Institut Universitaire de Technologie,
Aix-Marseille Université

RAPPORT DE STAGE
Diplôme Universitaire de Technologie
Spécialité Réseaux et Télécommunications

Automatisation en Python des mises à jour d'une base de données sous PostgreSQL

Léo VIGUIER

Direction Départementale des Territoires de l'Aveyron

Responsable entreprise : Aurélie Bonnefis

Responsable académique : Éric Wurbel

2021

Table des matières

1 Introduction

Pour conclure mon DUT* en Réseaux et Télécommunications, 10 semaines de stage en entreprise en rapport avec la formation viennent conclure cette scolarité. Étant donné que j'apprécie la programmation, ce stage était une réelle opportunité pour en apprendre plus en Python* et m'améliorer dans ce domaine de l'informatique que j'apprécie tant.

A la Direction Départementale des Territoires de l'Aveyron (DDT12) à Rodez, je me situais au sein du Service Aménagement du Territoire, Urbanisme et Logement (SATUL), et plus précisément dans l'unité de la Mission Aménagement, Analyse et Connaissance du Territoire (MAACT).

Ma mission principale était de réaliser un programme automatisant les mises à jour d'une base de données à partir de fichiers classeurs. Cette mission m'a fait travailler sur Python et PostgreSQL*, 2 compétences vues au cours de ma scolarité au sein de l'IUT.

J'aborderais dans ce rapport mes étapes dans la réalisation de la mission principale ainsi que les missions secondaires abordées à la fin de mon projet principal.

2 Présentation de l'Administration

2.1 Historique

Depuis le 1er janvier 2010, dans le cadre de la révision générale des politiques publiques, les DDT ont remplacé diverses directions préexistantes, notamment :

- Les Directions Départementales de l'Equipement (DDE) ;
- Les Directions Départementales de l'Agriculture et de la Forêt (DDAF) ;
- La Direction des Affaires Maritimes (DAM), pour les départements littoraux.

A partir du 1er janvier 2009, dans 47 départements de métropole, les DDE et les DDAF ont fusionné en Directions Départementales de l'Equipement et de l'Agriculture (DDEA), lesquelles se sont alors regroupées avec le service chargé du contrôle de légalité en matière d'environnement des préfectures pour donner naissance aux DDT.
La DDT est donc une DDI (Direction Départementale Interministérielle), un organisme sous l'autorité du préfet.

2.2 Activités au sein de la DDT

Les DDT veillent au développement équilibré et durable des territoires, tant urbains que ruraux, par le biais des politiques agricole, d'urbanisme, de construction, d'aménagement et de transport. Les principales missions des DDT recouvrent les domaines suivants :

Aménagement et urbanisme

Des enjeux importants de préservation des terres agricoles, de limitation de l'étalement urbain et de croissance responsable ; planification ou suivi des grands projets ; instruction des autorisations de construire et d'aménager relevant de la compétence de l'État.

Logement, habitat et construction

La gestion et le contrôle des aides publiques pour la construction de logements sociaux ainsi que la rénovation urbaine des quartiers sensibles, la lutte contre la précarité énergétique.

Agriculture et forêts

Promotion de leurs fonctions économiques, sociales et environnementales.

Préservation de l'environnement

Protection et gestion durable des eaux, des espaces naturels, forestiers et de leurs ressources : police de l'eau et des milieux aquatiques, sécurisation de la ressource en eau, de la biodiversité.
Protection et gestion de la faune et de la flore sauvages, notamment la préservation des habitats et des espèces menacées.
Gestion des déchets et des nuisances.

Prévention des risques

Élaboration et mise en œuvre des plans de prévention des risques naturels ou technologiques.

Éducation et sécurité routière

Déploiement de radars automatiques et réglementation relative aux routes dites à grande circulation ainsi que l'organisation et la tenue des examens du permis de conduire.

2.3 Organisation du SATUL

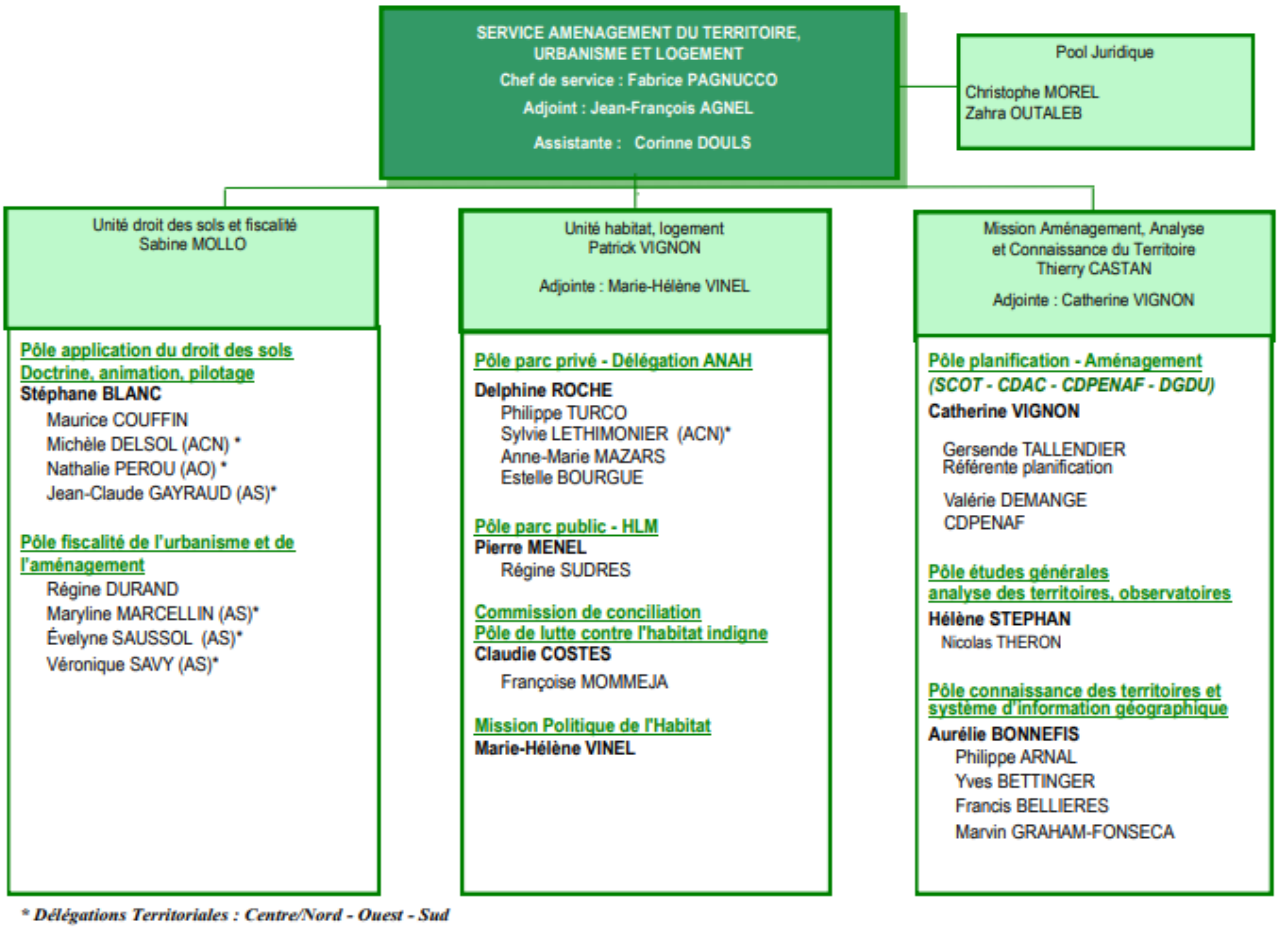

Figure 1 : Organigramme du SATUL

Unité droit des sols et fiscalité

Le pôle application du droit des sols délivre des permis de construire pour lesquels l'État est compétent, c'est-à-dire quand la commune est soumise au règlement national d'urbanisme. Dans le cas contraire, la commune se charge de délivrer les permis de construire car elle possède un plan local d'urbanisme. Ce pôle s'occupe aussi de délivrer des permis dits États : ce sont des chantiers important, tels que des centrales nucléaires, ou des grands bâtiments de l'État (DDT, Préfecture).

Le pôle fiscalité gère les taxes générées par les permis, qu'il convient de calculer et de mettre en recouvrement.

Unité habitat logement

Le pôle parc privé mets en place des aides pour les propriétaires bailleurs et les occupants afin de réhabiliter leurs logements grâce à des aides financières distribuées par l'agence nationale de

l'habitat. L'État concentre ses crédits sur des territoires précis au moyen de programme dits opération programmée d'améliorations de l'habitat. Il s'agit aussi de faire des études d'un secteur pour choisir le meilleur endroit où appliquer ces programmes.

Pôle parc public : L'État aide, suit et contrôle les organismes HLM (Habitations à Loyers Modérés). Il finance aussi la construction et la rénovation de logements locatifs sociaux.

La commission de conciliation propose un lieu d'échange entre les propriétaires et les locataires en cas de conflits. Le Pôle de lutte contre l'habitat indigne se charge de repérer les logements très dégradés et mettre en demeure les propriétaires de réaliser des travaux dans des délais contrains, auquel cas l'État s'en charge, au frais du propriétaire.

La mission politique de l'habitat se charge d'identifier et caractériser les problématiques en matière d'habitat dans l'Aveyron.

MAACT

Pôle planification - aménagement : leur rôle est d'accompagner les collectivités à définir les secteurs constructibles sur leur territoire, mais aussi de préserver les espaces naturels, agricoles et paysagés, et éviter la trop grande expansion des villes.

Le pôle étude est le lieu de réflexion et de rédaction de notes sur les grands enjeux nationaux relatifs notamment à l'implantation des énergies renouvelables, la gestion des friches industrielles et des zones d'activités économique (ZAE).

Le pôle connaissance du territoire est en charge de l'élaboration de cartes pour les services de la DDT, mais aussi les autres services de l'état et s'occupe aussi de la gestion des données publiques disponible pour la réalisation d'études nécessaires au service.

Pôle juridique

Il a pour mission d'analyser les procédures mises en œuvre contre l'état dans le domaine de compétence de la DDT, de répondre aux sollicitations des parquets et de verbaliser les infractions au code de l'urbanisme. Ce pôle est également chargé de contrôler la légalité des documents de planification et de la fiscalité

3 Description de la mission principale

3.1 Contexte

La connaissance des acteurs du territoire et des collectivités fait partie du socle de base du travail des agents de la DDT. Avant la création de la base de données (BDD) base_territoire, chaque service, chaque agent, travaillait avec ses propres outils d'information pour la connaissance de ces acteurs. Ce fonctionnement n'était pas efficient et l'homogénéité des informations n'était pas assurée.
La base Territoire a été créée avec l'objectif de rendre accessible à chacun des agents de la DDT des éléments de connaissances générique du département, administratifs et statistiques.
Cette base facilite le travail des agents dans leur recherche et leur analyse du territoire en fournissant des informations homogènes entre chacun.
Avec cette base, les agents peuvent facilement accéder aux données grâce au logiciel QGIS, permettant de créer des cartes et de se connecter à la base territoire afin d'afficher toutes les informations relatives à une commune par exemple, en un seul clic.
Avec la quantité d'informations que cette base stocke, il était difficilement concevable de la mettre à jour manuellement, surtout après des élections municipales où il y aurait potentiellement 285 noms et prénoms de maires à changer, un travail fastidieux.

Ma mission intervient à la dernière étape de la création de cette base : c'est l'automatisation de sa mise à jour.

3.2 Cahier des charges

Le programme créé pourra être lancé depuis n'importe quelle machine de la DDT, et sera assez simple et intuitif à utiliser. De plus, tout le code sera écrit en Python, étant donné que c'est un langage facile à comprendre et à apprendre.
Dans le cas d'erreurs ou de modifications, les agents peuvent être amenés à devoir modifier le programme donc il est nécessaire de l'écrire de manière compréhensible ainsi que d'expliquer le rôle de chaque fonction et aussi des variables.
Il va aussi de soi qu'une documentation sur l'installation et l'utilisation du programme, de ses potentielles erreurs, de sa configuration et de son fonctionnement doit être rédigée.
Enfin, étant donné que d'autres DDT sont dans la même situation de devoir automatiser la mise à jour de leur base de données, il faut rendre le programme le plus flexible et adaptatif possible afin de fonctionner dans des conditions différentes de celles de développement et de test.

4 Travail réalisé

En n'ayant pas eu une formation de développeur informatique, la question à se poser est : par où commencer ?
Je me suis d'abord renseigné sur la base de données pour voir sa structure et mieux comprendre ce qui devait être automatisé. Un protocole de mise à jour manuelle était aussi accessible, ce qui m'a grandement aidé à trouver un point de commencement.

4.1 Première tentative

4.1.1 Téléchargement des fichiers

Il y a quatre tables à mettre à jour automatiquement et chacune d'elles ont besoin de leurs propres fichiers qu'il faut donc télécharger avant de lancer la mise à jour.
Ce téléchargement peut d'habitude facilement être automatisé, sauf dans le cas où le site utilise du JavaScript* pour accéder au téléchargement. Des solutions existent, telles que l'utilisation d'un navigateur internet géré par un programme : une copie conforme d'un navigateur internet (Firefox ou Chrome notamment) mais avec une extension qui ouvre un socket* sur l'ordinateur et où un programme peut se connecter pour envoyer des instructions. Cette technique à l'avantage de ne pas être embêtée par le JavaScript et peut parfois même accepter les cookies*, qui peuvent être un véritable frein quand certains sites n'autorisent pas la navigation sans ceux-ci.
Cependant, la flexibilité du programme, un des critères de réalisation, n'est pas assurée car il faut indiquer manuellement au programme où cliquer, et ces codes ne sont pas intuitifs à comprendre :

```
/html/body/div[1]/div/div[4]/div[2]/div/ul/li[1]/ul/li[1]/ul/li[1]/a
```

Ceci correspond au chemin (XPath*) menant à un seul fichier à télécharger, et en sachant qu'il y en a une dizaine à télécharger et qu'il faut au minimum trois clics pour y accéder. On serait très vite perdu.

J'ai donc pris la décision de laisser cette partie en suspend et d'y revenir plus tard.

4.1.2 Chargement et traitement des fichiers

Les fichiers téléchargés sont au format xls ou dbf, un type de fichier de base de données que l'on peut ouvrir dans LibreOffice Calc. La première table dont je me suis occupé comptait 10 fichiers xls d'où il fallait extraire les informations et tout regrouper en un seul.

En ouvrant ces fichiers avec un éditeur de texte tel que le bloc-notes, leur structure est la suivante :

```
Colonne1   Colonne2   Colonne3
val11      val21      val31
val12      val22      val32
```

On voit que les lignes sont séparées par un retour à la ligne (\n en binaire) et les colonnes par une tabulation (\t en binaire). Il m'est donc venue à l'esprit de créer une fonction qui transforme ces fichiers xls en une liste de listes python, afin de pouvoir travailler dessus. Leur format ressemblerait à ceci :

```
[["Colonne1", "Colonne2", "Colonne3"],
["val11", "val21", "val31"],
["val12", "val22", "val32"]]
```

Très rapidement, je me suis rendu compte qu'avec des fichiers de plusieurs centaines de lignes, il y aurait beaucoup d'opérations à faire. Et même si Python n'est pas un langage très lent, il pêche un peu sur les boucles, en comparaison avec des langages compilés (tel que le C, C++, Java, etc..), me faisant hésiter sur cette méthode. Malheureusement, à ce moment du développement, je n'avais pas la connaissance de librairies pouvant effectuer des opérations facilement sur des fichiers.

J'ai donc décidé de continuer jusqu'à buter sur un problème plus conséquent. Pour me faciliter la tâche dans mon travail, j'ai eu l'idée de créer des dictionnaires en Python, afin de pouvoir itérer facilement à travers les colonnes et ainsi me faciliter quelques étapes du travail. Cependant, pour y arriver il fallait inverser la liste de listes que je venais de créer pour arriver à ce résultat :

```
{"Colonne1": ["val11", "val12"], "Colonne2": ["val21", "val22"],
"Colonne3": ["val31", "val32"]}
```

Cela correspond globalement à l'opération mathématique d'inversion d'une matrice donc ce n'est pas particulièrement compliqué à coder, mais cela rajoute beaucoup de lenteur dans le programme, en plus du fait qu'il faudrait effectuer l'opération inverse à la fin du programme.

C'est à ce moment-là que je me suis rendu compte que je ne pouvais pas continuer ainsi et après une rapide recherche sur internet, je découvre pandas, une librairie Python pour le chargement et l'extraction des données dans un fichier.

4.2 Tentative réussie

4.2.1 Chargement et traitement des fichiers

Grâce à Pandas, cette étape est très rapide à mettre en œuvre : la librairie fournit une multitude d'options pour ouvrir différents types de fichiers, rendant cette tâche très rapide et avec un risque d'erreurs très limité. Ces fichiers ont le nom de DataFrame* et se structurent de la manière suivante :

```
     Colonne1   Colonne2   Colonne3
0    val11      val21      val31
1    val12      val22      val32
```

La colonne sans nom et avec un 0 et 1 correspond aux index, cela permet d'accéder rapidement à une ligne grâce à l'attribut `loc` d'une Dataframe.

Dans tout le programme, j'utilise une "clé primaire" qui me sert à fusionner les fichiers, à ranger des valeurs dans un ordre et à sélectionner des cellules de ma DataFrame. Quelques fois, c'est la même clé primaire que celle dans la BDD, mais dans d'autres situations, j'ai dû utiliser une colonne commune à tous les fichiers, et n'ayant pas de cellules avec la même information. Cette dernière condition est primordiale pour les étapes suivantes.

Chaque fichier téléchargé contient des colonnes qui n'apparaissent pas dans la table donc il faut les supprimer. Aussi, une même colonne peut être contenue dans plusieurs fichiers donc pour éviter des problèmes, on ne garde qu'un exemplaire de chaque colonne.
Après avoir fait le tri, on fusionne les fichiers d'une manière similaire à une jointure SQL, qui est une fonction intégrée à pandas, en utilisant ma "clé primaire" comme point de jointure.

Enfin, un fichier final contenant toutes les colonnes nécessaires est créé et le nom des colonnes est traduit par leur nom dans la BDD. Ces quelques étapes semblent assez rapides, mais il y a beaucoup de traitement à effectuer en amont :

- Un formatage des valeurs de certaines colonnes par exemple 1 devient 'OUI' et 0 devient 'NON' ;
- La suppression des espaces entre les milliers des chiffres dans certaines colonnes, et le changement du type en un entier. Exemple : "12 345" devient 12345 ;
- La suppression des années dans certains noms de colonnes exemple : "Population de 2018" devient "Population de ". Cela permet d'éviter de devoir mettre à jour les fichiers de configuration tous les ans.

Toutes les tables ont leurs particularités au niveau du traitement des fichiers, c'est pourquoi il y a un programme Python par mise à jour de table. Chaque programme utilise en grande majorité des fonctions d'un fichier qui ne s'exécute pas mais sert uniquement de boite à outils pour les autres programmes. Chaque programme d'automatisation contient aussi des fonctions qui ne sont pas utilisées par les autres, car il y a des particularités pour chacun.
Cette méthode de programmation a été la plus simple pour moi, sachant que j'aurais pu faire un programme qui exécute la mise à jour de la même manière pour toutes les tables de la BDD, mais qui aurait été plus compliqué. En revanche, cette méthode a l'inconvénient de ne pas être facilement adaptative : certains programmes n'accepteront que des fichiers xls, un autre que du dbf et enfin un dernier qui prendra du dbf et xls sans problème. Ceci est dû principalement à des différences d'encodages des fichiers xls et dbf mais aussi à cause d'un traitement différent entre les deux.

A la fin du traitement et de la fusion des fichiers, on se retrouve avec une copie presque conforme de la table qui doit être mise à jour.

4.2.2 Mise à jour de la table

Dans les premières semaines, lors de la réflexion pour l'élaboration du programme, je devais me questionner sur la méthode de mise à jour lors de l'enregistrement des modifications dans la BDD. Deux idées me sont venues à l'esprit :

- La première consistait en 2 étapes simples à réaliser en SQL : d'abord, tronquer (vider) la table avant sa mise à jour et ensuite, la reremplir avec les nouvelles informations à jour ;
- La seconde consistait à générer des requêtes SQL à partir des données mises à jour et ensuite de mettre à jour chaque ligne une par une pour ne pas avoir à supprimer la table.

La première méthode avait l'avantage d'être bien plus simple que la seconde en termes de connaissances en SQL, mais nécessitait un peu plus de travail au niveau de la programmation Python. Étant plus confiant en Python qu'en SQL, la première solution s'avérait parfaite pour moi, en plus du fait qu'au moment du choix, je ne voyais aucun problème avec cette méthode.

Pour la mise à jour avec la première méthode, j'avais aussi prévu une option pour insérer les données dans une autre table que celle qui doit être mise à jour, afin de ne pas perdre les informations, et pour que je puisse toujours visualiser les changements. J'ai, jusqu'à la fin, choisit cette méthode de mise à jour pour les tests. Ce n'est que lors des derniers tests avant la livraison du programme que je me suis rendu compte du problème.

Si des tables autres que celle mise à jour possèdent des clés étrangères pointant sur la table à mettre à jour, selon la configuration de ces tables, des données peuvent être définitivement perdues lors de la suppression de la table mère. Il était bien entendu hors de question d'appliquer cette méthode donc, il fallait utiliser la seconde.

Ce problème était en réalité une bonne nouvelle car, en programmant, j'ai découvert des subtilités en SQL qui m'ont permis de simplifier le programme.

Ma seconde méthode de mise à jour se base sur des requêtes semblables à la suivante :

```
UPDATE nom_table SET colonne = 'charactères', colonne_a = chiffre WHERE colonne_cle_primaire = cle_primaire;
```

Il faut savoir que certaines informations présentes dans des tables ont été renseignées manuellement car il n'y a pas de source de mise à jour par des fichiers. Il serait très bête d'écraser ces précieuses informations par du vide quand on ne l'a pas donc, dans ma première méthode, une fonction s'occupait de comparer chaque cellule dans la table créée et l'ancienne table et si une cellule était vide dans la table créée, cela prenait l'information dans l'ancienne table. Cela fonctionnait très bien mais, avec la seconde méthode, un moyen encore plus simple m'a sauté aux yeux : si la cellule de la table créée est vide, on ne la met tout simplement pas à jour à travers la requête UPDATE. Ceci m'a aussi permis de simplifier les fichiers de configurations, où il fallait auparavant renseigner les colonnes dont on n'avait pas besoin de faire la mise à jour comme étant vides, déclenchant la fonction de comparaison sur ces cellules.

Le mode de fonctionnement de cette méthode est le suivant :

- En premier, on regarde s'il n'y a pas des éléments qui ont disparus, cela peut être des EPCI qui ont changé d'identifiant, ou s'il y a des nouveaux éléments tels que des communes qui fusionnent pour en faire une nouvelle ;
- Ensuite, on construit les requêtes SQL. Il y a surtout des requêtes UPDATE mais aussi des DELETE FROM pour les éléments disparus et enfin, des INSERT INTO pour ceux qui sont apparus ;
- Enfin, toutes ces requêtes sont exécutées pour mettre à jour la table dans la BDD.

La fonction qui crée ces requêtes est assez complexe car il faut veiller à tous les détails de SQL. Par exemple, une chaine de caractères doit être entourée d'apostrophes pour que SQL puisse comprendre. Et s'il y a déjà des apostrophes dans la chaine de caractères, il faut les doubler.

Si cette étape se passe bien, on peut alors afficher les changements qui ont eu lieux, et aussi enregistrer la table dans un fichier csv, consultable en cas de dysfonctionnement lors de l'entrée des données.

4.2.3 Fichiers de configuration

En programmant, je me suis rapidement rendu compte que beaucoup d'éléments du programme dépendent d'une personne devant renseigner des informations manuellement. La correspondance entre les colonnes des fichiers téléchargés et les colonnes de la table ne peut pas être devinée par le programme, il faut que quelqu'un l'indique. Les paramètres de connexion au serveur de BDD sont aussi renseignés dans ces fichiers.
Les fichiers ont une extension .ini, c'est un format de fichier standard pour la configuration, car ils sont simples à comprendre et on peut les modifier avec un simple éditeur de texte.
Voici un exemple de la structure d'un fichier .ini :

```
[bdd]
base = Nom de la base
utilisateur = Nom d'utilisateur
; Un commentaire
```

```
[variables]
nom_col_siren_bdd = siren_epci
nom_col_nom_epci_bdd = nom_epci
```

Un fichier .ini est divisé en sections identifiées par des crochets. Nous avons donc dans cet exemple 2 sections : bdd et variables. Dans ces sections, les variables sont identifiées par nom_de_la_variable = contenu de la variable. Les commentaires ne peuvent être écrits que sur toute la ligne et sont précédés d'un point-virgule.

Du côté de Python, ces fichiers peuvent être lus très facilement grâce à la librairie Configparser, qui accède aux fichiers comme s'ils étaient des dictionnaires en langage Python.

4.2.4 Interface graphique

Bien que la méthode des fichiers de configuration fonctionne correctement pour éviter que les agents de la DDT aillent dans les fichiers Python, elle n'est pas très conviviale, surtout quand on a l'habitude des interfaces graphique. Voilà pourquoi j'ai choisi d'en réaliser une pour le programme.

Les langages de prédilection pour cette partie de la programmation sont le C et le C++ car c'est avec ces langages que Windows est principalement écrit. Cependant, plusieurs librairies permettent la création d'interfaces graphique sous Python, notamment Tkinter, PySimpleGUI, PyQt et bien d'autres. La dernière d'entre elles, PyQt, m'intéresse tout particulièrement car elle propose une extension à la librairie pour construire des interfaces graphiques à partir d'un logiciel. Le principe est simple : sur le panneau de gauche (cf. Figure 2), on peut sélectionner des éléments et venir les déposer à l'endroit voulu sur la fenêtre au milieu. Sur le panneau de droite, on peut éditer le nom de chaque élément et configurer leur fonctionnement.

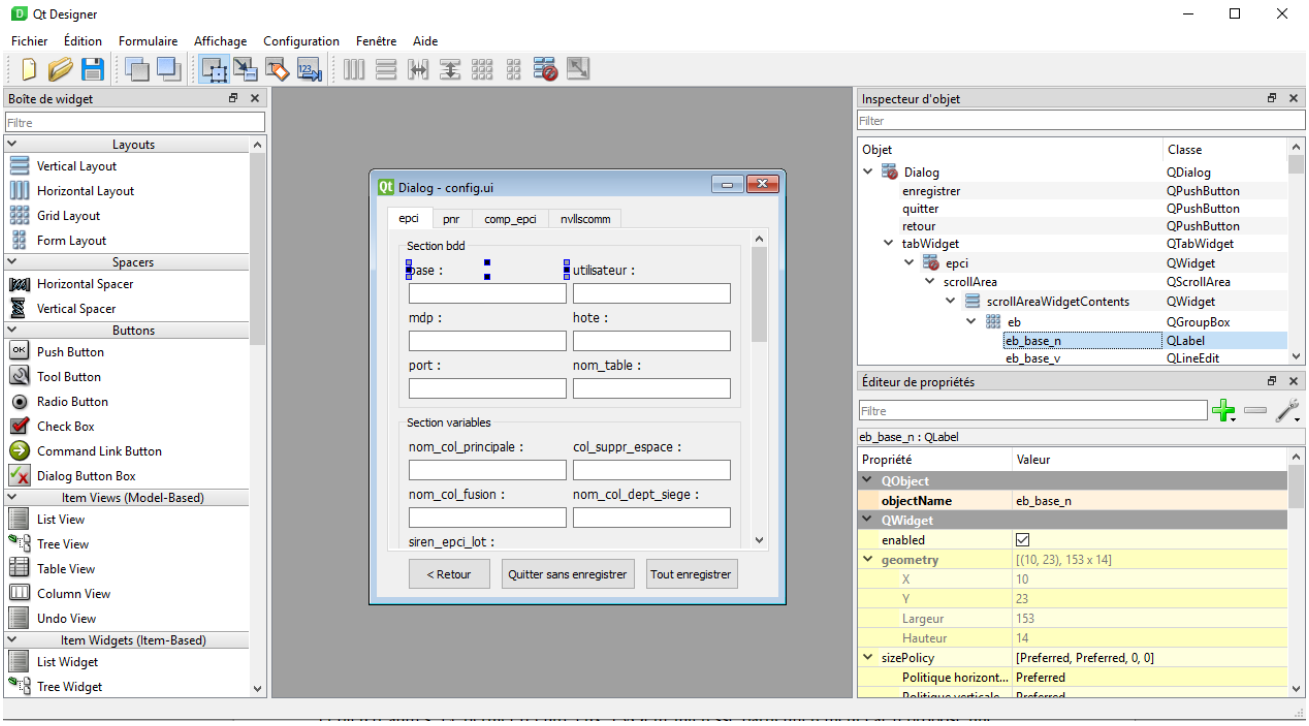

Figure 2 : Capture du logiciel Qt Designer

Ce logiciel génère des fichiers .ui avec une structure semblable à du XML qui peut ensuite être traduite en code Python grâce à une commande fournie par PyQt5. Le seul inconvénient de cette méthode, c'est que pour des interfaces avec beaucoup d'éléments, les fichiers Python générés peuvent faire plusieurs centaines de lignes, rendant leur compréhension assez complexe.

Voici les 4 fenêtres constituant mon interface graphique :

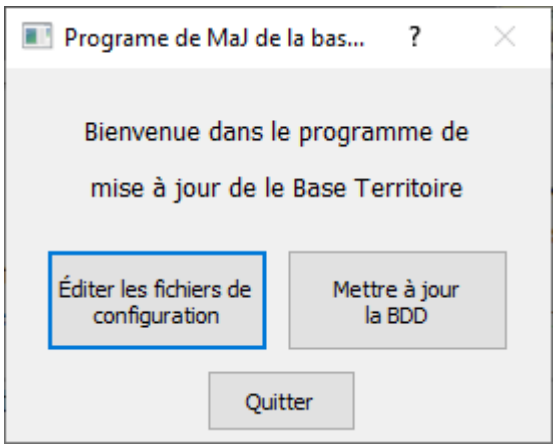

Figure 3 : Fenêtre d'initialisation

On peut, sur cette fenêtre, choisir entre mettre à jour la base de données ou modifier les fichiers de configuration.

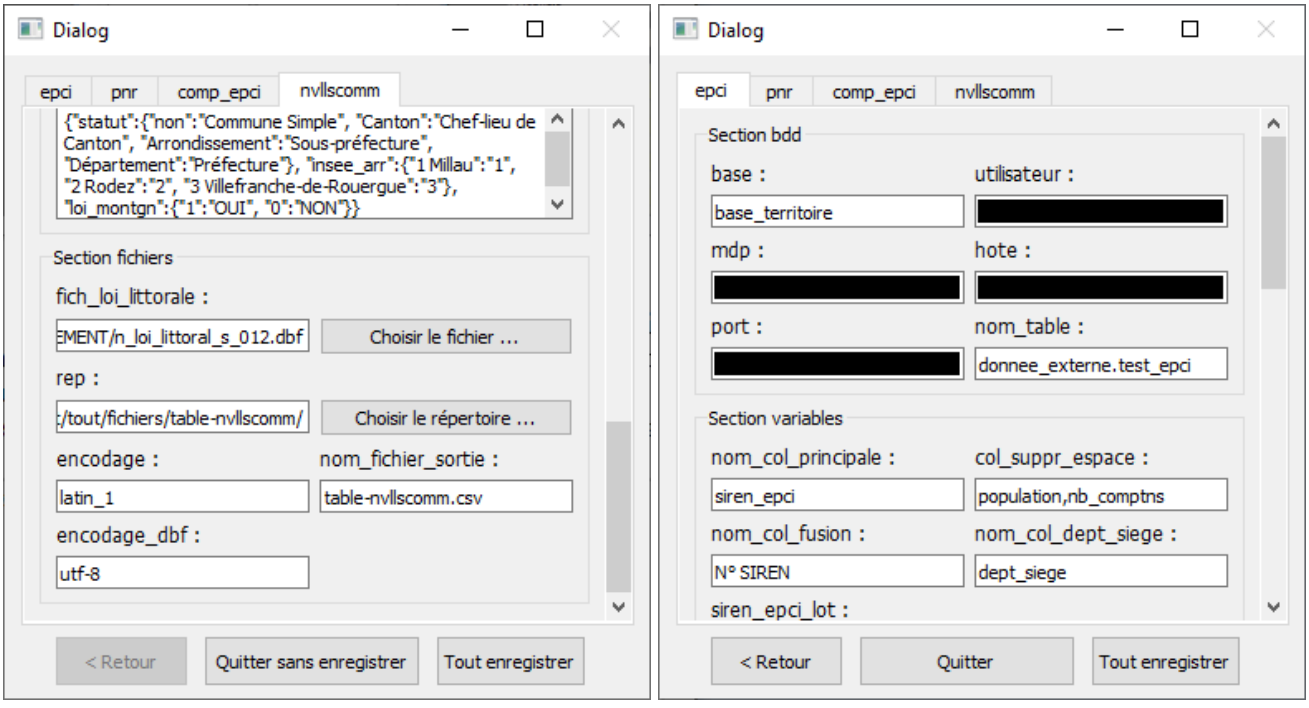

Figure 4 et 5 : interface de configuration

La fenêtre de configuration charge le contenu des fichiers de configuration dans leurs cellules respectives au lancement de l'interface. On peut ensuite modifier les paramètres et enregistrer.

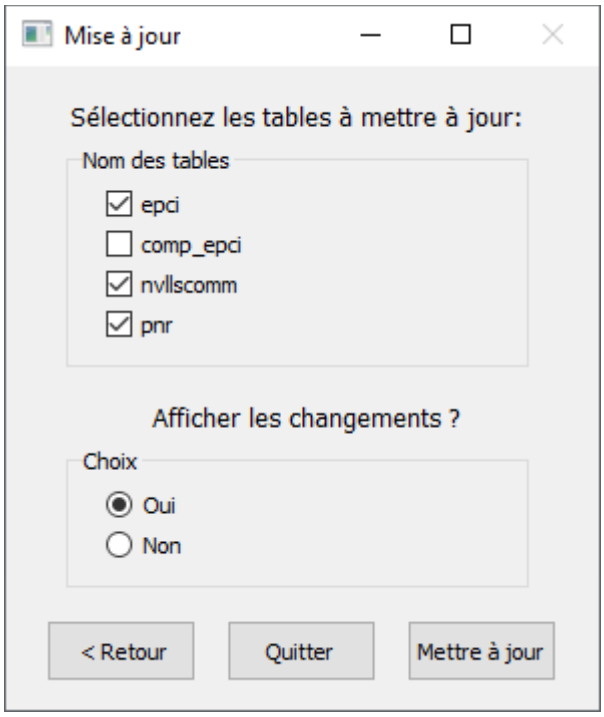

Figure 6 : Interface de mise à jour

On peut ici choisir les tables à mettre à jour ainsi que si on veut afficher les changements effectués dans la BDD. Cette option est importante car je n'ai probablement pas pensé à toutes les erreurs possibles donc il sera plus simple de les voir dans le terminal si les différences ne sont pas affichées.

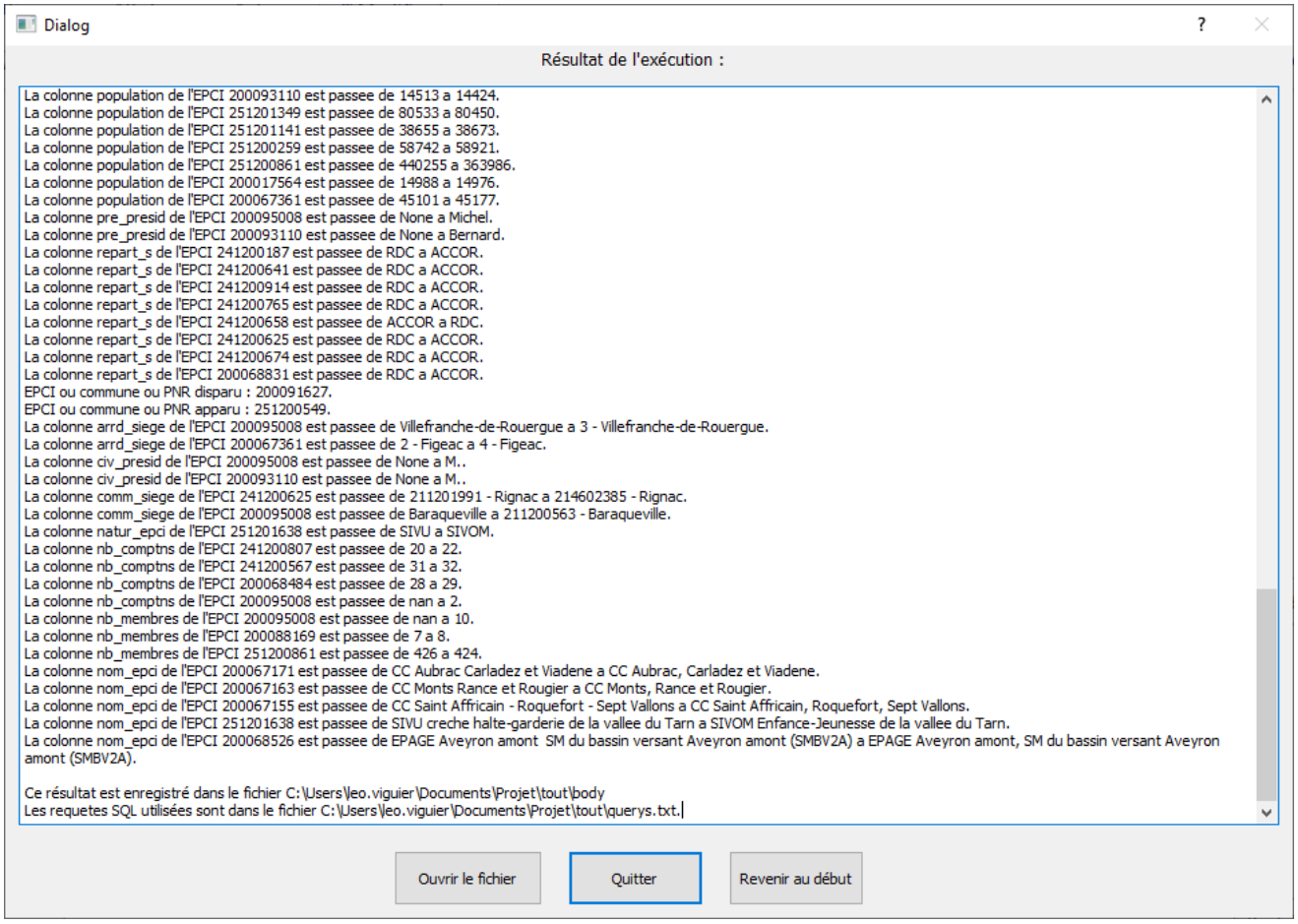

Figure 7 : Fenêtre du résultat de l'exécution

Dans cette dernière fenêtre s'affiche tous les changements effectués dans la BDD et aussi les erreurs que j'ai prévues, et que j'ai documentées. Le bouton "Ouvrir le fichier" permet d'ouvrir le fichier contenant tous ces changements. Je sauvegarde aussi toutes les requêtes SQL générées dans un fichier texte qui peut été consultable en cas d'erreur d'insertion des données dans la BDD.

4.2.5 Installation

Pour l'étape d'installation, j'avais encore une fois un choix à faire : je pouvais soit utiliser un compilateur pour Python tel que Cython ou Nuitka qui permet de rendre les fichiers exécutables sur n'importe quel ordinateur, soit installer une bonne version de Python avec toutes les librairies nécessaires.
En effet, toutes les versions de Python ne fonctionnent pas avec certains éléments que j'ai programmés. Lors de mes tests sur la version de Python 3.8.1, une erreur *RuntimeError: dictionary changed size during iteration* apparaissait pour chaque mise à jour de table car au passage de Python 3.7 à 3.8, le comportement des dictionnaires a changé, empêchant une importante partie de mon programme de fonctionner. J'ai donc pris la décision de faire fonctionner mon programme uniquement de la version 3.7.1 à 3.7.10.

La première méthode d'installation à l'avantage d'être extrêmement simple pour les utilisateurs car ils auraient juste à décompresser l'archive zip et à double cliquer sur l'application. Cependant, cette méthode demanderait de recompiler les fichiers en cas de modification, une chose qui n'est pas aisée car il faudrait, selon le logiciel, un compilateur C ou C++ d'installé sur l'ordinateur ou encore une version de Python fonctionnelle.

Avec la seconde méthode, deux scripts en langage BATCH* me sont nécessaires : un pour tester si Python est installé avec une bonne version et l'installer si ce n'est pas le cas, et un autre pour installer les librairies.
L'étape d'installation vérifie d'abord si l'ordinateur possède une version de Python entre 3.7.1 et 3.7.10 et si ce n'est pas le cas, on lance le téléchargement de Python 3.7.8 (la version sur laquelle j'ai fait le programme) puis, on l'installe.
Lors de l'installation, cette nouvelle version de Python est ajoutée à la variable d'environnement PATH*, et donc Windows la reconnait comme un Python "par défaut". Si une autre version de Python est déjà présente par défaut dans Windows, on ne va pas mettre à jour le PATH, car cela peut avoir de graves conséquences sur les logiciels déjà installés sur l'ordinateur de l'utilisateur. À la place, à chaque fois qu'on devra utiliser Python 3.7.8, on spécifiera le chemin absolu menant à l'exécutable.
Ce sont donc trois cas qui sont traités :

- Python est dans le PATH et dans la bonne version ➔ il n'y a rien à faire ;
- Python est dans le PATH mais n'est pas dans la bonne version ➔ on installe Python 3.7.8 et on crée le fichier contenant le chemin pointant sur l'exécutable ;
- Python n'est pas dans le PATH ➔ on installe Python 3.7.8 en inscrivant le chemin vers son exécutable dans le PATH et on ne crée pas le fichier pointant sur l'exécutable.

Le deuxième script BATCH s'occupe du téléchargement des librairies nécessaires au programme.
Il faut avoir deux scripts BATCH car au lancement d'un terminal, celui-ci charge en mémoire la variable d'environnement PATH et il n'y a pas de moyen direct pour faire comprendre au terminal que celle-ci a changée et qu'elle doit être rechargée.

4.2.6 Lancement

Avant le lancement du programme, il faut aller télécharger les fichiers nécessaires manuellement, car il n'y a pas de solutions facilement adaptatives. J'ai bien réussi à automatiser le téléchargement de deux fichiers, mais ceux-ci constituaient une trop grande source d'erreur, d'autant plus que le lien d'un des fichiers a été interdit pas le proxy entre temps. J'ai donc pris la décision de ne pas automatiser le téléchargement, malgré le fait que cela aurait grandement accéléré la tâche d'automatisation, et, aurait contribué à la réduction des erreurs humaines.
Étant donné qu'il y a un certain nombre de fichiers à télécharger et à ranger au bon endroit, un script affichant l'arborescence du programme est disponible, afin de la comparer avec celle dans la documentation :

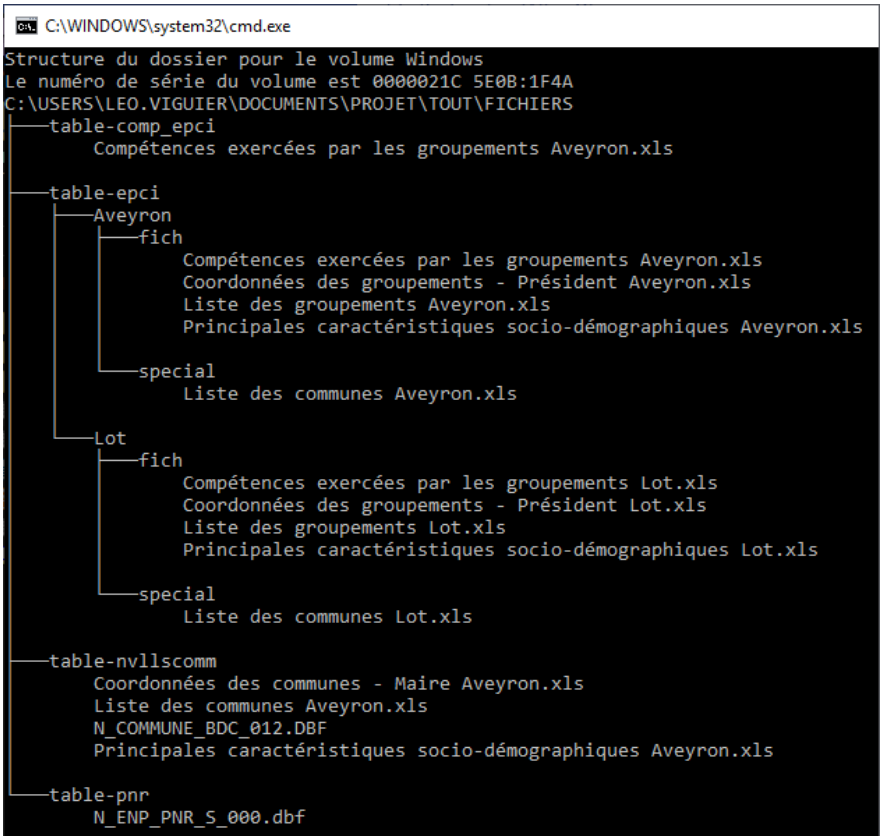

Figure 8 : Arborescence des fichiers requis.

Pour le lancement du programme, il y a là encore, deux scripts. Le premier lance l'interface graphique et le second lance un terminal avec des questions à répondre pour effectuer la mise à jour.
L'intérêt d'avoir deux méthodes de lancement est évident : si l'interface graphique venait à ne plus fonctionner, il y aurait toujours une solution de replis pour mettre à jour la BDD.
Avant le lancement du programme, une sauvegarde de la BDD est effectuée, au cas où une erreur dans mon programme casserait tout. Le nom de cette sauvegarde est constitué de la date et heure, afin de les trier facilement car la plus vieille est supprimée s'il y en a plus de dix.
Le script en ligne de commande se présente de la manière suivante :

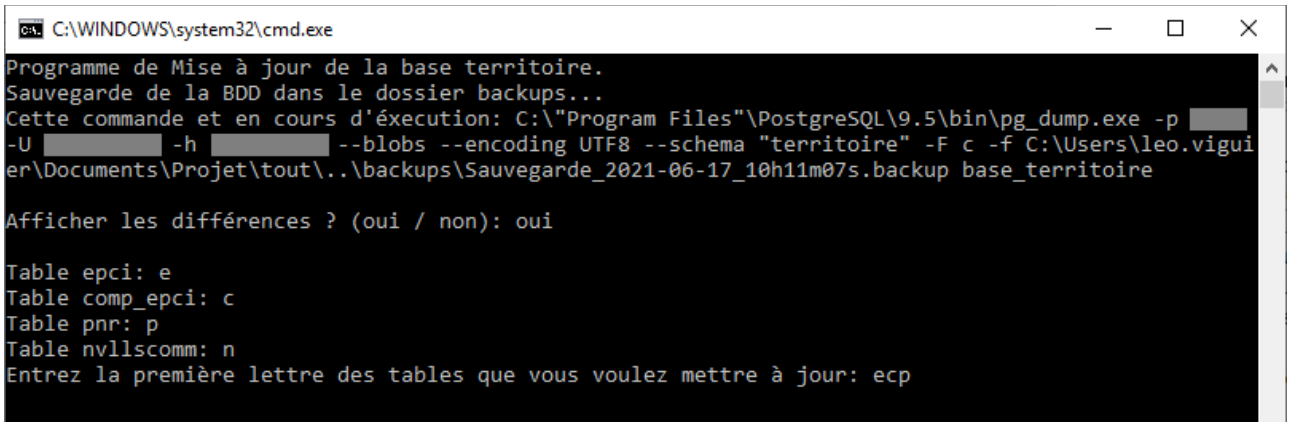

Figure 9 : Capture de l'exécution du programme en ligne de commande.

4.2.7 Détails de programmation et Documentation

Il va de soi que tout au long de la programmation, il faut tester le programme, et trouver les potentielles erreurs que les utilisateurs pourraient faire. En effet, une grande partie du programme dépend de l'utilisateur, et il est important de trouver les erreurs qu'ils pourraient générer, afin de leur donner une voie de correction. Cette étape de la programmation est d'autant plus importante, en sachant que ces utilisateurs n'ont pas de connaissances particulières en Python, car il serait difficilement concevable pour eux de décrypter les erreurs affichées par Python.
Toutes les erreurs sont donc écrites avec un code que l'on peut aller consulter dans la documentation, avec une explication succincte dans le message d'erreur.
Dans la documentation se trouve aussi des notes pour l'utilisation, la modification, la configuration et enfin, l'installation.

Les chemins vers les fichiers sont aussi un point primordial à ne pas oublier. Dans le cas d'un programme mal construit, il faudrait se situer dans le bon dossier contenant l'exécutable au lancement du programme. Or, rien n'empêche l'utilisateur de créer des raccourcis vers le bureau par exemple, afin d'accéder au programme plus rapidement. Pour pallier ce problème, chaque accès à un fichier est fait avec un chemin absolu, construit en fonction du répertoire de l'utilisateur et de la localisation du programme.

5 Missions secondaires

5.1 Interface d'extraction de données

Tous les agents de la DDT n'ont pas les compétences en PostgreSQL pour pouvoir consulter la BDD lors de leur travail, c'est pourquoi ils font appel au pôle SIG qui créent les requêtes SQL et enregistrent le résultat dans un fichier csv que les agents pourront consulter. Ce processus n'est pas idéal car il prend du temps prend du temps pour les agents et nécessite une intervention du pôle SIG.
C'est pourquoi il m'est demandé de réaliser un programme qui permettrait d'interroger la BDD et de sauvegarder le résultat dans un fichier csv.
Le programme permet de faire des extractions relativement simples en se limitant à devoir choisir une table, au moins une colonne et au moins une ligne, en fonction de la clé primaire.
Pour ce qui est de faire des extractions simples, c'est maintenant possible pour tous les agents. En revanche, il n'est pas possible de créer des requêtes SQL avec une jointure ou des requêtes imbriquées. Pour cela, il faut encore passer par le pôle SIG, mais cette fois-ci, les requêtes peuvent être enregistrées sous un nom compréhensible, afin d'être retrouvées rapidement par les agents.
J'ai donc encore une fois utilisé PyQt5 pour la réalisation de l'interface graphique :

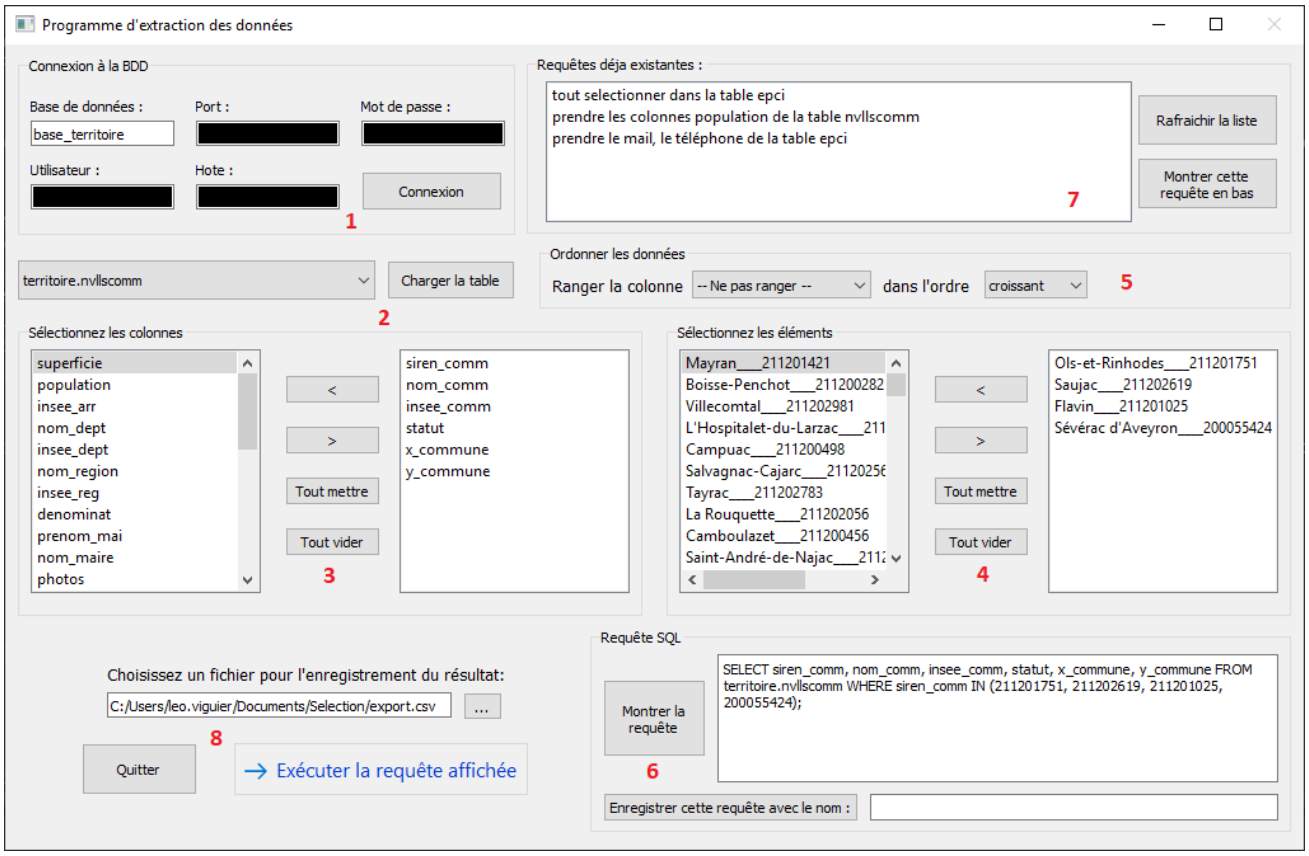

Figure 10 : Capture du logiciel d'extraction des données

La première chose à faire est de se connecter à la BDD (1). Ces informations de connexion sont stockées dans un fichier .ini, en suivant le même principe que dans le programme précédent. Une fois connecté, on peut sélectionner une table parmi celles dans la liste déroulante (2) et charger le nom des colonnes et la valeur de la clé primaire de chaque lignes (3 et 4).
Au moment du chargement, les colonnes et valeurs de clé primaires se situent dans la liste de gauche et on peut les sélectionner puis cliquer sur la flèche > pour effectuer la sélection de cette colonne, ce qui les amènes à droite. Il va de même avec la liste des clés primaires.
Enfin, il y a la possibilité d'ordonner les lignes en fonction d'une colonne (5)
Une fois les sélections faites, on peut cliquer sur "Montrer la requête" (6), ce qui appelle une fonction pour construire la requête SQL en fonction des sélections. Il y a aussi l'option d'enregistrement avec

un nom, cela permet aux agents de rapidement retrouver leurs requêtes SQL, et de ne pas perdre de temps à les reconstruire. Ces enregistrements sont listés en haut à droite (7), avec un nom compréhensible pour tous.
Enfin, on choisit la destination du fichier csv qui contiendra le résultat de la requête et on peut l'exécuter (8).

Il y a bien entendu une brève documentation pour son utilisation ainsi que des messages quand des cases ne sont pas complètes, ou en cas d'erreurs.

5.2 Présentation de Python au pôle SIG

Une introduction à Python était demandée pour permettre aux agents du pôle SIG de pouvoir comprendre pourquoi telle ou telle chose était écrite dans le programme de mise à jour. Cette introduction permettrait aussi d'avoir une chance de résoudre les erreurs que j'aurais oubliés dans le programme.
Cette introduction se devait en fait d'être un peu plus que des simples bases de Python, car dans le programme, j'ai utilisé tout ce que l'on a appris en cours, et même plus. J'ai donc abordé les thèmes suivants :

Variables et Types
C'est selon moi le cœur de la programmation et il convient donc d'expliquer comment définir une variable, mais aussi de montrer les actions sur les types les plus récurrents, qui sont susceptibles d'être rencontrés le plus de fois. J'ai donc parlé des types str, int, float, list et bool, en listant les transformations de types possible ainsi qu'un exemple de valeur et de mode de création.
Une partie était dédiée uniquement au type str, où j'y donnait une liste des attributs les plus utiles et les plus commun, tels que `.replace()`, `.split()`, `.upper()` et `.lower()`. Il y a aussi une brève liste montrant les options de créations d'une chaine de caractères (addition, `.format()` et f-string) et enfin une explication sur le slicing*.

If-else, opérateurs logiques et mathématiques
Les opérateurs mathématiques sont décrits dans un tableau récapitulatif où l'on peut trouver l'opérateur en Python, les types supportant l'opération et des exemples d'opérations.
Il va de même pour les opérateurs logiques et de comparaison, dont leurs comportements face aux deux booléens False et True sont décrits dans des tableaux individuels.
Enfin, If-else montre un exemple sur les tests de conditions, tout en introduisant à l'importance de l'indentation en Python.

Index et boucles
Dans la liste des types que j'ai fournie, deux d'entre eux sont itérables, et sont aussi indexés en partant de 0 jusqu'à leur longueur - 1. Un rapprochement est fait avec la fonction `range(n)`, qui crée un itérable s'arrêtant à n-1, idéal pour accéder aux valeurs d'une liste par exemple.

Fonctions, dictionnaires, fichiers...
J'ai expliqué dans cette partie des choses un peu plus avancée pour une simple introduction à Python, mais que j'ai beaucoup utiliser dans le programme, notamment les dictionnaires, que les utilisateurs devront peut-être modifier un jour. Les `class` en Python sont aussi expliquée, bien que je n'aie pas abordé le thème pendant la présentation car c'est bien trop avancé, mais je l'ais aient utilisés donc il est important de les expliquer.
La gestion des erreurs est aussi abordée, ainsi que comment retrouver la source d'une erreur, car cela peut arriver, même si j'ai fait de mon mieux pour en trouver le plus possible, il en reste sûrement.
Enfin, une courte liste des raccourcis utilisés dans le programme tels que des boucles for sur une ligne, des tests de condition sur une ligne, et, plus communément, les raccourcis d'opérations mathématiques.

J’ai donc résumé deux semestres de programmation en deux heures. Cette présentation n’était pas pour but qu’ils comprennent tout d’un coup car c’est impossible, mais plutôt comme une explication orale de ce qui est écrit dans le diaporama, afin de retrouver plus vite des éléments rencontrés dans le programme ou ailleurs.

6 Conclusion

Durant mes 10 semaines de stage j'ai pu élargir mes compétences en programmation, grâce à ma mission principale où j'ai beaucoup appris sur les techniques de programmation dans un milieu professionnel. Cette mission étant demandée depuis quelques temps, je me devais d'y répondre avec la plus grande rigueur et sérieux.

Bien que le point sur la portabilité du programme vers d'autres DDT ne puisse pas être vérifié, j'ai utilisé tous les moyens à ma disposition pour rendre cette tâche faisable. Les autres exigences ont été réalisées avec succès, ce que j'ai pu observer après les tests réussis de mes collègues.

Ce stage a été très enrichissant, que ce soit sur un plan technique ou sur un plan personnel. Les recherches et manipulations que j'ai réalisé ces 10 dernières semaines m'ont permis d'acquérir une meilleure autonomie et de nouvelles connaissances, en plus de celles acquises au cours de mes études. J'ai ainsi appris à observer les bonnes pratiques lors de la programmation d'un logiciel (documentation, description du programme, clarté d'écriture) qui sont devenues des automatismes au fil des semaines.

Cette première expérience très positive du monde professionnel me conforte dans l'idée de poursuivre mes études en école d'ingénieur dans le domaine de l'informatique, mais aussi dans l'administration système, un domaine que j'apprécie tout autant.

7 Remerciements

Je tiens tout d'abord à adresser mes remerciements à Thierry CASTAN, qui m'a aidé à trouver ce stage malgré cette période compliquée.

Je tiens aussi à remercier Marvin GRAHAM-FONSECA, qui m'a accueilli au SATUL et guidé dans mon travail à réaliser au cours de la première semaine.

Je remercie également ma tutrice de stage, Aurélie BONNEFIS pour ses conseils, son temps et les connaissances techniques qu'elle m'a accordé.

J'exprime enfin ma reconnaissance à Fabrice PAGNUCCO, à Jean-François AGNEL, au SATUL ainsi qu'à toute la DDT pour m'avoir accueilli chaleureusement au sein de cette administration.

8 Glossaire

DUT, Diplôme Universitaire de Technologie

Python, Langage de programmation

PostgreSQL, Langage de base de données dérivé de SQL.

JavaScript, Langage de programmation utilisé par pratiquement tous les sites internet, afin de rendre la navigation plus agréable.

Socket, Interface logicielle permettant à un développeur de facilement accéder aux services de connexion à internet.

Cookies, En français, témoin de connexion permettant d'améliorer la navigation dans un site, et permettant notamment d'empêcher la déconnexion d'un site pendant un changement de page sur ce même site.

XPath, Un langage de requête pour localiser une portion d'un document XML. Une page web étant structurée de manière similaire à un document XML, le Xpath y fonctionne aussi très bien.

DataFrame, Un tableau similaire à un fichier calc, mais en Python.

Batch, Langage de programmation de Windows

PATH, Variable contenant tous les chemins de répertoires menant à des exécutables. Quand on écrit le nom d'un logiciel ou d'une commande dans un terminal, l'ordinateur va regarder dans tous les répertoires listés par le PATH pour aller l'exécuter. Cela évite de devoir écrire des chemins absolus.

Slicing, Méthode d'accès à un morceau de chaine de caractères ou, plus généralement, à un itérable indexé.