puis réfléchir à comment faire nos tables en sachant que parfois une colonne dans un csv n’a pas le même nom dans un autre csv ou parfois les données ne sont pas classées par colonnes. L’autre contrainte c’était de bien réussir la base de données et les tables qui vont permettre de répondre aux questions. Si elle est mal faite devoir recommencer une partie de la base de données est parfois obligatoires. Il faut donc faire

très attention quand on fait nos tables on doit penser à leurs futures efficacités pour répondre aux questions

Nous avons eu aussi des difficultés avec les DROM-COM, où le code postal est en 5 chiffres mais cette fois-ci les 3 premiers chiffres représentent le département, il a fallu alors rajouter un code pour lire ceci.

```sql
-- Correction affiché les villes et départements d'outre-mer
USE Population;
SELECT c.LIB_MOD AS Commune, c.CODGEO AS CodeCommune, d.NomDepart AS Departement,d.CodeDepart AS CodeDepartement
FROM Commune c
JOIN Departement d
    ON d.CodeDepart = LEFT(c.CODGEO, 3)
WHERE LENGTH(c.CODGEO) = 5
ORDER BY d.CodeDepart, c.LIB_MOD;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Commune	CodeCommune	Departement	CodeDepartement
Anse-Bertrand	97102	Guadeloupe	971
Baie-Mahault	97103	Guadeloupe	971
Baillif	97104	Guadeloupe	971
Basse-Terre	97105	Guadeloupe	971
Bouillante	97106	Guadeloupe	971
Capesterre-Belle-Eau	97107	Guadeloupe	971
Capesterre-de-Marie-Galante	97108	Guadeloupe	971
Deshaies	97111	Guadeloupe	971
Gourbeyre	97109	Guadeloupe	971
Goyave	97114	Guadeloupe	971
Grand-Bourg	97112	Guadeloupe	971
La Désirade	97110	Guadeloupe	971
Lamentin	97115	Guadeloupe	971

FIGURE 7 – Code SQL pour liée les DROM-COM avec leur département

3.3 Requêtes des résultats

Ville	Departement	Region	Population_2020
L'Abergement-Clémenciat	Ain	Auvergne-Rhône-Alpes	806
L'Abergement-de-Varey	Ain	Auvergne-Rhône-Alpes	262
Ambérieu-en-Bugey	Ain	Auvergne-Rhône-Alpes	14288
Ambérieux-en-Dombes	Ain	Auvergne-Rhône-Alpes	1782
Ambléon	Ain	Auvergne-Rhône-Alpes	113
Ambronay	Ain	Auvergne-Rhône-Alpes	2827
Ambutrix	Ain	Auvergne-Rhône-Alpes	768
Andert-et-Condon	Ain	Auvergne-Rhône-Alpes	324
Anglefort	Ain	Auvergne-Rhône-Alpes	1101
Apremont	Ain	Auvergne-Rhône-Alpes	368
Aranc	Ain	Auvergne-Rhône-Alpes	327
Arandas	Ain	Auvergne-Rhône-Alpes	143

FIGURE 8 – Requête sur Liste des populations en 2020 avec le nom de ville, département, région

Population_1968_en_million	Population_2020_en_million	Evolution_en_million
54.8057	70.7006	15.8949

FIGURE 9 – Requête sur l'Évolution de la population française de 1968 à 2020.

	Departement	Population_2020
▶	Ain	657856
	Aisne	529374
	Allier	335628
	Alpes-de-Haute-Provence	165451
	Hautes-Alpes	140605
	Alpes-Maritimes	1097410
	Ardèche	329325
	Ardennes	269701
	Ariège	153954
	Aube	311435
	Aude	375217
	Aveyron	279554

FIGURE 10 – Requête sur Liste des populations en 2020 par département avec leurs noms

	Region	Population_2020
▶	Auvergne-Rhône-Alpes	8078652
	Hauts-de-France	5997734
	Provence-Alpes-Côte d'Azur	5098666
	Grand Est	5562651
	Occitanie	5973969
	Normandie	3325522
	Nouvelle-Aquitaine	6033952
	Centre-Val de Loire	2574863
	Bourgogne-Franche-Comté	2801695
	Bretagne	3373835
	Corse	343701
	Pays de la Loire	3832120

FIGURE 11 – Requête sur Liste des populations en 2020 par région avec leurs noms

	Arrondissement	Population
	Paris 1er Arrondissement	16030
	Paris 2e Arrondissement	21130
	Paris 3e Arrondissement	33402
	Paris 4e Arrondissement	29064
	Paris 5e Arrondissement	57380
	Paris 6e Arrondissement	39625
	Paris 7e Arrondissement	48520
	Paris 8e Arrondissement	35631
	Paris 9e Arrondissement	60168
	Paris 10e Arrondissement	83459
	Paris 11e Arrondissement	144292
	Paris 12e Arrondissement	140311
	Paris 13e Arrondissement	177833
	Paris 14e Arrondissement	133967
	Paris 15e Arrondissement	229472
	Paris 16e Arrondissement	162820
	Paris 17e Arrondissement	166326

FIGURE 12 – Requête Population de Paris par arrondissement.

	Ville	Evolution
▶	Montpellier	137186
	Toulouse	127207
	Saint-Denis	67557
	Cergy	64895
	Saint-Paul	61172
	Nantes	60488
	Évry-Courcouronnes	58814
	Aix-en-Provence	57556
	Annecy	53500
	Le Tampon	49400

FIGURE 13 – Requête sur Liste des 10 villes ayant cru le plus de 1968 à 2020.

	Departement	Evolution
▶	Seine-et-Marne	824296
	Haute-Garonne	725045
	Essonne	632793
	Gironde	627001
	Hérault	597576
	Yvelines	595341
	Loire-Atlantique	583089
	Bouches-du-Rhône	577799
	Val-d'Oise	558535
	Rhône	557054

FIGURE 14 – Requête sur Liste des 10 départements ayant cru le plus de 1968 à 2020.

	Region	Evolution
▶	Île-de-France	3023163
	Auvergne-Rhône-Alpes	2343654
	Occitanie	2081625
	Provence-Alpes-Côte d'Azur	1799830
	Nouvelle-Aquitaine	1356957
	Pays de la Loire	1249254
	Bretagne	905608
	Hauts-de-France	603356
	Grand Est	596478
	Centre-Val de Loire	584625

FIGURE 15 – Requête sur Liste des 10 régions ayant cru le plus de 1968 à 2020.

Region	Evolution
Île-de-France	3023163
Auvergne-Rhône-Alpes	2343654
Occitanie	2081625
Provence-Alpes-Côte d'Azur	1799830
Nouvelle-Aquitaine	1356957
Pays de la Loire	1249254
Bretagne	905608
Hauts-de-France	603356
Grand Est	596478
Centre-Val de Loire	584625

FIGURE 16 – Requête sur Liste des 10 régions ayant cru le plus de 1968 à 2020.

Ville	Naissances
Paris	167944
Marseille	76424
Lyon	45835
Toulouse	40456
Saint-Denis	27848
Nice	27827
Nantes	25172
Strasbourg	23628
Montpellier	22893
Lille	19858

FIGURE 17 – Requête sur Liste des 10 villes où on nait le plus.

Ville	Deces
Paris	83459
Marseille	44962
Nice	22406
Lyon	20690
Toulouse	16654
Nantes	12656
Strasbourg	12204
Le Havre	10892
Toulon	10884
Bordeaux	10842

FIGURE 18 – Requête sur Liste des 10 villes où on meurt le plus.

Departement	Naissances
Nord	198898
Seine-Saint-Denis	174004
Paris	167944
Rhône	155723
Bouches-du-Rhône	155131
Hauts-de-Seine	140722
Val-de-Marne	124731
Val-d'Oise	117962
Seine-et-Marne	113505
Yvelines	113286

FIGURE 19 – Requête sur Liste des 10 départements où on nait le plus.

Departement	Deces
Nord	134855
Bouches-du-Rhône	107221
Pas-de-Calais	87833
Paris	83459
Gironde	79410
Rhône	77003
Seine-Maritime	71840
Alpes-Maritimes	70385
Var	69792
Loire-Atlantique	67855

FIGURE 20 – Requête sur Liste des 10 départements où on meurt le plus.

Departement	Naissances	Deces	Mouvements
Ain	42406	29269	305457
Aisne	35413	34190	2198
Allier	17548	26243	-42210
Alpes-de-Haute-Provence	8749	11348	63237
Hautes-Alpes	7699	8460	49576
Alpes-Maritimes	71293	70385	374432
Ardèche	18246	22120	76272
Ardennes	16063	17321	-38421
Ariège	7754	11152	18874
Aube	19562	18977	40525
Aude	20178	25859	102575
Aveyron	14044	20784	4726

FIGURE 21 – Requête sur Comparaison pour 2020 des naissances / décès / mouvements de population par département.

Region	Naissances	Deces	Mouvements
Auvergne-Rhône-Alpes	552884	405062	2195832
Hauts-de-France	432571	330807	501592
Provence-Alpes-Côte d'Azur	346358	299742	1753214
Grand Est	347891	312859	561446
Occitanie	358420	347138	2070343
Normandie	212284	199473	555191
Nouvelle-Aquitaine	332943	381143	1405157
Centre-Val de Loire	162334	158090	580381
Bourgogne-Franche-Comté	168227	177155	315244
Bretagne	196480	205757	914885
Pays de la Loire	246038	203619	1206835
Île-de-France	1064595	446587	2405155
Corse	17389	18765	139809

FIGURE 22 – Requête sur Comparaison pour 2020 des naissances / décès / mouvements de population par région.

Naissances	Deces	Mouvements_en_million
4905823	3707670	14.6967

FIGURE 23 – Requête sur Comparaison par recensement des naissances / décès / mouvements de population de la France.

```
298     -- Requête 1 :
299 •   SELECT
300         r.nomRegion AS Region,
301         SUM(p.P20_POP) AS Population2020,
302         SUM(p.D68_POP) AS Population1968,
303         ((SUM(p.P20_POP) - SUM(p.D68_POP)) / SUM(p.D68_POP) * 100) AS CroissancePourcentage
304     FROM Region r
305     JOIN Departement d ON r.CodeRegion = d.CodeRegion
306     JOIN Population p ON d.CodeDepart = LEFT(p.CODGEO, 2)
307     GROUP BY r.nomRegion
308     ORDER BY CroissancePourcentage DESC;
309
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Region	Population2020	Population1968	CroissancePourcentage
Corse	343701	205268	67.4401
Occitanie	6494145	4349857	49.2956
Pays de la Loire	3902979	2630670	48.3644
Provence-Alpes-Côte d'Azur	4815492	3467410	38.8786
Auvergne-Rhône-Alpes	7471578	5457349	36.9086
Bretagne	3420105	2509470	36.2879
Centre-Val de Loire	2673950	2054724	30.1367
Nouvelle-Aquitaine	6270899	4904087	27.8709
Normandie	3794639	3035238	25.0195
Île-de-France	14548986	11955984	21.6879
Hauts-de-France	6382544	5532281	15.3691
Grand Est	5517746	4796331	15.0410
Bourgogne-Franche-Comté	3119589	2772151	12.5332

FIGURE 24 – Requête sur La croissance en pourcentage de la population entre 1968 à 2020 par région.

```sql
-- Requête 2 :
SELECT
    r.nomRegion AS Region,
    SUM(p.NAIS1420 + p.NAIS0914 + p.NAIS9909 + p.NAIS9099 + p.NAIS8290 + p.NAIS7582 + p.NAIS6875) AS TotalNaissances,
    SUM(p.DECE1420 + p.DECE0914 + p.DECE9909 + p.DECE9099 + p.DECE8290 + p.DECE7582 + p.DECE6875) AS TotalDeces,
    (SUM(p.NAIS1420 + p.NAIS0914 + p.NAIS9909 + p.NAIS9099 + p.NAIS8290 + p.NAIS7582 + p.NAIS6875) -
     SUM(p.DECE1420 + p.DECE0914 + p.DECE9909 + p.DECE9099 + p.DECE8290 + p.DECE7582 + p.DECE6875)) AS SoldeNaturel
FROM Region r
JOIN Departement d ON r.CodeRegion = d.CodeRegion
JOIN PopulationSerieHistorique2020 p ON d.CodeDepart = LEFT(p.CODGEO, 2)
GROUP BY r.nomRegion
ORDER BY SoldeNaturel DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Region	TotalNaissances	TotalDeces	SoldeNaturel
Île-de-France	10405705	5175403	5230302
Hauts-de-France	4573002	3117725	1455277
Auvergne-Rhône-Alpes	4134748	2718752	1415996
Grand Est	3613699	2526160	1087539
Pays de la Loire	2294802	1533166	761636
Normandie	2472460	1734651	737809
Provence-Alpes-Côte d'Azur	2210847	1686335	524512
Bourgogne-Franche-Comté	1967526	1642696	324830
Bretagne	1927767	1617198	310569
Centre-Val de Loire	1601202	1306821	294381
Occitanie	3151547	2873826	277721
Nouvelle-Aquitaine	3214687	3157801	56886
Corse	146807	143069	3738

737809

FIGURE 25 – Requête sur La différence de naissance et de décès entre 1968 à 2020 par région.

```sql
-- Requête 3 : Évolution des mouvements de population (1968-2020) par département
SELECT
    d.NomDepart AS Departement,
    SUM(p.D68_POP) AS Population1968,
    SUM(p.P20_POP) AS Population2020,
    SUM(p.NAIS1420 + p.NAIS0914 + p.NAIS9909 + p.NAIS9099 + p.NAIS8290 + p.NAIS7582 + p.NAIS6875) AS TotalNaissances,
    SUM(p.DECE1420 + p.DECE0914 + p.DECE9909 + p.DECE9099 + p.DECE8290 + p.DECE7582 + p.DECE6875) AS TotalDeces,
    (SUM(p.NAIS1420 + p.NAIS0914 + p.NAIS9909 + p.NAIS9099 + p.NAIS8290 + p.NAIS7582 + p.NAIS6875) -
     SUM(p.DECE1420 + p.DECE0914 + p.DECE9909 + p.DECE9099 + p.DECE8290 + p.DECE7582 + p.DECE6875)) AS SoldeNaturel,
    (SUM(p.P20_POP) - SUM(p.D68_POP)) AS VariationPopulation
FROM Departement d
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Departement	Population1968	Population2020	TotalNaissances	TotalDeces	SoldeNaturel	VariationPopulation
Seine-et-Marne	604340	1428636	827068	407800	419268	824296
Haute-Garonne	854041	1543492	743359	514973	228386	689451
Essonne	722027	1356371	885644	385246	500398	634344
Bouches-du-Rhône	2418831	3052299	1289021	909968	379053	633468
Gironde	1066523	1688120	839818	663130	176688	621597
Hérault	591397	1188973	538359	431578	106781	597576
Yvelines	854382	1449723	1002254	417802	584452	595341
Loire-Atlantique	862082	1445171	809929	500363	309566	583089
Val-d'Oise	693269	1251804	862473	344119	518354	558535
Rhône	1854183	2405665	1225742	636417	589325	551482
Oise	1167100	1708321	1009269	795972	213297	541221
Var	626859	1142087	551713	505255	46458	515228
Isère	767678	1277513	745905	420624	325281	509835
Haute-Savoie	378550	835206	428556	227657	200899	456656
Ille-et-Vilaine	652722	1088855	607376	376139	231237	436133
Seine-Saint-Denis	1264388	1675947	1299361	497251	802110	411559

FIGURE 26 – Requête sur La variation de population par région entre 1968 et 2020.

```sql
-- Requête 4 : Taux de natalité et mortalité par région en 2020
SELECT
    r.nomRegion AS Region,
    SUM(p.NAIS1420) AS TotalNaissances2020,
    SUM(p.DECE1420) AS TotalDeces2020,
    SUM(p.P20_POP) AS PopulationTotale2020,
    (SUM(p.NAIS1420) / SUM(p.P20_POP)) * 100 AS TauxNatalite,
    (SUM(p.DECE1420) / SUM(p.P20_POP)) * 100 AS TauxMortalite
FROM Region r
JOIN Departement d ON r.CodeRegion = d.CodeRegion
JOIN PopulationSerieHistorique2020 p ON d.CodeDepart = LEFT(p.CODGEO, 2)
GROUP BY r.nomRegion
ORDER BY TauxNatalite DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Region	TotalNaissances2020	TotalDeces2020	PopulationTotale2020	TauxNatalite	TauxMortalite
Île-de-France	1239304	539654	14548986	8.5181	3.7092
Provence-Alpes-Côte d'Azur	349869	267896	4815492	7.2655	5.5632
Hauts-de-France	458504	356190	6382544	7.1837	5.5807
Auvergne-Rhône-Alpes	532495	361759	7471578	7.1269	4.8418
Pays de la Loire	249664	207859	3902979	6.3968	5.3256
Normandie	239742	226455	3794639	6.3179	5.9678
Centre-Val de Loire	168220	164432	2673950	6.2911	6.1494
Grand Est	346718	307253	5517746	6.2837	5.5685
Occitanie	389907	376760	6494145	6.0040	5.8015
Bourgogne-Franche-Comté	186878	198508	3119589	5.9905	6.3633
Bretagne	199379	208631	3420105	5.8296	6.1001
Nouvelle-Aquitaine	347168	397575	6270899	5.5362	6.3400
Corse	17389	18765	343701	5.0593	5.4597

FIGURE 27 – Requête sur Le taux de natalité et mortalité par région en 2020.

```
352    -- Requete 5
353 •  SELECT
354        d.NomDepart AS Departement,
355        SUM(p.D68_POP) AS Population1968,
356        SUM(p.P20_POP) AS Population2020,
357        (SUM(p.P20_POP) - SUM(p.D68_POP)) AS PertePopulation
358    FROM Departement d
359    JOIN Population p ON d.CodeDepart = LEFT(p.CODGEO, 2)
360    GROUP BY d.NomDepart
361    HAVING PertePopulation < 0
362    ORDER BY PertePopulation ASC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Departement	Population1968	Population2020	PertePopulation
Paris	5181542	4291812	-889730
Nièvre	247702	202670	-45032
Haute-Marne	214340	171798	-42542
Creuse	224461	184016	-40445
Indre	247178	218707	-28471
Meuse	209372	183001	-26371
Vosges	388201	362397	-25804
Cantal	169330	144379	-24951
Cher	304601	300933	-3668
Lozère	77258	76633	-625

FIGURE 28 – Requête sur La perte de population des villes entre 1968 et 2020.

4 Conclusion

Le projet de base de données sur la population nous a permit de travailler un peu plus profondément les requêtes MySQL avec une grosse base de données. Le projet nous a permit de travailler sur MySQL avec des données bien concrètes et en sortir des résultats satisfaisants. C'était un défi aussi car le passage du csv en table SQL n'était pas simple à réaliser. Néanmoins le projet était complexe nous avons eu souvent des complications pour des problèmes de colonnes. Les questions du projet sont un cran au-dessus de la difficulté des requêtes Pokemon travailler en cours. Malgré la difficulté nous avons réussi avancer et répondre aux attentes demandées. Ce projet est une bonne application des requêtes vues et un bon moyen d'approfondir la connaissance sur l'utilisation des requêtes malgré certaines difficultés qui n'étaient pas toujours évidentes à régler.

5 Annexe graphique

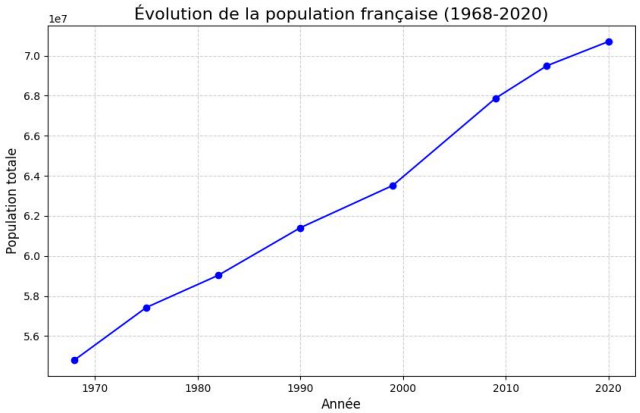

FIGURE 29

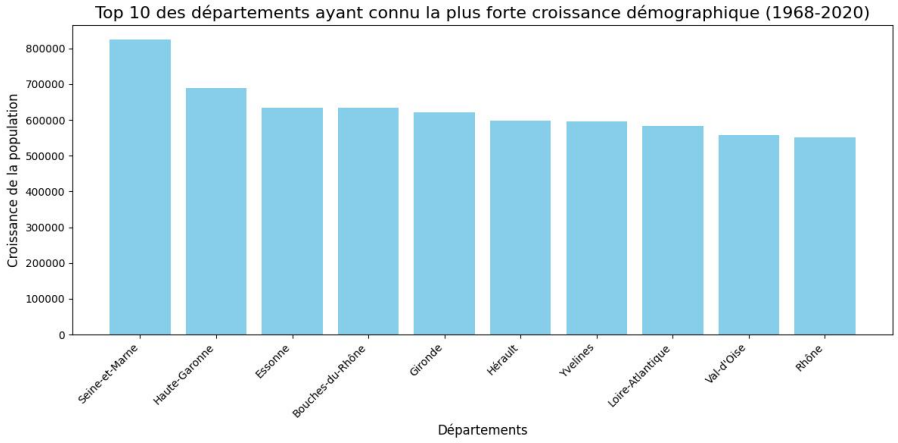

FIGURE 30

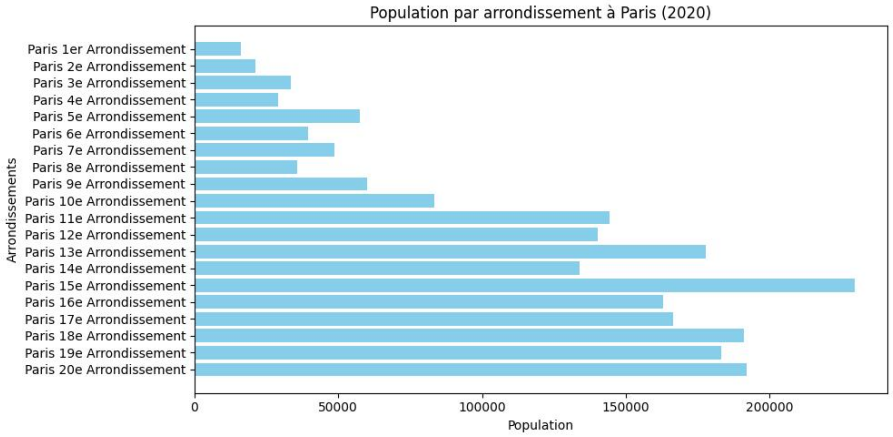

FIGURE 31

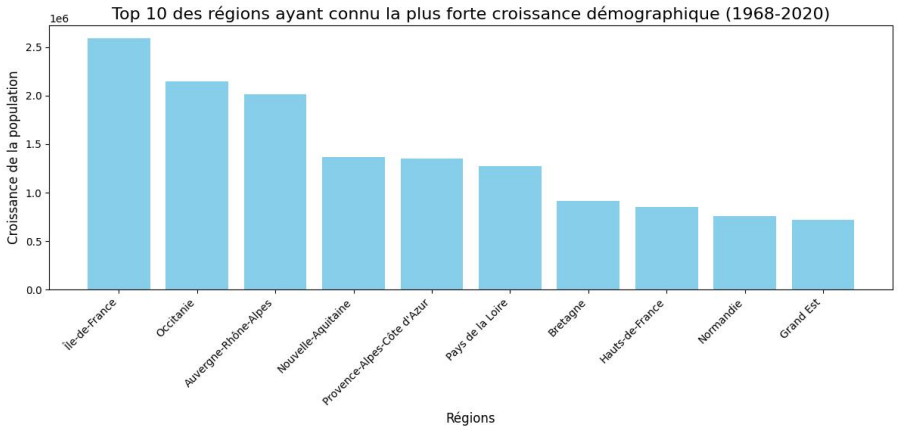

FIGURE 32

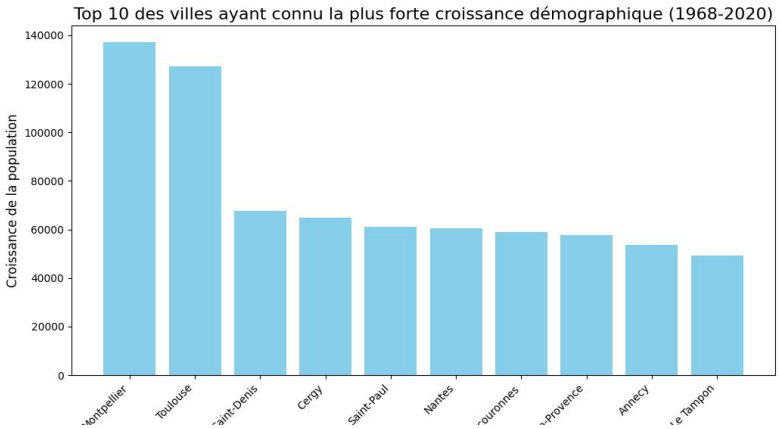

FIGURE 33

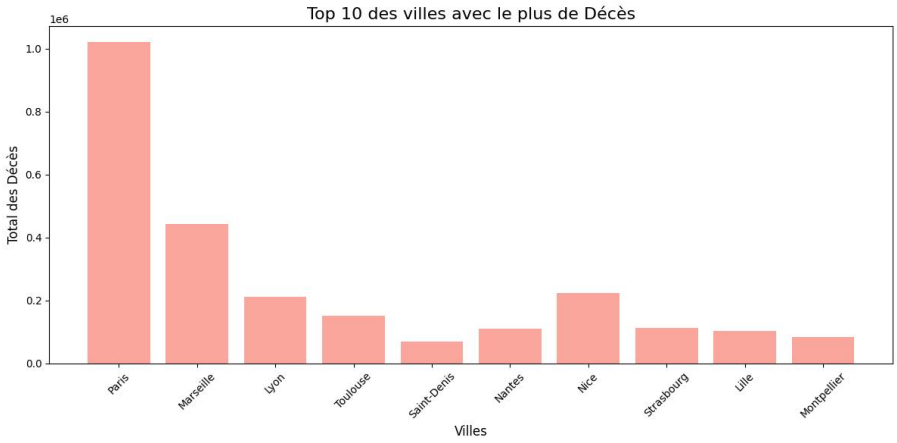

FIGURE 34

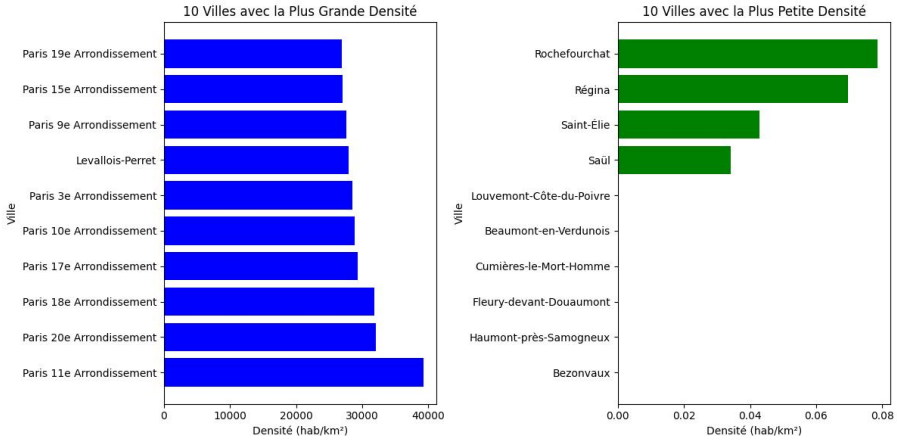

FIGURE 35

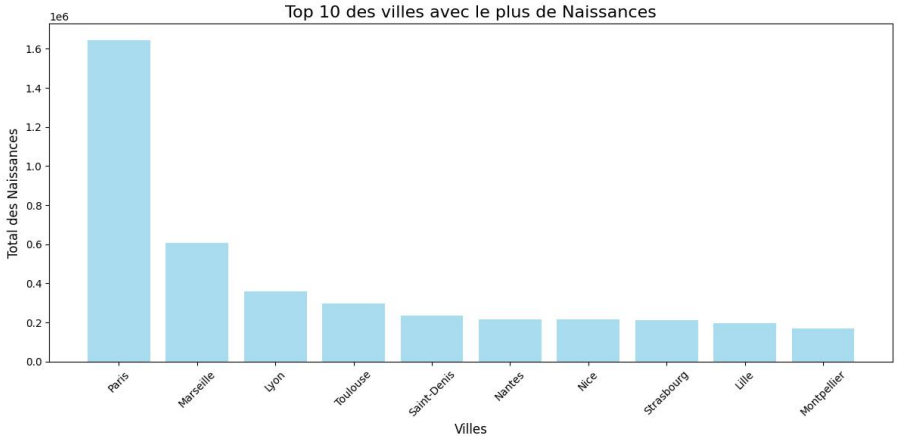

FIGURE 36