CORNIER Alexandre
07/05/2020

université de BORDEAUX

Projet INSEE

Introduction :

Dans le cadre de notre projet en Bases de Données, nous nous intéressons aux données sociales et environnementales pour les départements Français de 2008 à 2016. L'INSEE (Institut National Statistiques Études Économiques) publie régulièrement de nombreuses informations et est en charge du recensement de la population Française.

A l'aide du langage de programmation Python nous allons récupérer les différentes données qui ont été exportées dans des fichier CSV (Comma-separated values) et dans des fichiers Excel. Puis par la suite, nous allons les implémenter dans un système de gestion de base de données relationnelles et objet (SGBDRO). Une fois ce premier script réalisé, nous en développerons un second afin d'interagir avec cette Base de Données en effectuant des interrogations à partir d'un menu dans la console, avec une présentation facilitant la lecture.

Le but du projet est donc d'apprendre à stocker des informations dans une Base de Données par le biais de scripts Python, et par la suite de pouvoir les manipuler grâce à des requêtes SQL afin de répondre aux questions posées pour ce projet.

2020

CORNIER Alexandre
07/05/2020

Plan :

Matériel et logiciels utilisés

Pour ce projet, nous avons simplement besoin d'un ordinateur sur lequel est installé un système de gestion de Base de Données et un langage de programmation. Concernant la partie Base de Données, nous avons choisi PostgreSQL pour différentes raisons :

- Il fonctionne sur les principaux systèmes d'exploitation (Linux, MacOS et Windows),
- Étant open-source, il dispose d'une grande communauté mondiale,
- Il dispose également d'une très bonne compatibilité avec SQL car il suit les normes SQL 92 et 99,
- Enfin il existe de nombreux tutoriels pour l'installation et l'utilisation.

Concernant l'installation, il faut se rendre sur le site https://www.postgresql.org. Une fois arrivé sur la page d'accueil on clique sur l'onglet « Donwload », ensuite il suffit de choisir le système d'exploitation de notre ordinateur, dans notre cas il s'agit de Mac OSX.

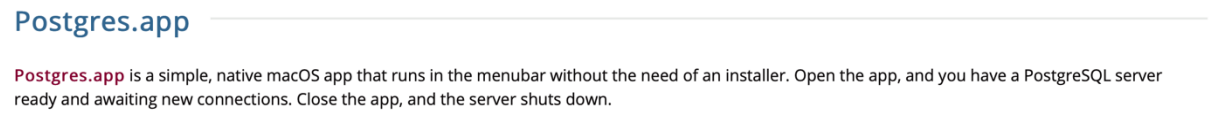

Figure 1 : Téléchargement de l'application postgres

Maintenant nous avons besoin d'un langage de programmation afin de construire et d'interagir avec la Base de Données. Python est un langage de programmation puissant et facile à apprendre. Il possède des structures de données de haut niveau et permet une approche simple mais efficace de la programmation orientée objet. Parce que sa syntaxe est élégante, son typage dynamique et il est interprété, Python est un langage idéal pour l'écriture de scripts et de développement rapide d'applications dans de nombreux domaines sur la plupart des plateformes.

Python est un langage interprété : comme PHP, un programme Python ne nécessite pas d'étape de compilation en langage machine pour fonctionner. Le code est interprété au moment de l'exécution. Python présente néanmoins une caractéristique intéressante : comme en Java, le code Python est compilé en octet-code (format intermédiaire) avant son lancement, ce qui optimise ses performances. Ce langage de programmation a un typage dynamique fort : cela signifie que le typage, bien que non vérifié lors de la "compilation", Python effectue des vérifications de cohérence sur les types utilisés, et permet de transformer explicitement une variable d'un type à l'autre. Dans le cadre de notre projet, python est un

langage de programmation idéal car il dispose d'une très grande communauté et donc d'une grande variété de bibliothèques disponibles. Pour ce projet, nous avons utilisé les bibliothèques suivantes :

- Psycopg2 est l'adaptateur de Bases de Données PostgreSQL le plus populaire pour le langage de programmation Python,
- Le module CSV implémente des classes pour lire et écrire des données tabulaires au format CSV,
- Pandas permet la manipulation, l'analyse de données, et dans une certaine mesure, leur présentation,
- Numpy permet la manipulation des matrices ou des tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux,
- Jinja2 qui est une dépendance de pandas pour la prise en charge des styles HTML.

Création de la base de données

Le premier code, read.py, contient les requêtes SQL permettant de créer notre Base de Données.

1. Tables sources de la base de données

La méthode de conception d'une Base de Données débute en principe par la création d'un modèle de données théorique, qui permet ensuite de générer un script pour créer la base de données, puis charger les données dont on dispose.
Dans notre cas, nous disposons d'une source de données sous la forme de 3 fichiers qui fournissent des tables et les données, d'où le premier script de création de la base et de chargement des données.

Les 3 fichiers sont les suivants :

- region2020.csv qui contient la liste des régions avec leurs codes et leurs noms,
- departement2020.csv qui contient la liste des départements avec leurs codes et leurs noms,
- DD-indic-reg-dep_janv2018.xls qui fournit plusieurs tables relatives aux informations environnementales et sociales pour les régions et les départements.

2. Requêtes SQL utilisées

Nous avons choisi de créer sur le serveur une Base de Données spécifique, que nous nommons cremi. La création de la Base repose sur 3 types de requêtes SQL :

- CREATE DATABASE pour construire la Base de Données "vide",
- CREATE TABLE pour créer la structure des tables, avec la définition de leurs colonnes,
- INSERT INTO pour charger les données dans les tables.

Pour chaque table il faut préciser les colonnes et leurs types, ainsi que la clé principale.

Afin de faciliter les relations entre les tables, nous avons au préalable analysé les colonnes pour déterminer lesquelles seront utilisées pour les jointures, et pour harmoniser leurs types. Par exemple, pour créer la table départements, voici la requête SQL exécutée :

```
CREATE TABLE departements(
       dep text PRIMARY KEY,
       reg text,
       cheflieu text,
       tncc text,
       ncc text,
       nccenr text,
       libelle text);
```

La colonne reg aurait pu être définie en integer, cependant elle permet la jointure avec la table régions qui contient des valeurs alphanumériques. C'est pourquoi nous l'avons définie en string.

3. Préparation des données

Nous aurions pu modifier manuellement les fichiers source cependant ce n'est pas une bonne pratique. De plus c'est un procédé difficile à passer à l'échelle si nous devions charger des milliers de lignes. Une fois la base construite et les données chargées, nous avons modifié certaines valeurs grâce à des requêtes SQL pour garantir la jointure entre les tables. Il s'agit des colonnes contenant les codes de région et de département dans les tables régions et départements.
Étant donné que le format est string, il faut que la chaîne de caractères soit identique pour une même région entre toutes les tables jointes. Nous avons choisi d'ajouter un zéro aux codes inférieurs à 10 pour harmoniser.

Pour cela nous utilisons des requêtes SQL de type UPDATE SET, par exemple pour mettre à jour la table départements :

```
UPDATE departements
SET reg = CONCAT('0', reg)
WHERE dep LIKE '_';
```

la fonction CONCAT permet d'ajouter un zéro en début de chaîne de caractères et l'opérateur LIKE suivi de l'expression régulière '_' permet de n'agir que sur les valeurs ne contenant qu'un seul caractère, donc inférieur à 10 dans notre contexte.

4. Code python et détails techniques

La structure générale du code python est la suivante :

- Création de la base de données,
- Création des tables, avec pour chacune d'elles :
    - Création de la table,
    - Chargement des données depuis le fichier correspondant
    - Enregistrement des données dans la table,
- Préparation des données.

Concernant la partie code python permettant de se connecter à la base, chaque requête utilise un curseur pointant sur une connexion à la base. Dans notre contexte postgres la connexion est créée avec la chaîne suivante envoyée au module python psycopg2 :

```
psycopg2.connect("host=localhost port=5432 user=postgres")
```

Nous utilisons 2 connections successives, la première sur le serveur pour créer la base, puis celle-ci est refermée et réouverte, en pointant cette fois sur la base nouvellement créée.

Pour la partie chargement des CSV, nous alimentons les tables à partir des fichiers CSV, en utilisant :

```
file = open("departement2020.csv", encoding="utf8")
reader = csv.reader(file, delimiter=",")
```

L'encodage utf8 forcé permet la gestion correcte des accents (à ajuster selon la plateforme) et le délimiteur ',' permet d'identifier les colonnes dans les lignes de texte.
Pour la partie chargement des tableaux Excel, nous utilisons le module Pandas :

```
myFile = 'DD-indic-reg-dep_janv2018.xls'
df1 = pd.read_excel(myFile, sheet_name='Social',
usecols="A:F",skiprows=3,nrows=21,header=None)
df1.replace(["nd", "nd ", "nc", "nc"], np.NaN, inplace = True)
```

5. Schéma de la base de données obtenue

Le schéma ci-dessous montre la base de données résultante, avec la représentation des différentes relations que nous utilisons dans les interrogations.

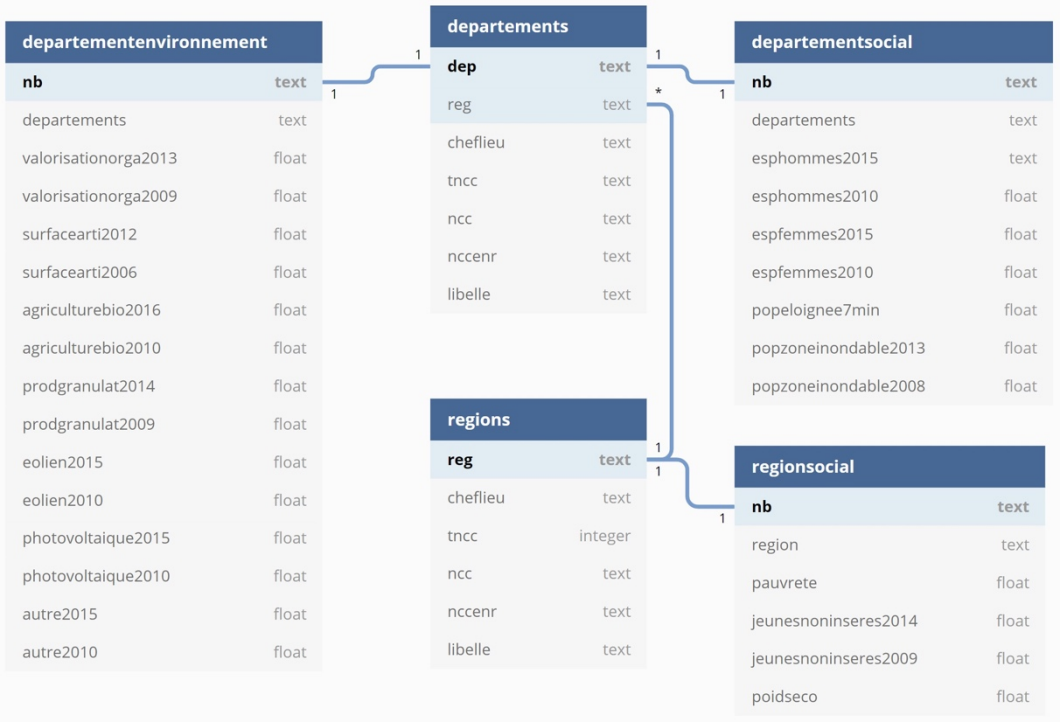

Figure 2: schéma de la base de données du projet INSEE

Interrogation de la Base de Données

Le second code, request.py, contient les requêtes SQL permettant de répondre aux questions posées pour l'exercice.

1. Requêtes SQL utilisées

Dans un premier exemple nous allons nous intéresser à la requête n°2. Cette requête permet d'obtenir la liste des départements. Pour cela il suffit d'interroger la table départements pour lister les valeurs de la colonne libelle. C'est ce que produit le SELECT libelle FROM departement.

Il est d'usage lorsque l'on présente une liste de départements d'associer le numéro du département et de présenter la liste dans l'ordre. Pour cela nous avons ajouté la colonne dep et la commande ORDER BY.

```
SELECT dep, libelle FROM departements ORDER BY dep;
```

Le résultat de la requête avec une présentation HTML est enregistré dans le fichier Question_02.html.

01	Ain
02	Aisne
03	Allier
04	Alpes-de-Haute-Provence
05	Hautes-Alpes
06	Alpes-Maritimes

Figure 3 : Résultat de la requête n°2 au format HTML

Dans un second exemple, nous allons voir la requête n° 6 avec jointure et somme. Cette requête plus complexe montre comment résoudre des contraintes nécessitant des calculs, et sur différentes tables jointes. La question posée est : "Quels sont les départements dont la région a eu une production de granulats supérieure à 25 000 000 tonnes en 2014".

Pour cela nous utilisons la table departement (pour le libelle du département) qui est liée à la fois à la table région (pour le libelle de la région) et la table departementenvironnement qui contient la colonne prodgranulat2014. L'objectif est d'obtenir la liste des régions pour lesquelles la somme totale de la production de tous ses départements est supérieure à 25 000 000 tonnes.

C'est le résultat fourni par le second select, entre parenthèses, ci-dessous :

```
SELECT departements.reg, regions.libelle
AS region, departements.libelle
AS departement
        FROM departements, regions
        WHERE departements.reg
        IN (SELECT departements.reg from departements
            INNER JOIN departementenvironnement
            ON departements.dep = departementenvironnement.nb
            INNER JOIN regions
            ON departements.reg = regions.reg
            GROUP BY departements.reg
            HAVING SUM(prodgranulat2014) > 25000000
            AND SUM(prodgranulat2014) <> 'NaN')
        ORDER BY region, departement
```

Departement et departementenvironnement sont jointes par les colonnes dep et nb, c'est le code du département dans chaque table. Departement et regions sont jointes par les colonnes reg et reg, c'est le code de la région dans chaque table.

- Le GROUP BY permet de faire la somme des départements pour chaque région,
- Le HAVING SUM réalise la somme et vérifie que la production est supérieure à 25 000 000 tonnes.

Le premier résultat obtenu est la liste des régions qui satisfont aux exigences. La question porte sur les départements, il faut donc ensuite lister tous les départements appartenant aux régions trouvées précédemment. C'est ce que réalise le premier SELECT.

Code région	Département
44	Ardennes
44	Aube
44	Bas-Rhin
44	Haut-Rhin
44	Haute-Marne
44	Marne

Figure 4 : Résultat de la requête n°6 au format HTML

Comme pour l'exemple 1, dans un souci qualitatif de présentation, nous sélectionnons le code de région, son libellé, et le libelle du département, en accentuant ce dernier dans le visuel HTML. Notons que pour les colonnes reg et libelle il est nécessaire de préciser la table puisque ces colonnes existent avec le même nom dans plusieurs tables.

Les jointures précédentes s'appliquent et permettent de choisir des colonnes issues de plusieurs tables liées, ici departements et regions. La commande WHERE departements.reg IN permet de sélectionner les valeurs uniquement lorsque la région considérée appartient à la liste obtenue précédemment. Pour terminer, le ORDER BY présente les données dans l'ordre croissant des codes de régions.

2. Méthode de développement employée

Le code contient trois langages distincts :

- SQL pour les requêtes dans la Base de Données,
- Python pour le code principal, les appels de fonctions,
- HTML pour les éléments de présentation.

Nous nous sommes d'abord concentrés sur les requêtes SQL. Pour cela nous avons utilisé pgAdmin4 qui permet de visualiser et gérer la base mais aussi de développer et tester les requêtes SQL grâce à l'outil "Query Tool". Nous aurions disposé de l'équivalent avec un autre SGBDR, par exemple WorkBench pour MySQL ou SQL Studio pour MSSQL. Notons qu'il existe de nombreux outils, par exemple l'Open Source (ASL) dbeaver. Les codes obtenus ont ensuite été intégrés dans le code python.

La deuxième partie a consisté à bâtir le programme, avec son menu et l'exécution des requêtes. Pour terminer, nous avons travaillé sur la qualité de la présentation en intégrant des éléments HTML et des styles.

3. Code et détails techniques

3.1. Structure générale du code

La réalisation de ce programme met en œuvre plusieurs solutions techniques dont voici quelques précisions. Le code contient les fonctions suivantes dans l'ordre :

- Connection à la Base de Données,
- Ensemble des fonctions :
    - Affichage HTML sous forme de fonction car l'apparence générale est identique pour toutes les requêtes,
    - 11 fonctions, une par requête correspondant aux questions,
    - Menu principal,
- L'équivalent du "main" qui gère les demandes interactives de l'utilisateur.

3.2. Pandas DataFrame, styles et HTML

Il existe plusieurs solutions pour charger le résultat d'une requête et le représenter. La solution basique est d'utiliser une boucle For sur le curseur pointant sur la requête, puis de manipuler la présentation en mode string. Cette méthode a toutefois des limites et la qualité de représentation des tableaux est à la fois difficile à obtenir et très spécifique à la requête.

Nous avons choisi de passer par un DataFrame offert par le module Pandas qui propose des fonctions avancées pour la gestion des styles et pour l'export dans un fichier HTML. Notons que certains styles tel que le format des float est utilisable à la fois pour la version console et pour la version HTML.

```
df = pd.DataFrame(query_result, columns=['Code région', 'Région', 'Département'])

    html = (df.style
        .set_table_styles([
            {'selector': 'tr:nth-of-type(odd)', 'props': [('background', '#eee')]},
            {'selector': 'tr:nth-of-type(even)', 'props': [('background',
'white')]},
            {'selector': 'th', 'props': [
                ('background', '#606060'),
                ('color', 'white'),
                ('font-family', 'verdana')]},
            {'selector': 'td', 'props': [('font-family', 'verdana')]}])
        .apply(lambda x: ['background: lightblue' if x.name == "Département" else
'' for i in x])
        .hide_index()
        .render())
```

Le DataFrame est construit à partir du résultat de la requête SQL exécutée par le curseur sur la connexion. Les noms des colonnes peuvent être remplacés facilement lors de la création (c'est le columns=xxx). On assigne ensuite à une variable string (html) le corps HTML de la présentation qui sera intégrée dans le document HTML par fonction d'affichage.

- .set_table_styles permet de modifier les éléments de la table : couleurs de fond, police de caractère,
- .apply d'une fonction lambda permet d'accentuer en bleu clair dans ce cas, la colonne qui correspond à la réponse attendue,
- .hide_index est la méthode qui permet de cacher à l'affichage les index par défaut du DataFrame, qui n'appartiennent pas à la table dans la Base,
- .render est la méthode finale qui construit le code HTML répondant aux éléments stylistiques précédents.

3.3. Autres éléments de présentation

Les fonctionnalités de style de Pandas DataFrame permettent d'améliorer la qualité, et donc la lecture des données. Nous avons utilisé par exemple les effets suivants.

Concernant le cadrage à droite pour certains nombres :
`.set_properties(subset=["Valeur"], **{'text-align': 'right'})`
Placé avant le .render, cela permet de préciser l'alignement souhaité pour une ou plusieurs colonnes, dans ce cas un alignement à droite pour la colonne "Valeur".

Pour le format d'un float :

- .format({"Valeur": "{:.1f}"}) : permet de présenter sous la forme x.x un nombre de type float,
- .format({"Croissance": "{:.1f}pts"}) : idem avec dans ce cas l'ajout de l'unité, ici pts, pour points précise qu'il s'agit d'un écart entre deux pourcentages,
- .format({"Poids de l'économie sociale": "{:.1f}%"}), même principe, cette fois pour un pourcentage. Attention, le % est après le }. S'il était placé avant, une conversion aurait eu lieu et il aurait fallu diviser par 100 la valeur pour obtenir le pourcentage souhaité.

Pour la partie conversion en numérique des valeurs d'une colonne :
Dans certaines colonnes, il existe des valeurs alphanumériques. Cela se traduit par un type object qui ne permet pas à Pandas d'exploiter certaines fonctionnalités tel que le format ou le gradient qui ne prennent en charge que des nombres. Pour y remédier, il est possible, toujours avec Pandas, de convertir les valeurs en type numérique (float dans notre cas) puis de réaliser les changements de style.

Exemple :

```
df["Part moyenne du photovoltaïque"] = pd.to_numeric(
    df["Part moyenne du photovoltaïque"], errors='coerce').fillna(0)
```

Converti cette colonne en float (par défaut), et remplace les "Nan" en 0 lorsque la valeur initiale n'était pas un nombre.

Pour améliorer encore la présentation et le lecture, nous avons ajouté un style proposé par Pandas, le .background_gradient, qui définit une couleur de fond pour chaque cellule d'après sa valeur, et en fonction des min et max de la liste des données. Notons que cette méthode doit être placée dans la liste des styles avant le formatage de la colonne car ce dernier transforme le type en string du fait par exemple de l'ajout d'un symbole pts ou % :

```
.background_gradient(cmap='Blues', subset=["Poids de l'économie sociale"])
.format({"Poids de l'économie sociale": "{:.1f}%"})
```

Dans cet exemple, la colonne "Poids de l'économie sociale" est affichée en nuances de bleus, et seulement ensuite, les valeurs sont formatées en x.x%.

Enfin pour l'affichage HTML et console en option, par défaut nous utilisons une présentation HTML qui apporte une meilleure qualité et plus de styles possibles. Cependant, nous avons conservé la possibilité d'afficher les résultats à la console, en utilisant un simple print(df), df étant le DataFrame.

Conclusion

Pour conclure, ce projet nous a permis d'apprendre à manipuler les différents outils liés aux Bases de Données. Notamment, le système de gestion de bases de données relationnelle et objet PostgreSQL.
Savoir interagir avec cette dernière en réalisant des scripts Python pour pouvoir répondre aux différentes questions proposer par notre enseignant. Ensuite pour le format de présentation des résultats nous avons choisi un affichage console qui est simple mais efficace. Mais pour aller plus loin dans la démarche et proposer une visualisation plus agréable et plus simple à lire, nous avons choisi d'utiliser un affichage HTML permettant mettre en valeurs les données.

Références

- https://www.postgresql.org
- https://www.pgadmin.org
- https://stackoverflow.com
- https://www.geeksforgeeks.org
- https://www.python.org
- https://dbeaver.io