

TRAVAIL DE FIN D'ETUDE

En vue de l'obtention du diplôme de bachelier en biotechnique

Option bioinformatique



**MISE EN PLACE D'UNE SOLUTION DE BASE DE DONNÉES POUR LE
STOCKAGE DES DONNÉES BIOINFORMATIQUES ISSUES DE GENBANK**



Présenté par :

TONGANG Marie Carelle

Promoteur : **Monsieur David Coornaert**

Année académique 2022-2023

Session d'août 2023

**MISE EN PLACE D'UNE SOLUTION DE BASE DE DONNÉES POUR LE
STOCKAGE DES DONNÉES BIOINFORMATIQUES ISSUES DE GENBANK**

Remerciements

Avant tout, je tiens à exprimer ma sincère gratitude envers Monsieur David Coornaert, qui m'a apporté un soutien inestimable tout au long de ce travail de fin d'études. Malgré ma situation, il a été disponible et à l'écoute.

Ensuite, mes remerciements les plus profonds vont à ma famille et à mes amis pour leurs encouragements, et plus particulièrement à Dowono Joel. Sa présence constante pendant les moments difficiles, ses mots réconfortants et sa confiance en moi, même lorsque je doutais, ont été d'une valeur inestimable.

Enfin, je souhaite exprimer ma reconnaissance envers l'ensemble des enseignants de la Haute École en Hainaut. Leur remarquable travail pédagogique et les connaissances qu'ils m'ont transmises tout au long de cette formation méritent d'être soulignés.

Avec tout mon cœur,

Tongang Marie Carelle

GENBANK représente une base de données biologiques gérée par le NCBI, offrant des informations précieuses pour les recherches en bioinformatique. Elle permet d'explorer divers aspects des mécanismes biologiques et de mener des études approfondies sur la vie sous différents angles.

Cependant, la taille considérable de genbank et la fréquence limitée des mises à jour par le NCBI (tous les deux mois) peuvent parfois rendre difficile la recherche d'informations spécifiques. L'idée qui a émergé est la mise en place d'un système de stockage dédié pour les données provenant de genbank. Ce système visera à apporter davantage d'ordre et de précision, à faciliter l'utilisation et l'exploitation des données. L'approche consistera à regrouper les organismes en fonction de leur famille, permettant ainsi aux biologistes de retrouver aisément tous les êtres vivants appartenant à une famille spécifique, simplifiant ainsi leurs recherches par rapport à genbank directement.

Le processus débutera par la compréhension de l'arbre taxonomique conçu par Monsieur David Coornaert, puis se poursuivra par le nettoyage de cet arbre, la gestion des mises à jour et l'intégration d'un système d'alerte. Par la suite, cette méthodologie sera appliquée à une base de données, permettant ainsi de comparer les deux systèmes de stockage et de sélectionner celui qui se révèle le plus convivial et optimal pour répondre aux besoins spécifiques.

Summary

GENBANK represents a biological database managed by the NCBI, providing valuable information for bioinformatics research. It enables the exploration of various aspects of biological mechanisms and facilitates in-depth studies of life from different perspectives.

However, the substantial size of genbank and the limited frequency of updates by the NCBI (every two months) can sometimes make it challenging to search for specific information. The idea that emerged is to establish a dedicated storage system for data from genbank. This system aims to bring more organization and precision, facilitating the use and exploitation of data. The approach involves grouping organisms based on their families, allowing biologists to easily find all living beings belonging to a specific family, thereby simplifying their searches compared to directly using genbank.

The process will commence with understanding the taxonomic tree designed by Mr. David Coornaert, followed by the cleansing of this tree, management of updates, and integration of an alert system. Subsequently, this methodology will be applied to a database, enabling a comparison of the two storage systems and selecting the one that proves to be the most user-friendly and optimal for meeting specific needs.

Table des matières

I.	Introduction.....	10
A.	La base de données genbank du NCBI.....	11
B.	Composition de la base de données Genbank	12
C.	Description de quelques champs de genbank (18)	13
II.	Objectifs de ce travail	14
III.	MATERIEL ET METHODES	15
A.	Mise en place de l'arbre taxonomique.....	15
1.	Analyse d'un fichier au format genbank	15
2.	Quelques exemples d'organismes au format genbank	16
3.	Problème rencontré	17
4.	Comment fait biopython pour parser ?.....	18
5.	Solution apportée par Monsieur David Coornaert.....	21
6.	Description de l'arbre	22
7.	Nettoyage de l'arbre taxonomique et mise à jour des fichiers	24
8.	Ajout d'un système d'alerte dans l'arborescence	28
B.	MISE EN PLACE DE LA BASE DE DONNEES	30
1.	Stratégie de création des tables constituant la base de données	31
2.	Création de la base de données et des tables.....	31
3.	La gestion des mises à jour dans la base de données	35
4.	Ajout d'un système d'alerte	38
IV.	DISCUSSION ET PERSPECTIVES	41
V.	CONCLUSION	42
VI.	REFERENCES BIBLIOGRAPHIQUES	43
VII.	ANNEXES.....	45

Liste des abréviations

Abréviation	Signification
NCBI	National Center for Biotechnology Information
ADN	Acide Désoxyribonucléique
DDBJ	DNA Data Bank of Japan
EMBL	European Molecular Biology Laboratory
NLM	National Library of Medicine
NIH	National Institutes of Health
FTP	File Transfer Protocol
ORG	Organisme
REF	Référence
BIG	Ordinateur de Monsieur Coornaert, adresse IP 10.1.120.50
BDD	Base de données
Id	Identifiant
Gb	Fichier contenant les identifiants d'une famille donnée
SQL	Structured Query Language

Liste des figures

Figure 1. Une entrée genbank du gène *Synechococcus elongatus* PCC.

Figure 2. Un extrait du fichier gbbct1.seq au format genbank.

Figure 3. Présentation de l'organisme AB000389 au format genbank, de la ligne du champs LOCUS et des lignes comprises entre les champs ORGANISM et REFERENCE.

Figure 4. Présentation de l'organisme AB425276 au format genbank.

Figure 5. Taxonomie de l'organisme AB425276 réalisée par biopython.

Figure 6. Explication du mécanisme utilisé par biopython pour parser.

Figure 7. Présentation des premiers éléments des lignes comprises entre ORG et REF du fichier gbbct1.seq.bz2.

Figure 8. Présentation de l'organisme AB679348 qui illustre "Bacteria."

Figure 9. Taxonomie de l'organisme AB679348 réalisée par biopython.

Figure 10. Les familles principales de l'arbre taxonomique.

Figure 11. Présentation graphique d'un extrait de l'arbre (famille des Archaea).

Figure 12. Présentation du premier niveau de la famille des Viruses.

Figure 13. Présentation graphique de la famille des Viruses et contenu d'un fichier gb.

Figure 14. Les fichiers de genbank utilisés pour ce projet.

Figure 15. Logigramme établi pour nettoyer et mettre l'arborescence à jour.

Figure 16. Affichage du contenu de gbdel.txt.gz.

Figure 17. Traitement du fichier gbdel.txt.gz.

Figure 18. Rapport envoyé par mail en cas de modification de la famille qui nous intéresse.

Figure 19. Script alerte.py.

Figure 20. Programme DbVisualiZer utilisé.

Figure 21. Requête qui compte le nombre de lignes de la table family.

Figure 22. Requête qui compte le nombre de lignes de la table "organism".

Figure 23. Requête qui montre le détail des dix premiers identifiants de la table "organism" appartenant au chemin mentionné.

Figure 24. Requête qui affiche tous les enfants directs de la famille des Viruses, en d'autres termes il s'agit des familles ayant pour id_parent = 1.

Figure 25. Requête qui affiche le nom de tous les enfants directs de la famille des “Archaea” ; en d’autres termes il s’agit des familles ayant pour id_parent = 7467.

Figure 26. Requête qui affiche les organismes dont la date de mise à jour actuelle est plus récente que la dernière date de mise à jour (expérience1).

Figure 27. Requête qui affiche les organismes dont la date de mise à jour actuelle est plus récente que la dernière date de mise à jour (expérience 2).

Figure 28. Script organisms_update.py.

Figure 29. Requête de vérification des identifiants qui ont été mis à jour.

Figure 30. Rapport envoyé par mail en cas de modification.

Figure 31. Le script monitoring.py.

Liste des tableaux

Tableau 1. Taxonomie de Genbank.

I. INTRODUCTION

La discipline de la bioinformatique a pris forme dans les années 70, grâce notamment aux bases de données EMBL et GENBANK (4). Elle met à disposition des outils et des logiciels pour gérer, analyser et exploiter les informations génétiques, génomiques et protéomiques en vue de produire de nouvelles connaissances (4).

Les bases de données jouent un rôle crucial en fournissant les données primaires à la bioinformatique, ce qui contribue directement à la génération de nouvelles connaissances. Dans le domaine de la bioinformatique, trois bases de données généralistes sont mondialement reconnues : GENBANK du NCBI (1), la base de données d'ADN du Japon (DDBJ) (2), la base du laboratoire de biologie moléculaire européen (EMBL) (3).

Ces bases de données ont un point commun : elles font toutes partie de la collaboration "International Nucleotide Sequence Databases" et possèdent une structure similaire (1). Elles collaborent étroitement pour s'enrichir mutuellement et garantir un ensemble d'informations cohérent et exhaustif (1).

Dans ce travail de recherche, nous nous intéresserons à la banque de données genbank et nous tenterons de concevoir un système de stockage des données provenant de genbank qui soit plus convivial, efficace et facile à gérer. Le premier système de stockage des données est un arbre taxonomique conçu par Monsieur David Coornaert. Mon rôle consistera à comprendre, puis expliquer la mise en place de cet arbre, à le nettoyer, à gérer les mises à jour, ensuite mettre en place un système d'alerte en cas de modification. Le deuxième système de stockage sera une base de données dans laquelle nous appliquerons des traitements similaires. L'objectif sera de les évaluer pour déterminer quel système correspond le mieux à nos besoins.

Cette étude est menée en raison des contraintes inhérentes à la base de données genbank du NCBI. Malgré son contenu général, ces données nécessitent des traitements complexes pour être utilisées efficacement par les phylogénéticiens. Plusieurs problèmes ont été identifiés, incitant cette recherche. Les requêtes complexes employées par les bioinformaticiens sont fréquemment lentes, en grande partie en raison de l'ampleur constante de la base de données. En tant qu'utilisateurs, notre influence sur genbank est restreinte, dépendant des mises à jour de NCBI. De plus, les outils de filtrage de données ne satisfont pas toujours nos besoins spécifiques, nous forçant à nous adapter aux méthodes de genbank.

A. La base de données genbank du NCBI

Fondée en 1982 par la société IntelliGenetics et gérée par le NCBI à Los Alamos, aux États-Unis, genbank est une base de données publique pour les séquences d'acides nucléiques (4). Depuis sa création, genbank a connu une croissance significative, avec son contenu doublant tous les 18 mois et des mises à jour tous les deux mois. En octobre 2006, elle comptait par exemple 66 925 938 907 nucléotides (4).

Genbank renferme des séquences nucléotidiques accessibles au public pour près de 260 000 espèces dûment répertoriées (18). Elle est élaborée et diffusée par le National Center for Biotechnology Information (NCBI), un département de la National Library of Medicine (NLM), localisé sur le campus des National Institutes of Health (NIH) à Bethesda, dans le Maryland, aux États-Unis (5).

Les sources principales de ces séquences sont les soumissions des laboratoires individuels et les projets de grande envergure tels que les séquençages complets de génomes et les échantillonnages environnementaux (1). Genbank est accessible via le système de recherche Entrez du NCBI, intégrant des données issues des principales bases de données de séquences d'ADN et de protéines, ainsi que des informations en lien avec la taxonomie, le génome, la cartographie, la structure et les domaines protéiques, ainsi que la littérature biomédicale par le biais de PubMed. (1). Des versions complètes de genbank sont publiées bimestriellement, et des mises à jour quotidiennes sont accessibles via FTP (18).

Pour accéder à genbank et à ses services d'exploration et d'analyse connexes, il suffit de se rendre sur la page d'accueil du NCBI : (1) qui met gratuitement à disposition les données de genbank sur Internet, à travers le protocole FTP, ainsi que par le biais d'une gamme étendue de services en ligne dédiés à la recherche et à l'analyse (6).

B. Composition de la base de données Genbank

La base de données de taxonomie constitue une classification et une nomenclature minutieusement construite pour englober tous les organismes présents au sein des bases de données publiques de séquences. (7).

Genbank est partagée en 18 divisions que sont présentées dans le tableau 1 (18).

1.	PRI	Séquences des primates
2.	ROD	Séquences des rongeurs
3.	MAM	Séquences des autres collagènes de mammifères
4.	VRT	Séquences des autres vertébrés
5.	INV	Séquences des invertébrés
6.	PLN	Séquences des plantes, des champignons et des algues
7.	BCT	Séquences des bactéries
8.	VRL	Séquences des virus
9.	PHG	Séquences des bactériophages
10.	SYN	Séquences des synthétiques
11.	UNA	Séquences non-annotées
12.	EST	Séquences des EST
13.	PAT	Séquences brevetées
14.	STS	Séquences des STS
15.	GSS	Séquences d'enquête de génome
16.	HTG	Séquences génomiques de haut débit
17.	HTC	Séquences cDNA à haut débit
18.	ENV	Séquences environnementales

Tableau 1 : Taxonomie de Genbank

La classification taxonomique basée sur les séquences de la base de données sont classées et peuvent être recherchées grâce à une taxonomie basée sur les séquences exhaustives (7), élaborée en collaboration entre le NCBI, EMBL-Bank et DDBJ, avec la contribution précieuse de conseillers externes et de conservateurs (8).

Exemple d'entrée de genbank (gène *Synechococcus elongatus*)

Synechococcus elongatus PCC 7942 genes for intrinsic membrane protein, malK-like protein, cyanase, complete cds

GenBank: AB000100.1

[FASTA](#) [Graphics](#)

[Go to:](#) ☒

LOCUS AB000100 2992 bp DNA linear BCT 15-MAY-2009
DEFINITION *Synechococcus elongatus* PCC 7942 genes for intrinsic membrane protein, malK-like protein, cyanase, complete cds.
ACCESSION AB000100
VERSION AB000100.1
KEYWORDS .
SOURCE *Synechococcus elongatus* PCC 7942 = FACHB-805
ORGANISM [Synechococcus elongatus PCC 7942 = FACHB-805](#)
Bacteria; Cyanobacteriota; Cyanophyceae; Synechococcales; Synechococcaceae; *Synechococcus*.
REFERENCE 1
AUTHORS Harano,Y., Suzuki,I., Maeda,S., Kaneko,T., Tabata,S. and Omata,T.
TITLE Identification and nitrogen regulation of the cyanase gene from the cyanobacteria *Synechocystis* sp. strain PCC 6803 and *Synechococcus* sp. strain PCC 7942
JOURNAL J. Bacteriol. 179 (18), 5744-5750 (1997)
PUBMED [9294430](#)
REFERENCE 2 (bases 1 to 2992)
AUTHORS Omata,T.
TITLE Direct Submission
JOURNAL Submitted (26-DEC-1996) Contact:Tatsuo Omata School of Agricultural Sciences, Nagoya University, Department of Applied Biological Sciences; Chikusa, Nagoya, Aichi 464-01, Japan
COMMENT On Aug 16, 1997 this sequence version replaced gi:[1943948](#).

Figure 1 : Une entrée genbank du gène *Synechococcus elongatus* PCC (1), (9).

C. Description de quelques champs de genbank (18)

- **LOCUS** : C'est un mot clé qui décrit bien la définition de la séquence.
- **DEFINITION** : c'est un résumé concis de la séquence qui commence par une vue d'ensemble et se détaille progressivement.
- **NUMERO ACCESSION** : Le numéro d'accès principal est unique et immuable, présentant deux formats : 6 caractères ou 8 caractères, appelés "numéros d'accès".
- **VERSION** : La version est un code constitué d'un numéro d'accès et d'un numéro de version, lié à la séquence actuelle dans l'enregistrement. Il est suivi d'une clé entière (GI) attribuée par le NCBI.
- **KEYWORDS** : Ce sont des courtes phrases décrivant les produits génétiques et fournissant d'autres détails relatifs à l'enregistrement.
- **SOURCE** : Elle correspond au nom courant ou au terme le plus répandu pour désigner l'organisme, tel qu'utilisé fréquemment dans la littérature.

ORGANISM : Cet élément comprend le nom scientifique et officiel de l'organisme, ainsi que les niveaux de classification taxonomique.

- **REFERENCE** : Ce descripteur englobe les références aux articles renfermant les données liées à cette entrée.

II. OBJECTIFS DE CE TRAVAIL

L'objectif de cette étude est de tenter de développer un système de stockage simplifié pour organiser et regrouper les données de genbank, facilitant ainsi leur exploitation. Pour cela, nous aborderons deux volets principaux :

Dans la première phase, nous nous concentrerons sur le nettoyage de l'arbre taxonomique élaboré par Monsieur David Coornaert. Nous assurerons la gestion des mises à jour de cet arbre tout en intégrant un mécanisme d'alerte pour signaler les modifications.

Dans la deuxième phase, nous répliquerons ce processus dans une base de données. Ensuite, nous procéderons à une comparaison approfondie des deux systèmes de stockage des données, afin d'identifier celui qui présente la meilleure adéquation à nos besoins.

III. MATERIEL ET METHODES

A. Mise en place de l'arbre taxonomique

Un arbre taxonomique, également connu sous le nom d'arbre phylogénétique, est une représentation graphique qui illustre les relations évolutives entre différentes espèces ou groupes d'organismes (10). Cet arbre hiérarchique montre comment les organismes sont liés par des ancêtres communs et comment ils ont divergé au fil du temps (10). Les branches de l'arbre symbolisent les liens évolutifs, tandis que les nœuds indiquent les points de divergence où de nouveaux groupes d'organismes ont émergé. L'arbre taxonomique est utilisé en biologie pour visualiser la classification et l'évolution des êtres vivants (10).

Le format GenBank est une structure complexe utilisée pour stocker à la fois des séquences génétiques et leurs annotations (11). Il se compose principalement d'un en-tête qui décrit l'organisme source, d'un corps contenant diverses annotations, et d'un bloc 'ORIGIN' contenant la séquence complète. Cependant, cette richesse d'informations rend sa manipulation assez difficile (11).

1. Analyse d'un fichier au format genbank

Présentation d'un extrait de fichier de taxonomie au format genbank

```
Fichier  Édition  Affichage  Recherche  Terminal  Aide
GBBCT1.SEQ      Genetic Sequence Data Bank
                  June 15 2023

NCBI-GenBank Flat File Release 256.0

Bacterial Sequences (Part 1)

102014 loci, 184372562 bases, from 102014 reported sequences

LOCUS      AB000100      2992 bp      DNA      linear      BCT 15-MAY-2009
DEFINITION Synchococcus elongatus PCC 7942 genes for intrinsic membrane
            protein, malK-like protein, cyanase, complete cds.
ACCESSION  AB000100
VERSION    AB000100.1
KEYWORDS
SOURCE     Synchococcus elongatus PCC 7942 = FACHB-805
ORGANISM   Synchococcus elongatus PCC 7942 = FACHB-805
            Bacteria; Cyanobacteria; Synchococcales; Synchococcaceae;
            Synchococcus.
REFERENCE  1
AUTHORS    Harano,Y., Suzuki,I., Maeda,S., Kaneko,T., Tabata,S. and Omata,T.
TITLE      Identification and nitrogen regulation of the cyanase gene from the
            cyanobacteria Synchocystis sp. strain PCC 6803 and Synchococcus
            sp. strain PCC 7942
JOURNAL    J. Bacteriol. 179 (18), 5744-5750 (1997)
PUBMED     9294430
REFERENCE  2 (bases 1 to 2992)
AUTHORS    Omata,T.
TITLE      Direct Submission
JOURNAL    Submitted (26-DEC-1996) Contact:Tatsuo Omata School of Agricultural
            Sciences, Nagoya University, Department of Applied Biological
            Sciences, Chikusa, Nagoya, Aichi 464-01, Japan
COMMENT    On Aug 16, 1997 this sequence version replaced gi:1943948.
FEATURES   Location/Qualifiers
            source
            1..2992
            /organism="Synchococcus elongatus PCC 7942 = FACHB-805"
```

Figure 2 : Un extrait du fichier gbbct1.seq au format genbank.

Le fichier gbbct1.seq fait 92 mégas et comprend 359844 lignes.

Tout fichier au format genbank se présente comme la figure ci-dessus (figure 2). Il y a beaucoup d'informations concernant l'organisme présenté.

Pour construire un arbre taxonomique, nous avons besoin de l'identifiant de l'organisme, ainsi que de toutes les familles auxquelles il appartient. Comment extraire ces familles ?

La figure 2 présente l'organisme ayant pour identifiant AB000100. Sur ce format, plusieurs informations sont fournies, notamment les champs LOCUS, DEFINITION, ACCESSION, VERSION, KEYWORDS...etc.

Les champs qui nous intéressent sont principalement la ligne '**LOCUS**' car elle contient l'identifiant de l'organisme ; toutes les informations comprises entre les lignes '**ORGANISM**' ET '**REFERENCE**' qui contiennent les noms des familles pour situer chaque organisme dans l'arbre.

Au niveau de la ligne LOCUS, nous allons extraire l'identifiant **AB00100**. Entre les lignes ORGANISM et REFERENCE, nous allons extraire toutes les familles auxquelles AB000100 appartient notamment "**Bacteria; Cyanobacteria; Synechococcaceae; Synechococcus.**"

2. Quelques exemples d'organismes au format genbank

Utilisons un autre organisme pour illustrer tout ce qui a été dit plus haut.

```
//
LOCUS      AB000389                1508 bp    DNA        linear    BCT 27-OCT-2000
DEFINITION Pseudoalteromonas elyakovii gene for 16S rRNA.
ACCESSION  AB000389
VERSION    AB000389.1
KEYWORDS   .
SOURCE     Pseudoalteromonas elyakovii
  ORGANISM Pseudoalteromonas elyakovii
            Bacteria; Proteobacteria; Gammaproteobacteria; Alteromonadales;
            Pseudoalteromonadaceae; Pseudoalteromonas.
REFERENCE  1 (bases 1 to 1508)
```

```
LOCUS      AB000389                1508 bp    DNA        linear    BCT 27-OCT-2000
```

```
Bacteria; Proteobacteria; Gammaproteobacteria; Alteromonadales;
Pseudoalteromonadaceae; Pseudoalteromonas.
```

Figure 3 : Présentation de l'organisme AB000389 au format genbank, de la ligne du champs LOCUS et des lignes comprises entre les champs ORGANISM et REFERENCE.

Comment placer l'identifiant AB000389 dans un arbre taxonomique ?

Il suffit de nous focaliser sur la ligne LOCUS, et des lignes comprises entre ORGANISM et REFERENCE (figure 3).

3. Problème rencontré

Premier problème rencontré :

A première vue, on pourrait penser que c'est simple et facile, mais d'un point de vue programmatique, le problème est moins facile que ça en a l'air. Il y a des organismes qui ne respectent pas toujours cette structure, par exemple le nom de l'organisme peut être très long et aller sur plusieurs lignes, ceci représente un problème lors de l'extraction des lignes et parser devient compliqué.

Exemple : L'identifiant AB425276

Bifidobacterium catenulatum subsp. kashiwanohense gene for 16S ribosomal RNA, partial sequence, strain: HM2-2	
GenBank: AB425276.2	
FASTA Graphics	
Go to: <input type="checkbox"/>	
LOCUS	AB425276 1515 bp DNA linear BCT 17-OCT-2019
DEFINITION	Bifidobacterium catenulatum subsp. kashiwanohense gene for 16S ribosomal RNA, partial sequence, strain: HM2-2.
ACCESSION	AB425276
VERSION	AB425276.2
KEYWORDS	.
SOURCE	Bifidobacterium catenulatum subsp. kashiwanohense JCM 15439 = DSM 21854
ORGANISM	Bifidobacterium catenulatum subsp. kashiwanohense JCM 15439 = DSM 21854 Bacteria; Actinomycetota; Actinomycetes; Bifidobacteriales; Bifidobacteriaceae; Bifidobacterium.
REFERENCE	1

ORGANISM	Bifidobacterium catenulatum subsp. kashiwanohense JCM 15439 = DSM 21854 Bacteria; Actinobacteria; Bifidobacteriales; Bifidobacteriaceae; Bifidobacterium.
REFERENCE	1

Figure 4 : Présentation de l'organisme AB425276 au format genbank.

Sur la figure 4, on peut remarquer que le champs 'ORGANISM' va sur deux lignes. Alors, si on suit la logique selon laquelle on s'intéresse aux lignes comprises entre les champs ORGANISM et REFERENCE, ça ne marchera pas car dans ce cas 21854 sera considéré comme étant le nom d'une famille et ce n'est pas le cas. De ce fait, ça va être compliqué de parser.

4. Comment fait biopython pour parser ?

Biopython possède un parser pour les fichiers au format genbank , essayons d'extraire la taxonomie de l'organisme AB425276 et voyons s'il détecte un problème.

```
(bioP) marie@marieint:/data$ python3
Python 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from Bio import SeqIO
>>> for seq in SeqIO.parse("ab425276.entret","genbank"):
...     print(seq)
...
ID: AB425276.2
Name: AB425276
Description: Bifidobacterium catenulatum subsp. kashiwanohense gene for 16S ribosomal RNA, partial sequence, strain: HM2-2
Number of features: 2
/molecule_type=DNA
/topology=linear
/data_file_division=BCT
/date=17-OCT-2019
/accessions=['AB425276']
/sequence_version=2
/keywords=[]
/source=Bifidobacterium catenulatum subsp. kashiwanohense JCM 15439 = DSM 21854
/organism=Bifidobacterium catenulatum subsp. kashiwanohense JCM 15439 = DSM 21854
/taxonomy=['Bacteria', 'Actinobacteria', 'Bifidobacteriales', 'Bifidobacteriaceae', 'Bifidobacterium']
/references=[Reference(title='Bifidobacterium kashiwanohense sp. nov., isolated from healthy infant faeces', ...), Reference(title='Direct Submission', ...)]
/comment=On Aug 23, 2010 this sequence version replaced AB425276.1.
Seq('AGGGTTCGATTCTGGCTCAGGATGAACGCTGGCGGCTGCTTAACACATGCAAG...CCT')
>>> seq.annotations['taxonomy']
['Bacteria', 'Actinobacteria', 'Bifidobacteriales', 'Bifidobacteriaceae', 'Bifidobacterium']
>>>
```

Figure 5 : Taxonomie de l'organisme AB425276 réalisée par biopython.

Biopython parvient à extraire la taxonomie des organismes, même dans cette situation.

Comment le fait-il ?

```
elif line type == "ORGANISM":
    # Typically the first line is the organism, and subsequent lines
    # are the taxonomy lineage. However, given longer and longer
    # species names (as more and more strains and sub strains get
    # sequenced) the organism name can now get wrapped onto multiple
    # lines. The NCBI say we have to recognise the lineage line by
    # the presence of semi-colon delimited entries. In the long term,
    # they are considering adding a new keyword (e.g. LINEAGE).
    # See Bug 2591 for details.
```

Figure 6 : Explication du mécanisme utilisé par biopython pour parser.

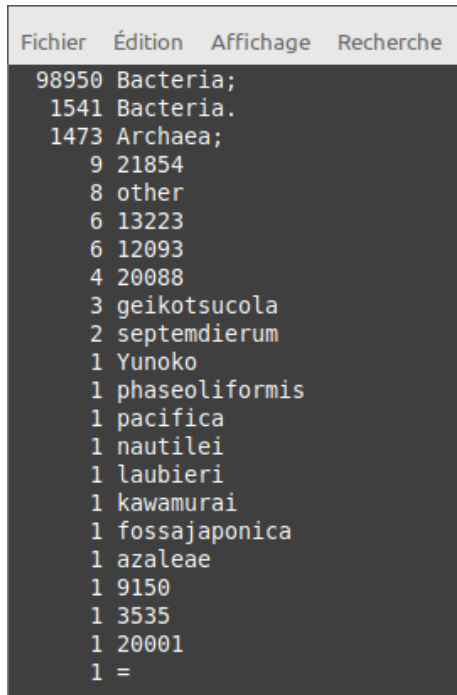
Dans le code source de biopython, il est mentionné que pour déterminer la taxonomie d'un organisme à partir de genbank, la ligne à parser est celle qui suit directement le ligne ORGANISM et le nom d'une famille qui constituera l'arborescence est toujours suivie d'un point-virgule “;” (**Figure 6**). Que faire lorsqu'un nom n'est pas suivi de point-virgule, ne fait-il donc pas partie de la taxonomie ?

Cette réponse de biopython soulève un deuxième problème.

Commande utilisée :

```
zcat gbbct1.seq.gz|sed -n '/^ ORGANISM/,/^REFERENCE/ p'|grep '^ ORGANISM' -A1|grep -v '^ ORGANISM'|grep -v '^--'|sed -e's/^\s|+//'|awk '{print $1}'|sort|uniq -c|sort -nr|less
```

Cette commande a pour rôle de parcourir le fichier gbbct1.seq.bz2 qui contient 100 000 entrées, choisir uniquement les lignes comprises entre ORG et REF, extraire les premiers éléments, les regrouper, les compter et les classer par ordre décroissant.



```
98950 Bacteria;
1541 Bacteria.
1473 Archaea;
9 21854
8 other
6 13223
6 12093
4 20088
3 geikotsucola
2 septemdierum
1 Yunoko
1 phaseoliiformis
1 pacifica
1 nautili
1 laubieri
1 kawamurai
1 fossajaponica
1 azaleae
1 9150
1 3535
1 20001
1 =
```

Figure 7 : Présentation des premiers éléments des lignes comprises entre ORGANISM et REFERENCE du fichier gbbct1.seq.bz2.

Que faire de tous ces éléments ? On voit qu’il ya des lignes avec **Bacteria**. Et d’après biopython il ne doit pas être considéré comme une famille, que faire de tous ces autres éléments qui apparaissent très peu de fois ?

Heterotrophic extracellular symbiont BG-C1 of Benthomodiolus geikotsucola gene for 16S ribosomal RNA, partial sequence

GenBank: AB679348.1

[FASTA](#) [Graphics](#)

[Go to:](#) ☐

LOCUS	AB679348	1462 bp	DNA	linear	BCT 05-DEC-2012
DEFINITION	Heterotrophic extracellular symbiont BG-C1 of Benthomodiolus geikotsucola gene for 16S ribosomal RNA, partial sequence.				
ACCESSION	AB679348				
VERSION	AB679348.1				
KEYWORDS	.				
SOURCE	heterotrophic extracellular symbiont BG-C1 of Benthomodiolus geikotsucola				
ORGANISM	heterotrophic extracellular symbiont BG-C1 of Benthomodiolus geikotsucola Bacteria.				
REFERENCE	1				

Figure 8 : Présentation de l’organisme AB679348 qui illustre “**Bacteria**.”.

```

>>> for seq in SeqIO.parse("ab679348.entret", "genbank"):
...     print(seq)
...
ID: AB679348.1
Name: AB679348
Description: Heterotrophic extracellular symbiont BG-C1 of Benthomodiolus geikotsucola gene for 16S ribosomal RNA, partial sequence
Number of features: 2
/molecule_type=DNA
/topology=linear
/data file division=BCT
/date=05-DEC-2012
/accessions=['AB679348']
/sequence_version=1
/keywords=[]
/source=heterotrophic extracellular symbiont BG-C1 of Benthomodiolus geikotsucola
/organism=heterotrophic extracellular symbiont BG-C1 of Benthomodiolus geikotsucola Bacteria.
/taxonomy=[]
/references=[Reference(title='Symbiosis and evolution in the whale-fall mussel Benthomodiolus geikotsucola', ...), Reference(title='Direct Submission', ...)]
Seq('ATTGAACGCTGGCGGCATGCGCTAACACATGCAAGTCGAACGGAAACGATGCTAG...GTG')
>>> seq.annotations['taxonomy']
[]
>>>

```

Figure 9 : Taxonomie de l'organisme AB679348 réalisée par biopython.

Biopython ne trouve pas la taxonomie de cet organisme (figure 9).

Grâce à ces contre-exemples (figure 5 et 9), on ne peut pas entièrement se fier à ce qui est mentionné dans le code source de biopython :

- **“La ligne à parser est celle qui suit la ligne ‘organism’”** : car on remarque qu’il y a des éléments qui suivent directement la ligne ORGANISM mais ne font pas partie du lignage de la taxonomie. (9 fois 21854, une fois 9150, =, 8 fois other...), figure 9.
- **“On ne peut pas non plus se fier aux “;”, mentionné par biopython**, car cet exemple nous montre que le nom d’une famille n’est pas toujours suivi de point-virgule. (1541 fois Bacteria.)

Comment Monsieur Coornaert fait-il pour contourner ce problème ?

5. Solution apportée par Monsieur David Coornaert

Pour contourner ce problème, Monsieur Coornaert a écrit un programme qui tourne depuis des années (**parseorginline.py**).

Ce programme lit les lignes comprises entre ORGANISM et REFERENCE, il cherche une ligne qui commence par un des mots corrects : (**Archaea / Bacteria / Cellular_organisms / Eukaryota / Other_sequences / Unclassified / Unclassified_sequences / Viruses**). Tant que ce n'est pas l'un de ces mots le programme continue de lire sans toutefois extraire quoique ce soit, lorsqu'il rencontre un de ces mots, il extrait le nom.

Cet arbre est stocké dans le dossier **/data/org**.

```
marie@mariemint:/data/org$ tree -d -L 1 /data/org/  
/data/org/  
├── Archaea  
├── Bacteria  
├── cellular_organisms  
├── Eukaryota  
├── other_sequences  
├── Unclassified  
├── unclassified_sequences  
└── Viruses
```

Figure 10 : Les familles principales de l'arbre taxonomique.

```
├── Archaea  
│   ├── Asgard_group  
│   │   ├── Candidatus_Heimdallarchaeota  
│   │   │   ├── Candidatus_Heimdallarchaeum  
│   │   │   │   ├── gb  
│   │   │   │   └── gb  
│   │   ├── Candidatus_Lokiarchaeota  
│   │   │   ├── Candidatus_Lokiarchaeum  
│   │   │   │   ├── gb  
│   │   │   ├── Candidatus_Prometheoarchaeum  
│   │   │   │   ├── gb  
│   │   │   └── gb  
│   │   ├── Candidatus_Odinarchaeota  
│   │   │   ├── Candidatus_Odinarchaeum  
│   │   │   │   ├── gb  
│   │   └── Candidatus_Thorarchaeota  
│   │   │   └── gb  
│   ├── Candidatus_Aenigmarchaeota  
│   │   ├── Candidatus_Aenigmarchaeum  
│   │   │   ├── environmental_samples  
│   │   │   │   ├── gb  
│   │   └── environmental_samples  
│   │   │   ├── gb  
│   │   └── gb  
│   ├── Candidatus_Altiarchaeota  
│   │   ├── Candidatus_Altiarchaeales  
│   │   │   └── gb  
│   └── Candidatus_Bathyarchaeia  
│       ├── environmental_samples  
│       │   ├── gb  
│       └── gb
```

Figure 11 : Présentation graphique d'un extrait de l'arbre (famille des Archaea).

6. Description de l'arbre

L'arbre taxonomique /data/org est constitué de grandes familles principales qui constituent la profondeur 1 de l'arbre : **Archaea / Bacteria / Cellular_organisms / Eukaryota / Other_sequences / Unclassified / Unclassified_sequences / Viruses**.

Chaque grande famille est à son tour constituée de sous familles (profondeur 2) ainsi que d'un fichier gb qui contient tous les identifiants des sous familles présentes dans le dossier.

Chaque sous-famille (profondeur 2) est à son tour constituée de familles (profondeur 3) et d'un fichier gb qui reprend le nom de tous les identifiants des organismes appartenant à cette branche.

Sur la figure ci-dessous (figure 10), on peut voir toutes les familles constituant le niveau 2 de la grande famille Viruses, ainsi que le fichier gb.

```
marie@marieint:/data/org$ cd Viruses/
marie@marieint:/data/org/Viruses$ ls -l
total 420
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Adenoviridae
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Adnaviria
drwxrwxr-x 2 marie marie 4096 jun 19 15:20 Adomaviridae
drwxrwxr-x 5 marie marie 4096 jun 20 14:44 Alphasatellitidae
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Ampullaviridae
drwxrwxr-x 29 marie marie 4096 jun 20 10:41 Anelloviridae
drwxrwxr-x 5 marie marie 4096 jun 19 15:20 Avsunviroidae
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Bicaudaviridae
drwxrwxr-x 4 marie marie 4096 jun 19 15:20 Caudovirales
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Circoviridae
drwxrwxr-x 2 marie marie 4096 jun 19 15:20 Circularisvirus
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Clavaviridae
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Cruciviridae
drwxrwxr-x 2 marie marie 4096 jun 19 15:20 Deltavirus
drwxrwxr-x 2 marie marie 4096 jun 19 15:20 Dinodnavirus
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 DNA_viruses
drwxrwxr-x 7 marie marie 4096 jun 19 15:20 dsDNA_viruses,_no_RNA_stage
drwxrwxr-x 4 marie marie 4096 jun 19 15:20 dsRNA_viruses
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Duplodnaviria
drwxrwxr-x 2 marie marie 4096 jun 20 15:22 environmental_samples
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Finnlakeviridae
drwxrwxr-x 4 marie marie 4096 jun 19 15:20 Fusesloviridae
-rw-rw-r-- 1 marie marie 198886 jui 6 01:12 gb
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Globuloviridae
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Guttaviridae
drwxrwxr-x 2 marie marie 4096 jun 19 15:20 Haloviruses
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Halspiviridae
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Herpesvirales
drwxrwxr-x 2 marie marie 4096 jun 19 15:20 Kirkovirus_group
drwxrwxr-x 2 marie marie 4096 jun 19 15:20 Marine_virus_AFVG
drwxrwxr-x 2 marie marie 4096 jun 19 15:20 Miresoil_virus_gcode6_group
drwxrwxr-x 6 marie marie 4096 jun 19 15:20 Monodnaviria
drwxrwxr-x 4 marie marie 4096 jun 19 15:20 Naldaviricetes
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Ortervirales
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Ovaliviridae
drwxrwxr-x 2 marie marie 4096 jun 19 15:20 Pandoravirus
drwxrwxr-x 3 marie marie 4096 jun 22 14:00 Pithoviridae
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Plasmaviridae
drwxrwxr-x 4 marie marie 4096 jun 19 15:20 Polydnaviriformidae
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Portogloboviridae
drwxrwxr-x 7 marie marie 4096 jun 19 15:20 Pospiviroidae
drwxrwxr-x 4 marie marie 4096 jun 19 15:20 Retro-transcribing_viruses
drwxrwxr-x 23 marie marie 4096 jui 11 23:15 Riboviria
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Ribozviria
drwxrwxr-x 4 marie marie 4096 jun 19 15:20 Satellites
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 Spiraviridae
drwxrwxr-x 4 marie marie 4096 jun 19 15:20 ssDNA_viruses
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 ssRNA_negative-strand_viruses
drwxrwxr-x 3 marie marie 4096 jun 19 15:20 ssRNA_positive-strand_viruses,_no_DNA_stage
drwxrwxr-x 4 marie marie 4096 jun 19 15:20 ssRNA_viruses
drwxrwxr-x 2 marie marie 4096 jun 19 15:20 Statovirus
drwxrwxr-x 4 marie marie 4096 jun 19 15:20 Tolecusatellitidae
drwxrwxr-x 2 marie marie 4096 jun 19 15:20 unclassified_archaeal_viruses
drwxrwxr-x 2 marie marie 4096 jui 3 10:44 unclassified_bacterial_viruses
drwxrwxr-x 4 marie marie 4096 jun 19 15:20 unclassified_viruses
drwxrwxr-x 4 marie marie 4096 jui 2 12:56 Varidnaviria
drwxrwxr-x 2 marie marie 4096 jun 19 15:20 Volvovirus
marie@marieint:/data/org/Viruses$
```

Figure 12 : Présentation du premier niveau de la famille des Viruses.

Présentation graphique :

```
marie@mariehint:/data/org/Viruses$ tree /data/org/Viruses/ | head -50
/data/org/Viruses/
├── Adenoviridae
│   ├── Mastadenovirus
│   │   └── Human mastadenovirus_B
│   │       └── gb
│   └── Adnaviria
│       └── Zilligvirae
│           └── Taleaviricota
│               └── Tokiviricetes
│                   └── Ligamenvirales
│                       ├── Lipothrixviridae
│                       │   ├── Alphalipothrixvirus
│                       │   │   ├── Alphalipothrixvirus_SBFV2
│                       │   │   │   └── gb
│                       │   │   └── Alphalipothrixvirus_SFV1
│                       │   │       └── gb
│                       │   └── Betalipothrixvirus
│                       │       └── gb
│                       │   └── Deltalipothrixvirus
│                       │       ├── Deltalipothrixvirus_SBFV3
│                       │       │   └── gb
│                       │       └── gb
│                       └── Rudiviridae
│                           ├── Azorudivirus
│                           │   └── Azorudivirus_SRV
│                           │       └── gb
│                           ├── Hoswirudivirus
│                           │   ├── Hoswirudivirus_ARV2
│                           │   │   └── gb
│                           │   ├── Hoswirudivirus_ARV3
│                           │   │   └── gb
│                           │   ├── Hoswirudivirus_MRV1
│                           │   │   └── gb
│                           │   └── Hoswirudivirus_SSRV1
│                           │       └── gb
│                           ├── Icerudivirus
│                           │   ├── Icerudivirus_SIRV1
│                           │   │   └── gb
│                           │   └── Icerudivirus_SIRV2
│                           │       └── gb
│                           ├── Itarudivirus
│                           │   └── Itarudivirus_ARV1
│                           │       └── gb
│                           ├── Japarudivirus
│                           │   └── Japarudivirus_SBRV1
│                           │       └── gb
│                           ├── Rudivirus
│                           │   └── gb
│                           └── Usarudivirus
│                               └── gb
└── marie@mariehint:/data/org/Viruses$
```

```
marie@mariehint:/data/org/Viruses$ cat gb | head -30
A22359
A22374
A22375
A32184
A32185
A32186
A32199
A32200
A32201
A32202
A32203
A59246
A81401
AB004561
AB015437
AB048798
AB161975
AB177605
AB182649
AB251919
AB331966
AB331967
AB331968
AB331969
AB331970
AB331971
AB331972
AB331973
AB331974
AB331975
marie@mariehint:/data/org/Viruses$
```

Figure 13 : Présentation graphique de la famille des “Viruses” et contenu d’un fichier gb.

7. Nettoyage de l'arbre taxonomique et mise à jour des fichiers

Je travaille sur les fichiers daily-nc du NCBI (12), notamment les fichiers nc*.flat.gz et leurs contigs con_nc.*.flat.gz comme illustré sur la figure 14.

Index of /genbank/daily-nc		
Name	Last modified	Size
Parent Directory		-
giu_at_gbrel_cod/	2023-06-22 09:20	-
tls/	2023-07-25 06:42	-
tsa/	2023-08-04 07:50	-
wgs/	2023-08-04 07:50	-
Last File	2023-08-04 01:52	15
README.genbank.daily-nc	2023-06-21 21:24	2.8K
con_nc.0616.flat.gz	2023-06-16 07:55	46K
con_nc.0617.flat.gz	2023-06-17 01:47	2.6K
con_nc.0621.flat.gz	2023-06-21 02:09	14K
con_nc.0624.flat.gz	2023-06-24 02:15	6.4K
con_nc.0627.flat.gz	2023-06-27 02:03	2.1K
con_nc.0628.flat.gz	2023-06-28 02:12	7.8K
nc0616.fsa.gz	2023-06-16 07:53	7.8M
nc0616.fsa_nt.gz	2023-06-16 07:53	1.0G
nc0616.gnp.gz	2023-06-16 07:53	15M
nc0617.flat.gz	2023-06-17 01:47	649M
nc0617.fsa.gz	2023-06-17 01:47	924K
nc0617.fsa_nt.gz	2023-06-17 01:47	465M
nc0617.gnp.gz	2023-06-17 01:47	2.2M
nc0618.flat.gz	2023-06-18 01:49	426M
nc0618.fsa.gz	2023-06-18 01:49	17M
nc0618.fsa_nt.gz	2023-06-18 01:49	255M

Figure 14 : Les fichiers de genbank utilisés pour ce projet.

Comment se fait la mise à jour de chaque fichier ?

Chaque jour, je prends le fichier de la veille, je nettoie et je le mets dans l'arbre. L'ordre des fichiers doit être respecté. Par exemple : le fichier nc0617.flat.gz, ensuite son contig con_nc.0617.flat.gz, le jour d'après je fais le nc0618.flat.gz et son contig s'il en a.

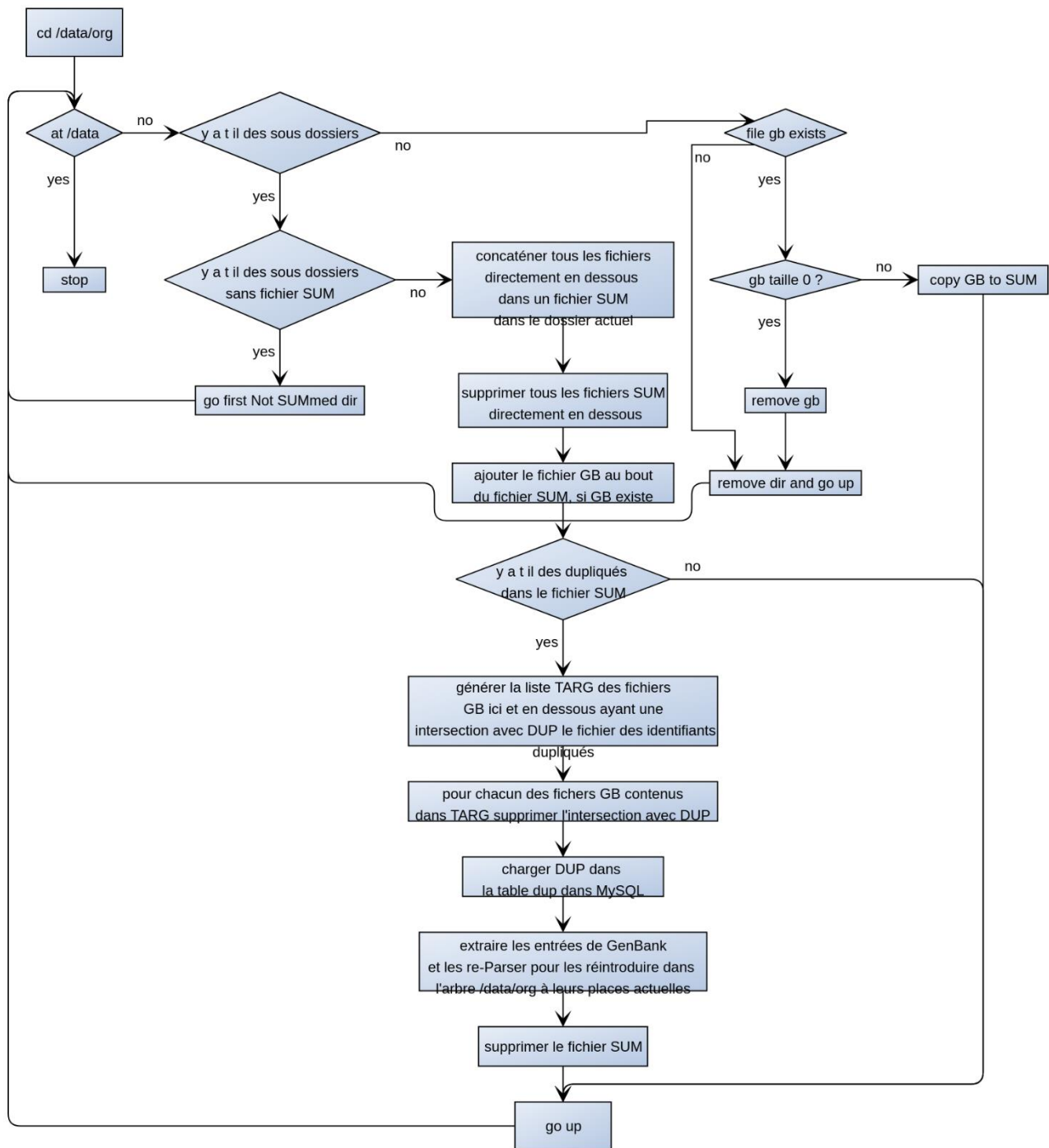


Figure 15 : Logigramme établi pour nettoyer et mettre l'arborescence à jour.

Les différentes étapes respectées pour nettoyer et mettre chaque fichier à jour :

→ Téléchargement et extraction de l'arbre taxonomique data/org de BIG pour ma machine :

wget big/org.tar.xz ensuite **tar xvf org.tar.xz**

→ Téléchargement du fichier à nettoyer (par exemple nc0517.flat.gz) :

wget https://ftp.ncbi.nlm.nih.gov/genbank/daily-nc/nc0517.flat.gz

→ Mise du fichier d'update dans l'arbre :

zcat nc0517.flat.gz | sed -n -f prune.sed | python3 parseorginline.py

→ Copie de tous les identifiants de l'arbre dans le fichier sum :

find . -name 'gb' -exec cat {} \; > sum

→ Trie de ces identifiants que je réécris dans sum :

sort sum -o sum

→ Extraction du fichier sum des identifiants qui se répètent que j'écris dans le fichier dup :

uniq -d sum > dup

→ Identification des fichiers dans lesquels ces identifiants apparaissent que je copie dans le fichier targ

La première fois j'utilisais ces deux commandes :

Sed -i -e 's/^/^/' -e 's/\$/\$/' dup

Grep -Hl -f dup -r --include='gb'

Ensuite, j'ai découvert cette option '-x' qui a permis de supprimer la première commande, on se retrouve avec une seule commande :

grep -Hl -r -x -f dup --include='gb' > targ

→ Tous les identifiants dupliqués sont supprimés des fichiers targ

sh delete_dup.sh

→ Ces identifiants sont ensuite remplacés convenablement à leur place

```
zcat nc0517.flat.gz | sed -n -f prune.sed | python3 parseorginline.py
```

Suite à ce nettoyage, il peut y arriver que certains fichiers gb deviennent vides, par conséquent certains dossiers peuvent également se retrouver vides ; il faut donc les supprimer.

→ Suppression des fichiers 'gb' vides de l'arbre :

```
find /data/org -type f -name "gb" -size 0 -delete -print
```

→ Suppression des dossiers vides de l'arbre :

```
find /data/org -type d -empty -delete -print
```

→ Suppression des identifiants que contient le fichier **gbdel.txt.gz**

```
marie@marieint:/data/org$ zcat gbdel.txt.gz | head -20
BCT165|CP015538
BCT165|CP015539
BCT165|CP015540
BCT165|CP015541
BCT165|CP015542
BCT165|CP015543
BCT165|CP015544
BCT165|CP015545
BCT165|CP015546
BCT165|CP015547
BCT165|CP015548
BCT165|CP015549
```

Figure 16 : Affichage du contenu de gbdel.txt.gz.

J'extrais uniquement la 2e colonne, et je fais le même traitement que le fichier dup :

```
marie@marieint:/data/org$ cat dup | head -20
CP015538
CP015539
CP015540
CP015541
CP015542
CP015543
CP015544
CP015545
CP015546
CP015547
CP015548
CP015549
CP015550
CP015555
CP015556
CP040012
CP040013
CP040014
CP061872
CP061873
marie@marieint:/data/org$ grep -Hl -r -x -f dup --include='gb' > targ
marie@marieint:/data/org$ cat targ | wc -l
43
marie@marieint:/data/org$ cat dup | wc -l
337
marie@marieint:/data/org$ cat delete_dup.
delete_dup.png delete_dup.sh
marie@marieint:/data/org$ cat delete_dup.sh
for f in $(cat targ)
do
    echo cleaning $f
    cat dup $f | sort | uniq -u > zzz
    mv zzz $f
done
marie@marieint:/data/org$
```

Figure 17 : Traitement du fichier gbdel.txt.gz.

8. Ajout d'un système d'alerte dans l'arborescence

Le script utilisé est `alerte.py`. L'objectif de ce script sera d'aviser un utilisateur par son mail renseigné en cas de mise à jour dans la famille qui l'intéresse.

```
marie@mariemint:/data/org/Bacteria$ python3 alerte.py
Sending report to maricatongang@gmail.com
mail envoyé avec succès
marie@mariemint:/data/org/Bacteria$
```

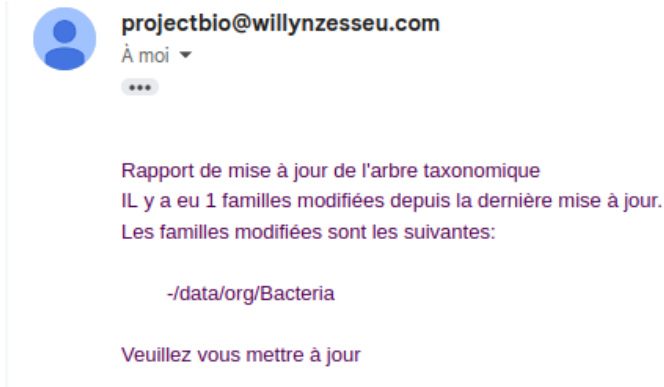


Figure 18 : Rapport envoyé par mail en cas de modification de la famille qui nous intéresse.

Pour mettre en place ce système d'alerte, je vais dans un dossier (famille) de l'arbre qui m'intéresse. Dans ce dossier, je crée un fichier drapeau qui constitue l'adresse électronique à laquelle un rapport sera envoyé. A partir de ce fichier drapeau, le script `alerte.py` pourra retrouver la famille à surveiller.

Lorsqu'on exécute du script `alerte.py`, l'utilisateur reçoit un mail comme indiqué plus haut (figure 18).

```

1  import os
2  import re
3  import smtplib
4  from email.mime.text import MIMEText
5  from email.mime.multipart import MIMEMultipart
6
7  def send_mail(email, rapport):
8      sender = "projectbio@willynzesseu.com"
9      password = "!~P^D=LaISAH"
10     message = MIMEMultipart()
11     message["From"] = sender
12     message["To"] = email
13     message["Subject"] = "Rapport de l'arbre taxonomique"
14     message.attach(MIMEText(rapport, "plain"))
15     try:
16         with smtplib.SMTP_SSL("mail.willynzesseu.com", 465) as server:
17             server.login(sender, password)
18             server.send_message(message)
19             print("mail envoyé avec succès")
20     except smtplib.SMTPException as ex:
21         print("Erreur d'envoi du mail:", str(ex))
22         pass
23
24
25     flag_files = {}
26     flag_regex = re.compile('(flag_).*')
27     gb_regex = re.compile('gb')
28
29     for root, dirs, files in os.walk("/data/org"):
30         for file in files:
31             if flag_regex.match(file):
32                 fullFileName = os.path.join(root, file)
33                 flag_files[fullFileName] = os.stat(fullFileName).st_mtime
34
35
36     for flag_file in flag_files:
37         baseDir = os.path.dirname(flag_file)
38         fileName = os.path.basename(flag_file)
39         date = flag_files[flag_file]
40         email = fileName.split('_')[-1]
41
42         count = 0
43         gbFiles = []
44
45         for root, dirs, files in os.walk(baseDir):
46             for file in files:
47                 if gb_regex.match(file):
48                     gbFile = os.path.join(root, file)
49                     if os.stat(gbFile).st_mtime > date:
50                         gbFiles.append(gbFile)
51                         count += 1
52
53         if count > 0:
54             rapport = ""
55             rapport += "Rapport de mise à jour de l'arbre taxonomique"
56             rapport += "IL y a eu {0} familles modifiées depuis la dernière mise à jour."
57             rapport += "Les familles modifiées sont les suivantes:"
58             rapport += "".format(count)
59             for gbFile in gbFiles:
60                 famille = os.path.dirname(gbFile)
61                 rapport += " "
62                 rapport += "-{0} ".format(famille)
63                 rapport += " "
64             rapport += "Veuillez vous mettre à jour"
65             rapport += ""
66
67         print("Sending report to "+email)
68         send_mail(email, rapport)

```

Figure 19 : Script alerte.py

B. MISE EN PLACE DE LA BASE DE DONNEES

➤ Logiciel de gestion et d'administration de la base de données : DbVisualiZer free

Le programme utilisé pour la création de la base de données est **DbVisualizer Free (13)**. Il est conçu pour faciliter le développement, la gestion et la navigation dans diverses bases de données relationnelles ; il prend en charge un large éventail de bases de données relationnelles, notamment Oracle, MySQL, PostgreSQL, Microsoft SQL Server, SQLite, et bien d'autres (13). Cela vous permet de travailler avec différentes bases de données sans avoir à changer d'outil ; il permet également de visualiser graphiquement la structure des bases de données, y compris les relations entre les tables et les vues (13).

Pour le lancer en ligne de commande :

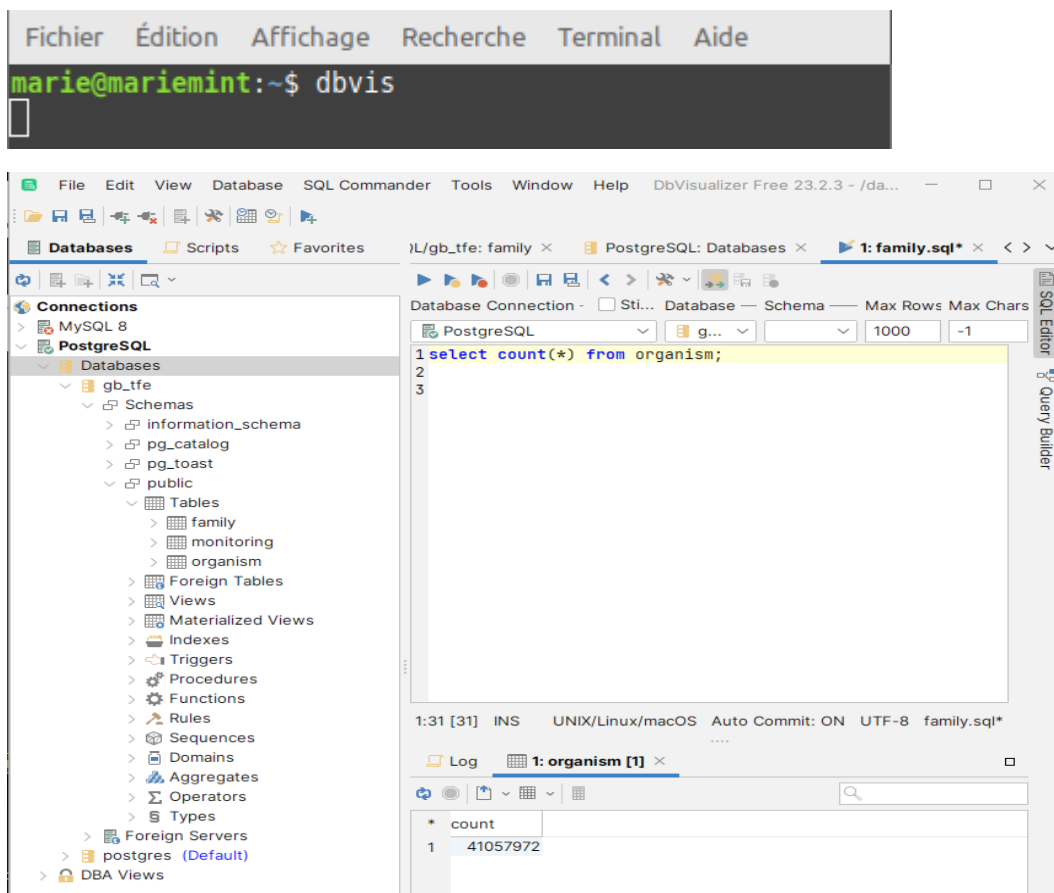


Figure 20 : Programme DbVisualiZer utilisé.

➤ **Le système de gestion de la base de données utilisé : POSTGRESQL**

PostgreSQL est un système de gestion de base de données relationnelle (SGBDR) open source et puissant (14). Il est conçu pour stocker et gérer de grandes quantités de données de manière efficace et sécurisée. PostgreSQL est réputé pour sa conformité aux normes SQL et sa capacité à gérer des charges de travail complexes (15),(16).

1. Stratégie de création des tables constituant la base de données

Pour assurer la qualité de la base de données que nous allons créer, différents paramètres sont pris en compte notamment :

→ La gestion des noms de famille identiques à des emplacements différents

- Éviter que les noms de famille identiques provenant de différents emplacements ne posent problème.

- Mettre en place une méthode de différenciation, comme l'utilisation d'identifiants uniques, pour distinguer les enregistrements similaires provenant de sources différentes.

→ Optimisation de la Structure des Tables pour les Requêtes SQL

- Concevoir la structure des tables de manière à offrir une variété maximale de choix de données lors de l'exécution de requêtes SQL.

- Organiser les données de manière logique et normalisée, en évitant les redondances inutiles.

En adoptant ces approches, vous pouvez créer une base de données qui évite les conflits liés aux noms de famille identiques provenant de différentes sources et qui offre une structure optimisée pour l'utilisation de requêtes SQL, favorisant ainsi une gestion et une analyse efficaces des données.

➤ **Provenance des données utilisées**

Les données employées pour remplir les tables de ma base de données proviennent du fichier le plus récent que j'ai mis à jour dans l'arborescence (**nc0706.flat.gz**). (12).

2. Création de la base de données et des tables

```
create database gb_tfe ;
```

La base de données gb_tfe est constituée de trois tables : **family, organism et monitoring**.

➤ **La table family**

```
create table family (id int primary key, family_name varchar (255), id_parent int) ;
```


La table 'family' est composée d'un attribut '**id**' qui assure une identification unique pour chaque famille dans la base de données. L'attribut '**family_name**' englobe toutes les familles présentes dans l'arbre généalogique, chaque famille étant associée à un identifiant unique. Étant donné la distinction entre les familles parentes et les familles enfants, il est essentiel de clairement définir les relations parent-enfant. L'attribut '**id_parent**' est utilisé pour déterminer explicitement le lien de parenté direct entre une famille et son parent ou enfant direct.

Le chargement des données dans la table family a pris **20 minutes**.

➤ La table organism

```
create table organism (id varchar (255), tree_path text, family_name varchar (255), date_update date) ;
```

Cette table englobe l'ensemble des identifiants issus d'un fichier. Chaque organisme appartient à une famille et une famille est définie par un chemin, d'où l'attribut '**tree_path**' qui représente le chemin pour retrouver chaque organisme ; L'appartenance à une famille est également caractérisée par le champ '**family_name**'. De plus, chaque identifiant dispose d'une date de dernière mise à jour, notée 'date_update'.

Le processus de chargement des données dans la table 'organism' s'est révélé chronophage. Après trois jours de chargement partiel, j'ai pris la décision d'interrompre ce processus et de travailler avec les données déjà intégrées, car mon espace de stockage était épuisé.

➤ La table monitoring

```
create table monitoring (email varchar (255), family_name varchar (255), family_id int, update_date date) ;
```

Cette table vise à habilitier un utilisateur à superviser une famille spécifique au sein d'un arbre généalogique. Par le biais de cette table, l'utilisateur aura la possibilité d'être notifié en cas de changements sur cette famille. La table "monitoring" est constituée des champs suivants :

- "email" : l'adresse électronique à laquelle les notifications de modifications seront envoyées.
- "family_name" : le nom de la famille qui suscite l'intérêt de l'utilisateur.
- "family_id" : l'identifiant unique associé à cette famille.
- "update_date" : la date de la dernière mise à jour effectuée.

En structurant les données de cette manière, les utilisateurs auront la capacité de surveiller et d'être informés des modifications relatives à une famille spécifique, facilitant ainsi leur interaction avec les données de l'arbre généalogique.

➤ Quelques requêtes SQL pour mieux comprendre la base de données gb_tfe

```
SELECT COUNT (*) FROM family ;
```

PostgreSQL g... 1000 -1

```
1 SELECT COUNT (*) FROM family ;
```

1:1 [1] INS UNIX/Linux/macOS Auto Commit: ON UTF-8 Untitled*

Log 1: family [1] x

	count
1	159717

Figure 21 : Requête qui compte le nombre de lignes de la table “family”.

La table family contient **159717** lignes qui correspondent aux noms des familles.

SELECT COUNT (*) FROM organism ;

PostgreSQL gb_tfe 1000 -1

```
1 SELECT COUNT (*) FROM organism ;
```

1:31 [31] INS UNIX/Linux/macOS Auto Commit: ON UTF-8 Untitled*

Log 1: organism [1] x

	count
1	94383844

Figure 22 : Requête qui compte le nombre de lignes de la table “organism”.

La table organism contient **94383839** lignes qui correspondent aux identifiants.

SELECT * FROM organism WHERE tree_path like 'Archaea>Asgard_group%' limit 10 ;

Database Connection PostgreSQL gb_tfe Schema Max Rows 1000 Max Cl -1

```
1 SELECT * FROM organism WHERE tree_path like 'Archaea>Asgard_group%' limit 10 ;
```

2

1:80 [80] INS UNIX/Linux/macOS Auto Commit: ON UTF-8 Untitled*

Log 1: organism [10] x

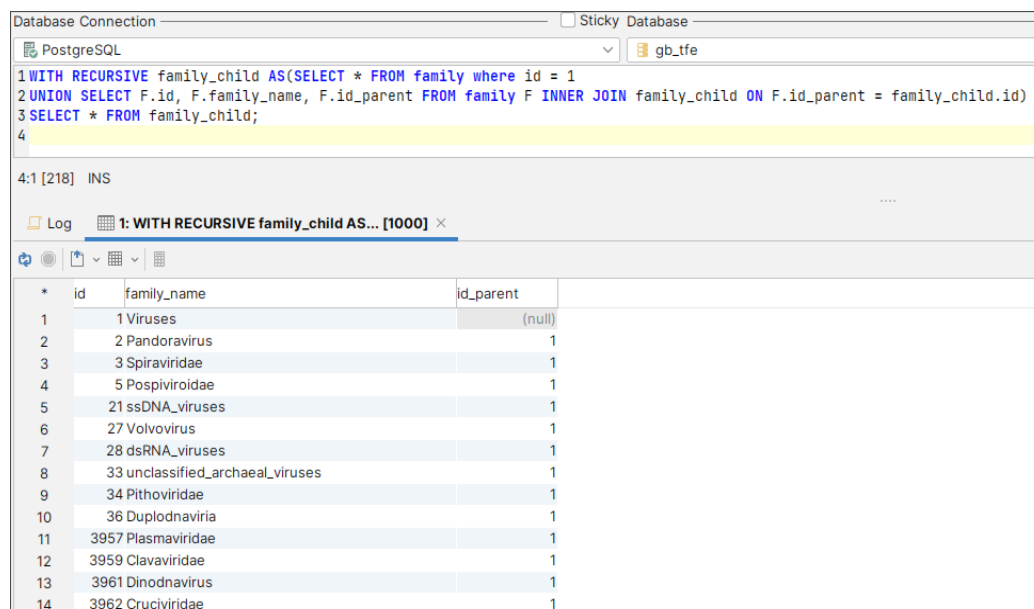
	id	tree_path	family_name	update_date
1	CP084166	Archaea>Asgard_group>Candidatus_Heimdallarchaeota	Candidatus_Heimdallarchaeota	2023-07-29
2	CP084167	Archaea>Asgard_group>Candidatus_Heimdallarchaeota	Candidatus_Heimdallarchaeota	2023-07-29
3	MG820600	Archaea>Asgard_group>Candidatus_Heimdallarchaeota	Candidatus_Heimdallarchaeota	2023-07-29
4	MG820608	Archaea>Asgard_group>Candidatus_Heimdallarchaeota	Candidatus_Heimdallarchaeota	2023-07-29
5	MG820609	Archaea>Asgard_group>Candidatus_Heimdallarchaeota	Candidatus_Heimdallarchaeota	2023-07-29
6	MG820610	Archaea>Asgard_group>Candidatus_Heimdallarchaeota	Candidatus_Heimdallarchaeota	2023-07-29
7	MK463862	Archaea>Asgard_group>Candidatus_Heimdallarchaeota	Candidatus_Heimdallarchaeota	2023-07-29
8	MW958324	Archaea>Asgard_group>Candidatus_Heimdallarchaeota	Candidatus_Heimdallarchaeota	2023-07-29
9	MW958325	Archaea>Asgard_group>Candidatus_Heimdallarchaeota	Candidatus_Heimdallarchaeota	2023-07-29
10	MW958326	Archaea>Asgard_group>Candidatus_Heimdallarchaeota	Candidatus_Heimdallarchaeota	2023-07-29

Figure 23 : Requête qui montre le détail des dix premiers identifiants de la table “organism” appartenant au chemin mentionné.

WITH RECURSIVE family_child AS(SELECT * FROM family where id = 1

UNION SELECT F.id, F.family_name, F.id_parent FROM family F INNER JOIN family_child ON F.id_parent = family_child.id)

SELECT * FROM family_child;



Database Connection: PostgreSQL | Sticky Database: gb_tfe

```

1 WITH RECURSIVE family_child AS(SELECT * FROM family where id = 1
2 UNION SELECT F.id, F.family_name, F.id_parent FROM family F INNER JOIN family_child ON F.id_parent = family_child.id)
3 SELECT * FROM family_child;
4

```

4:1 [218] INS

Log 1: WITH RECURSIVE family_child AS... [1000] X

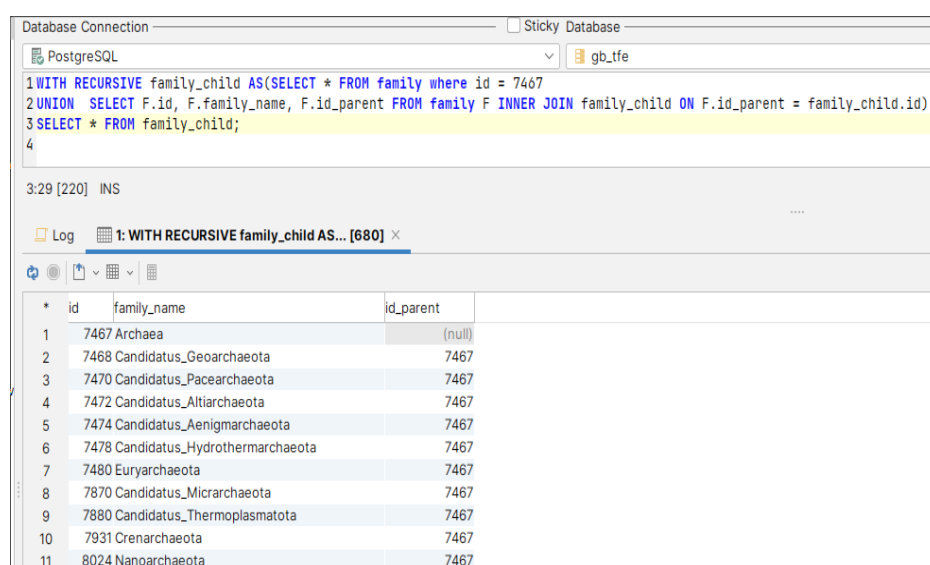
*	id	family_name	id_parent
1	1	Viruses	(null)
2	2	Pandoravirus	1
3	3	Spiraviridae	1
4	5	Pospiviroidae	1
5	21	ssDNA_viruses	1
6	27	Volvovirus	1
7	28	dsRNA_viruses	1
8	33	unclassified_archaea_viruses	1
9	34	Pithoviridae	1
10	36	Duplodnaviria	1
11	3957	Plasmaviridae	1
12	3959	Clavaviridae	1
13	3961	Dinodnavirus	1
14	3962	Cruciviridae	1

Figure 24 : Requête qui affiche tous les enfants directs de la famille des “Viruses”, en d’autres termes il s’agit des familles ayant pour id_parent = 1.

WITH RECURSIVE family_child AS(SELECT * FROM family where id = 7467

UNION SELECT F.id, F.family_name, F.id_parent FROM family F INNER JOIN family_child ON F.id_parent = family_child.id)

SELECT * FROM family_child;



Database Connection: PostgreSQL | Sticky Database: gb_tfe

```

1 WITH RECURSIVE family_child AS(SELECT * FROM family where id = 7467
2 UNION SELECT F.id, F.family_name, F.id_parent FROM family F INNER JOIN family_child ON F.id_parent = family_child.id)
3 SELECT * FROM family_child;
4

```

3:29 [220] INS

Log 1: WITH RECURSIVE family_child AS... [680] X

*	id	family_name	id_parent
1	7467	Archaea	(null)
2	7468	Candidatus_Geoarchaeota	7467
3	7470	Candidatus_Pacearchaeota	7467
4	7472	Candidatus_Altiarchaeota	7467
5	7474	Candidatus_Aenigmarchaeota	7467
6	7478	Candidatus_Hydrothermarchaeota	7467
7	7480	Euryarchaeota	7467
8	7870	Candidatus_Micrarchaeota	7467
9	7880	Candidatus_Thermoplasmata	7467
10	7931	Crenarchaeota	7467
11	8024	Nanoarchaeota	7467

Figure 25 : Requête qui affiche le nom de tous les enfants directs de la famille des “Archaea” ; en d’autres termes il s’agit des familles ayant pour id_parent = 7467.

3. La gestion des mises à jour dans la base de données

Dans cette partie, nous effectuerons la gestion des mises à jour des organismes dans la base de données.

Pour effectuer les mises à jour, la première étape consiste à parcourir le fichier de mise à jour "recup.txt", afin de récupérer l'identifiant de chaque entité ainsi que les noms des familles associées. Par la suite, il faudra exécuter soit une requête de mise à jour (UPDATE) ou une insertion (INSERT), en fonction de l'existence préalable de l'identifiant dans la base de données.

Face à cette situation, deux approches possibles me viennent à l'esprit :

Simulation :

Je vais débiter en modifiant la date de mise à jour de six organismes appartenant à la famille des "Archaea". L'intention est d'actualiser cette date pour qu'elle soit postérieure à la date précédente. L'objectif consiste à évaluer si ces trois modifications seront détectées dans la base de données.

La dernière date de mise à jour de ces organismes dans la base de données est : 2023-07-29.

Je modifie cette date en mettant une date plus récente. Ainsi, on pourrait penser que ces trois identifiants ont été mis à jour.

```
UPDATE organism set update_date = '2023-08-24' WHERE id in ('AF199372', 'AF477946', 'AF477947') ;
```

```
UPDATE organism set update_date = '2023-08-10' WHERE id in ('CP070665', 'CP070695', 'CP070760') ;
```

3 lignes ont été affectées.

1ère possibilité :

Pour commencer, je procède à la recherche de toutes les familles enfants de la famille qui suscite mon intérêt. Ensuite, j'effectue une recherche des identifiants ayant une date de mise à jour postérieure au 10 février 2023 et qui appartiennent à la famille des **Archaea**.

La famille des Archaea a pour identifiant **7467**. Je vérifie si la famille associée à chaque identifiant figure dans la liste des sous-familles préalablement identifiées.

Une fois cette étape accomplie, je suis en mesure de générer un rapport. Enfin, je mets à jour la table "monitoring" en attribuant la date du 11 février 2023.

```

1 SELECT * FROM organism WHERE update_date > '2023-07-29'
2 AND family_name IN (
3     WITH RECURSIVE family_child AS(
4         SELECT * FROM family WHERE id = 7467
5     UNION
6         SELECT F.id, F.family_name, F.id_parent FROM family F INNER JOIN family_child ON F.id_parent = family_child.id)
7     SELECT family_name FROM family_child);
8

```

8:1 [378] INS UNIX/Linux/macOS Auto Commi

Log 1: organism [6] X

* id	tree_path	family_name	update_date
1 AF199372	Archaea	Archaea	2023-08-24
2 AF477946	Archaea	Archaea	2023-08-24
3 AF477947	Archaea	Archaea	2023-08-24
4 CP070665	Archaea>Asgard_group>Candidatus_Heimdallarchaeota	Candidatus_Heimdallarchaeota	2023-08-10
5 CP070695	Archaea>Asgard_group>Candidatus_Heimdallarchaeota	Candidatus_Heimdallarchaeota	2023-08-10
6 CP070760	Archaea>Asgard_group>Candidatus_Heimdallarchaeota	Candidatus_Heimdallarchaeota	2023-08-10

Figure 26 : Requête qui affiche les organismes dont la date de mise à jour actuelle est plus récente que la dernière date de mise à jour (expérience 1).

2ème possibilité :

Je recherche dans la table “organism” tous les identifiants ayant une date de mise à jour supérieure à 2023-07-29 et dont le “family_path” contient le nom de la famille à surveiller (ayant pour identifiant 7467).

```

1 SELECT * FROM organism WHERE update_date > '2023-07-29'
2 AND tree_path LIKE ( SELECT CONCAT('%', family_name, '%') FROM family WHERE id = 7467);
3

```

3:1 [153] INS UNIX/Linux/macOS A

Log 1: organism [6] X

* id	tree_path	family_name	update_date
1 AF199372	Archaea	Archaea	2023-08-24
2 AF477946	Archaea	Archaea	2023-08-24
3 AF477947	Archaea	Archaea	2023-08-24
4 CP070665	Archaea>Asgard_group>Candidatus_Heimdallarchaeota	Candidatus_Heimdallarchaeota	2023-08-10
5 CP070695	Archaea>Asgard_group>Candidatus_Heimdallarchaeota	Candidatus_Heimdallarchaeota	2023-08-10
6 CP070760	Archaea>Asgard_group>Candidatus_Heimdallarchaeota	Candidatus_Heimdallarchaeota	2023-08-10

Figure 27 : Requête qui affiche les organismes dont la date de mise à jour actuelle est plus récente que la dernière date de mise à jour (expérience 2).

Pour ces deux expériences, nous retrouvons bien les organismes de départ.

Toutefois, la 2ème expérience est beaucoup plus simple mais il y a une grande possibilité d'avoir des erreurs, car ici, l'on ne tient pas compte de l'id des familles mais plutôt du nom ; or dans la base de données il y a des familles qui peuvent avoir le même nom mais dans des emplacements différents, ce qui pourrait rendre la recherche erronée.

➤ Mise à jour dans toute la base de données

Une simulation est faite en considérant que le fichier de mise à jour est **recup.txt**. Ceci signifie que le fichier recup.txt est toujours à jour à la date présente.

Code de mise à jour dans la base de données : organisms_update.py

```
from datetime import datetime
import psycopg2

map = {}
key = ""
file = open("/data/org/recup.txt", "r")

while True:
    line = file.readline()
    if not line: break

    if line.startswith("LOCUS"):
        organismKey = line.split()[1]
        treePath = str("")
        file.readline()
        while True:
            pathLine = file.readline()
            if pathLine.startswith("REFERENCE"):
                break
            else:
                treePath += pathLine.strip().replace(";", ">").replace(" ", "")
                #print(treePath)
        treePath = treePath[:-1]
        family = treePath.split(">")[-1]
        map[organismKey] = (family, treePath)
        #print("{} --> {} --> {}".format(organismKey, family, treePath))

date = datetime.datetime.now().strftime('%Y-%m-%d')

conn = psycopg2.connect(
    database="gb_tfe", user='postgres', password='marica1996', host='127.0.0.1', port= '5432'
)
cursor = conn.cursor()

for element in map:
    family = map[element][0]
    path = map[element][1]
    request = """INSERT INTO organism(id, tree_path, family_name, update_date) VALUES('{0}', '{1}', '{2}', '{3}') ON CONFLICT(id)
DO UPDATE SET tree_path = '{1}', family_name = '{2}', update_date = '{3}'""".format(element, path, family, date)

    cursor.execute(request)

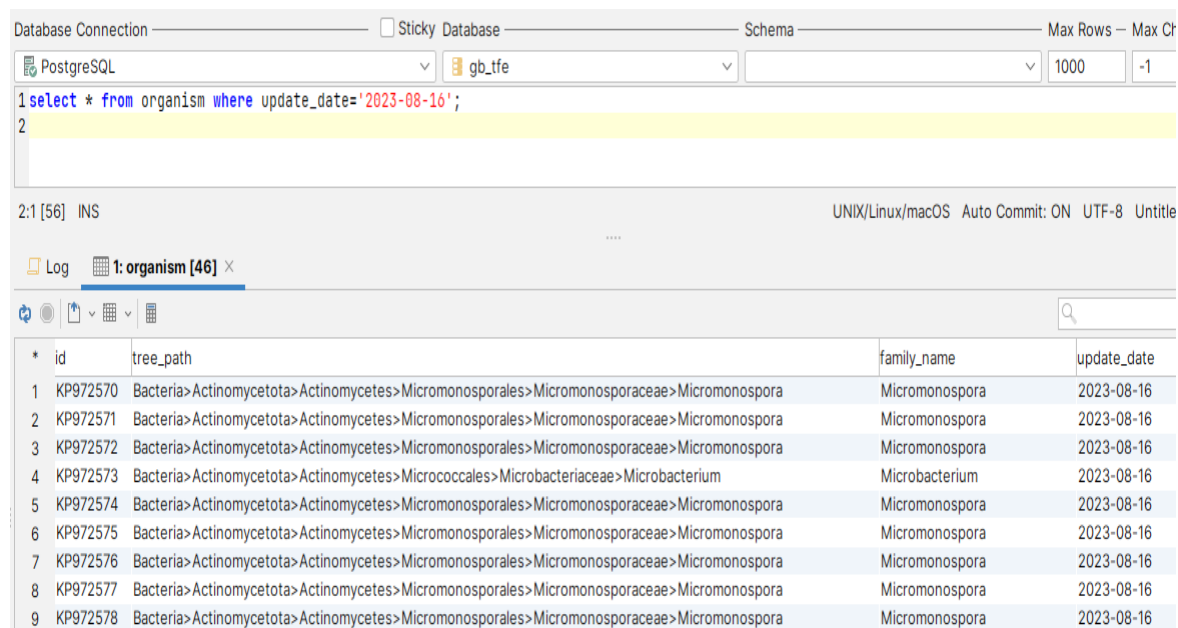
conn.commit()
conn.close()
```

Figure 28 : Script organisms_update.py.

Exécution du code en ligne de commande : **python3 organisms_update.py**

Ce script permet de mettre la base de données à jour, en considérant que la date du jour est 2023-08-16.

Ensuite je vérifie à l'aide d'une requête SQL les identifiants qui ont été mis à jour dans la base de données :



The screenshot shows a PostgreSQL query window with the following interface elements:

- Database Connection:** PostgreSQL
- Schema:** gb_tfe
- Max Rows:** 1000
- Max Columns:** -1
- Query:**

```
1 select * from organism where update_date='2023-08-16';
2
```
- Results:** 2:1 [56] INS. The results are displayed in a table with 4 columns: id, tree_path, family_name, and update_date. There are 9 rows of data.

* id	tree_path	family_name	update_date
1	KP972570 Bacteria>Actinomycetota>Actinomycetes>Micromonosporales>Micromonosporaceae>Micromonospora	Micromonospora	2023-08-16
2	KP972571 Bacteria>Actinomycetota>Actinomycetes>Micromonosporales>Micromonosporaceae>Micromonospora	Micromonospora	2023-08-16
3	KP972572 Bacteria>Actinomycetota>Actinomycetes>Micromonosporales>Micromonosporaceae>Micromonospora	Micromonospora	2023-08-16
4	KP972573 Bacteria>Actinomycetota>Actinomycetes>Micrococcales>Microbacteriaceae>Microbacterium	Microbacterium	2023-08-16
5	KP972574 Bacteria>Actinomycetota>Actinomycetes>Micromonosporales>Micromonosporaceae>Micromonospora	Micromonospora	2023-08-16
6	KP972575 Bacteria>Actinomycetota>Actinomycetes>Micromonosporales>Micromonosporaceae>Micromonospora	Micromonospora	2023-08-16
7	KP972576 Bacteria>Actinomycetota>Actinomycetes>Micromonosporales>Micromonosporaceae>Micromonospora	Micromonospora	2023-08-16
8	KP972577 Bacteria>Actinomycetota>Actinomycetes>Micromonosporales>Micromonosporaceae>Micromonospora	Micromonospora	2023-08-16
9	KP972578 Bacteria>Actinomycetota>Actinomycetes>Micromonosporales>Micromonosporaceae>Micromonospora	Micromonospora	2023-08-16

Figure 29 : Requête de vérification des identifiants qui ont été mis à jour.

Sur cette figure, on peut voir tous les identifiants qui ont été mis à jour dans la base de données (**id**), le nom de la famille (**family_name**) et le chemin des familles auxquelles ils appartiennent réellement (**tree_path**) à la date du jour (**update_date=2023-08-16**).

4. Ajout d'un système d'alerte

Cette partie consistera à alerter un utilisateur donné en cas de modification dans une famille qui l'intéresse. Le script utilisé est **monitoring.py**.

Pour mettre en place le système d'alerte, nous commençons par insérer nos informations dans la table monitoring, notamment l'adresse mail qui recevra le rapport, l'id et le nom de la famille à laquelle nous nous intéressons, ainsi que la dernière date de mise à jour.

Insertion d'une donnée dans la table monitoring

```
INSERT INTO monitoring (email, family_name, family_id, update_date)
VALUES ('maricatongang@gmail.com', 'Archaea', 7467, '2023-07-29');
```


En cas de mise à jour dans une famille spécifiée dans la table monitoring, un système d’alerte est envoyé à l’adresse mail renseigné via un serveur.

J’exécute monitoring.py pour avoir un rapport.

Exécution du script monitoring.py : **python3 monitoring.py**

```
marie@mariemint:~$ cd /data/org/  
marie@mariemint:/data/org$ python3 monitoring.py  
mail envoyé avec succès  
marie@mariemint:/data/org$
```

Rapport de l'arbre taxonomique [Boîte de réception x](#)


 **projectbio@willynzesseu.com**
À moi ▾

23 août 2023 23:33 (il y a 3 heu)

Rapport de mise à jour de l'arbre taxonomique
IL y a eu 1 familles modifiées depuis la dernière mise à jour.
Les familles modifiées sont les suivantes:


-/data/org/Bacteria

Veuillez vous mettre à jour

 **projectbio@willynzesseu.com**
À moi ▾

03:25 (il y a 7 minu)

Rapport de mise à jour de l'arbre taxonomique.
IL y a eu 3 organismes modifiés depuis la dernière mise à jour, pour la famille Archaea.

 **projectbio@willynzesseu.com**
À moi ▾

03:29 (il y a 3 minu)

Rapport de mise à jour de l'arbre taxonomique.
IL y a eu 0 organismes modifiés depuis la dernière mise à jour, pour la famille Archaea.

Figure 30 : Rapport envoyé par mail en cas de modification.


```

1 import psycpg2
2 import smtplib
3 from email.mime.text import MIMEText
4 from email.mime.multipart import MIMEMultipart
5
6 def send_mail(email, rapport):
7     sender = "projectbio@willynzesseu.com"
8     password = "!~P'D=LaISAH"
9     message = MIMEMultipart()
10    message["From"] = sender
11    message["To"] = email
12    message["Subject"] = "Rapport de l'arbre taxonomique"
13    message.attach(MIMEText(rapport, "plain"))
14
15    try:
16        with smtplib.SMTP_SSL("mail.willynzesseu.com", 465) as server:
17            # server.starttls()
18            server.login(sender,password)
19            server.send_message(message)
20            print("mail envoyé avec succès")
21    except smtplib.SMTPException as ex:
22        print("Erreur d'envoi du mail:", str(ex))
23        pass
24
25 def buildAndSendReport(email, family_name, data):
26     modified_data = len(data)
27     rapport = ""Rapport de mise à jour de l'arbre taxonomique.
28     IL y a eu {0} organismes modifiés depuis la dernière mise à jour, pour la famille {1}.""format(modified_data,family_name)
29     # print(rapport)
30     send_mail(email, rapport)
31
32 #establishing the connection
33 conn = psycpg2.connect(
34     database="gb_tfe", user='postgres', password='marica1996', host='127.0.0.1', port= '5432'
35 )
36 cursor = conn.cursor()
37 cursor.execute("SELECT * FROM monitoring")
38
39 # Fetch all rows method.
40 monitors = cursor.fetchall()
41
42 for monitor in monitors:
43     email = monitor[0]
44     family_name = monitor[1]
45     date = monitor[3]
46     id_parent = monitor[2]
47
48     request = ""SELECT * from organism where update_date > '{0}'
49     AND family name IN (
50         WITH RECURSIVE family_child AS(
51             SELECT * FROM family where id = {1}
52             UNION
53             SELECT F.id, F.family_name, F.id_parent FROM family F INNER JOIN family_child ON F.id_parent = family_child.id)
54         SELECT family_name FROM family_child
55         )""format(date, id_parent)
56
57     cursor.execute(request)
58     organism = cursor.fetchall()
59     buildAndSendReport(email,family_name,organism)
60     cursor.execute("""UPDATE monitoring SET update_date = CURRENT_DATE WHERE email = '{0}' AND family_id = {1}""format(email, id_parent))
61     conn.commit()
62
63 conn.close

```

Figure 31 : Le script monitoring.py.

IV. DISCUSSION ET PERSPECTIVES

La comparaison entre un arbre taxonomique et une base de données pour le stockage des données issues de GenBank a mis en lumière des éléments cruciaux concernant la gestion et l'utilisation optimale de ces données riches en informations. Le choix d'opter pour une base de données se révèle être une décision éclairée, compte tenu des avantages substantiels qu'elle offre en matière d'accessibilité, de flexibilité et de possibilités futures.

L'un des points les plus saillants de cette étude est la flexibilité inhérente d'une base de données par rapport à un arbre taxonomique. Alors que l'arbre taxonomique peut se révéler limitant en termes de stockage de métadonnées détaillées et de requêtes complexes, la base de données permet une structuration plus avancée et adaptée aux besoins spécifiques. Cela ouvre la voie à une exploration approfondie des données et à des analyses plus sophistiquées, favorisant ainsi de nouvelles découvertes dans le domaine de la taxonomie et de la génétique.

Un élément de cette discussion concerne l'interface utilisateur prévue pour la base de données. Bien que cette composante n'ait pas été réalisée dans le cadre de cette étude, il est essentiel de noter que son développement aurait pu constituer un atout majeur. Une interface conviviale facilite l'interaction avec la base de données, rendant l'accès et la manipulation des données plus accessibles à un plus large éventail d'utilisateurs, y compris ceux moins familiers avec les aspects techniques de la gestion des bases de données.

Une perspective prometteuse pour l'avenir serait de développer cette interface utilisateur, offrant ainsi une expérience complète et immersive pour les utilisateurs finaux. Cela pourrait inclure des fonctionnalités telles que des outils de requête visuelle, des filtres intuitifs et la visualisation graphique des données. De plus, la création d'un système de connexion pour les utilisateurs permettrait une collaboration plus fluide et sécurisée, renforçant ainsi la valeur de la base de données dans un contexte collaboratif.

L'accès à une base de données depuis diverses localisations se révèle plus simple et sécurisé que le partage de fichiers pour représenter un arbre taxonomique. En utilisant une base de données hébergée sur un serveur, l'utilisateur peut se connecter au moyen d'un plugin et d'un mot de passe, sans nécessiter un accès direct au serveur. En revanche, partager un arbre taxonomique via des fichiers exige souvent de donner à l'utilisateur un accès SSH complet, entraînant des risques de sécurité en lui conférant un contrôle étendu sur le système. En somme, l'option d'accès à la base de données offre une approche plus gérable et sécurisée, préservant l'intégrité du serveur tout en facilitant l'utilisation des données.

Un autre aspect à considérer est la portabilité de la base de données. Le fait qu'elle puisse être utilisée à la fois sur Windows et Linux est une caractéristique importante, permettant une adoption plus large et flexible par différents types d'utilisateurs et de systèmes informatiques.

V. CONCLUSION

En conclusion de ce travail de fin d'étude, il est clair que le choix d'une base de données pour le stockage et la gestion des données issues de genbank s'avère être une décision judicieuse et pertinente. L'analyse comparative entre un arbre taxonomique et une base de données a mis en évidence les nombreux avantages offerts par une base de données, tant du point de vue de l'accessibilité que de la flexibilité.

L'arbre taxonomique, bien qu'il puisse fournir une représentation visuelle de la relation entre les différentes entités biologiques, présente des limitations majeures en termes de capacité à stocker des métadonnées détaillées et à permettre des requêtes complexes. En revanche, une base de données offre une structure organisée et évolutive qui facilite la recherche, la récupération et la manipulation des données à différents niveaux de granularité. L'utilisation d'une base de données permet également de mettre en place des systèmes de gestion avancés, tels que la gestion des droits d'accès, la traçabilité des modifications et la possibilité de mettre en place des interfaces conviviales pour les utilisateurs. De plus, la capacité à interroger efficacement les données au sein de la base de données ouvre des perspectives de recherche et d'analyse plus approfondies, favorisant ainsi la découverte de nouvelles connaissances dans le domaine de la taxonomie et de la génétique.

En somme, ce travail souligne les avantages significatifs d'une base de données dans le contexte du stockage et de la gestion des données issues de genbank par rapport à un simple arbre taxonomique. Ce choix offre des opportunités accrues pour la recherche et l'analyse tout en garantissant une meilleure accessibilité et une manipulation plus aisée des données. Cependant, il convient de noter que la mise en œuvre d'une base de données nécessite une planification minutieuse et une gestion continue pour en tirer pleinement parti.

VI. REFERENCES BIBLIOGRAPHIQUES

- (1) <http://www.ncbi.nlm.nih.gov/>
- (2) <http://www.ddbj.nig.ac.jp/>
- (3) <http://www.ebi.ac.uk/Jembl/>
- (4) <http://www.ncbi.nlm.nih.gov/GenBank/>
- (5) Benson,D.A., Karsch-Mizrachi,I., Clark,K., Lipman,D.J., Ostell,J. and Sayers,E.W. (2012) GenBank. Nucleic Acids Res., 40, D48–D53.
- (6) NCBI Resource Coordinators. (2013) Database resources at the National Center for Biotechnology Information. Nucleic Acids Res., 41, D8–D20).
- (7) <https://www.ncbi.nlm.nih.gov/taxonomy>
- (8) Federhen,S. (2012) The NCBI Taxonomy database. Nucleic Acids Res., 40, D136–D143
- (9) <https://www.ncbi.nlm.nih.gov/nuccore/AB000100>
- (10) "Phylogenetic Trees Made Easy: A How-To Manual" , Barry G. Hall, 2001.
- (11) <https://connect.ed-diamond.com/GNU-Linux-Magazine/glmfhs-073/la-bioinformatique-avec-biopython>
- (12) <https://ftp.ncbi.nlm.nih.gov/genbank/daily-nc/>
- (13) <https://confluence.dbvis.com/display/UG100/Getting+Started>
- (14) <https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-upsert/>
- (15) <https://www.postgresql.org/docs/current/intro-what-is.html>
- (16) <https://planet.postgresql.org/>

- (17) <https://www.2ndquadrant.com/en/blog/pg-phriday-10-things-postgres-could-improve-part-3/>
- (18) Développement d'une base de données bioinformatique spécialisée GBANK UQAM, Rabah Djema, Québec Montréal, Mai 2008.
- (19) <https://stackoverflow.com/questions/5420789/how-to-install-psycopg2-with-pip-on-python>
- (20) https://www.tutorialspoint.com/python_data_access/python_postgresql_databaseconnection.htm

VII. ANNEXES

❖ Parseorginline.py:

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  #
4  import os
5  import sys
6  import re
7  red='\033[101m'
8  blue='\033[94m'
9  normal='\033[0m'
10 ss=re.compile("^s")
11 z=sys.stdin
12 l=z.readline().strip()
13 Okaybase=["Eukaryota","Bacteria","Viruses","Archaea","other sequences","unclassified sequences","cellular organisms","Unclassified"]
14 Okay=[]
15 Okay.extend([s+"." for s in Okaybase])
16 Okay.extend([s+";" for s in Okaybase])
17 rep=list("()*/*@+::")
18 todo=set()
19 LOC={}
20 OL=[]
21 broken=open("/data/org/broken","a")
22 while l:
23     if l.startswith("LOCUS"):locus=l.split()[1]
24     elif l.startswith(" ORGANISM"):OL=[]
25     elif not ss.match(l):
26         phyl=" ".join(OL)
27         if not [OL[0].startswith(ok) for ok in Okay].count(True):
28             broken.write(locus+"\n")
29             broken.write(phyl+"\n")
30             while not [OL[0].startswith(ok) for ok in Okay].count(True):
31                 broken.write(phyl+"\n")
32                 OL=OL[1:]
33                 phyl=" ".join(OL)
34             for c in rep:
35                 phyl=phyl.replace(c,"_")
36                 phyl=phyl.replace(";","/")
37                 phyl=phyl.replace(" ","_")
38                 phyl="/data/org/"+phyl[:-1]
39                 todo.add(phyl)
40                 if phyl in LOC:LOC[phyl].append(locus)
41                 else:LOC[phyl]=[locus]
42             else:OL.append(l.strip())
43             l=z.readline()
44 print("making %i directories"%len(LOC))
45 done=0
46 for p in todo:
47     os.system("mkdir -p %s"%p)
48     gbfn="%s/gb"%p
49     gbfmt="%s/gb.up"%p
50     if os.path.exists(gbfn):os.system(f"cp {gbfn} {gbfmt}")
51     gbfile=open(gbfmt,"a")
52     gbfile.write("\n".join(LOC[p]))
53     gbfile.write("\n")
54     gbfile.close()
55     os.system("sort -u -o %s %s"%(gbfmt,gbfn))
56     if os.path.exists(gbfn):
57         if os.popen(f"diff {gbfn} {gbfmt}").readlines():
58             print(f"{red}{gbfn}{normal}")
59             os.system(f"mv {gbfmt} {gbfn}")
60             done+=1
61         else:
62             print(f"{blue}{gbfn}{normal}")
63             os.remove(gbfmt)
64     else:
65         print(f"{red}{gbfn}{normal}")
66         os.system(f"mv {gbfmt} {gbfn}")
67         done+=1
68 print(f"changed {done} gb files")
69 if done:os.system("touch changed")
```

❖ prune.sed

```
marie@mariemint:/data/org$ cat prune.sed
/^LOCUS/ p
/^ ORGANISM/,/^[\^ ]/ p
```

❖ Script famille.py

```
1 import os
2
3 id = 1
4 query = ""
5
6 def add_directory(name, id_parent):
7     global query
8     global id
9     query += "INSERT INTO family (id, family_name, id_parent) VALUES ({0}, '{1}', {2});\n".format(id, name, id_parent)
10    id += 1
11    return id - 1
12
13 def traverse_and_insert(root_path, id_parent=None):
14     if os.path.exists(root_path):
15         entries = os.listdir(root_path)
16         if entries:
17             for entry in entries:
18                 full_path = os.path.join(root_path, entry)
19
20                 if os.path.isdir(full_path):
21                     sub_id_parent = add_directory(entry, id_parent)
22
23                     # Recursively traverse subdirectories
24                     traverse_and_insert(full_path, sub_id_parent)
25
26 traverse_and_insert("/data/org/")
27
28 sqlQuery = open("/data/org/BDD/family.sql", "w+")
29 sqlQuery.write(query)
30 sqlQuery.close()
```

❖ Fichier family.sql

```
monito_Carelle.py  monitor_Carelle_serveurJoel.py  famille.py  parseorgonline.py  prune.sed
1 INSERT INTO family (id, family_name, id_parent) VALUES (1, 'Viruses', None);
2 INSERT INTO family (id, family_name, id_parent) VALUES (2, 'Pandoravirus', 1);
3 INSERT INTO family (id, family_name, id_parent) VALUES (3, 'Spiraviridae', 1);
4 INSERT INTO family (id, family_name, id_parent) VALUES (4, 'Alphaspiravirus', 3);
5 INSERT INTO family (id, family_name, id_parent) VALUES (5, 'Pospiviroidae', 1);
6 INSERT INTO family (id, family_name, id_parent) VALUES (6, 'Pospiviroid', 5);
7 INSERT INTO family (id, family_name, id_parent) VALUES (7, 'Pospiviroid plvd', 6);
8 INSERT INTO family (id, family_name, id_parent) VALUES (8, 'Apscaviroid', 5);
9 INSERT INTO family (id, family_name, id_parent) VALUES (9, 'Apscaviroid aclsvd', 8);
10 INSERT INTO family (id, family_name, id_parent) VALUES (10, 'Apscaviroid pvd', 8);
11 INSERT INTO family (id, family_name, id_parent) VALUES (11, 'Apscaviroid pvd-2', 8);
12 INSERT INTO family (id, family_name, id_parent) VALUES (12, 'Apscaviroid glvd', 8);
13 INSERT INTO family (id, family_name, id_parent) VALUES (13, 'Apscaviroid plvd-I', 8);
14 INSERT INTO family (id, family_name, id_parent) VALUES (14, 'Apscaviroid dvd', 8);
15 INSERT INTO family (id, family_name, id_parent) VALUES (15, 'Apscaviroid cvd-VII', 8);
16 INSERT INTO family (id, family_name, id_parent) VALUES (16, 'Cocadviroid', 5);
17 INSERT INTO family (id, family_name, id_parent) VALUES (17, 'Coleviroid', 5);
18 INSERT INTO family (id, family_name, id_parent) VALUES (18, 'Coleviroid cbvd-6', 17);
19 INSERT INTO family (id, family_name, id_parent) VALUES (19, 'Coleviroid cbvd-5', 17);
20 INSERT INTO family (id, family_name, id_parent) VALUES (20, 'Hostuviroid', 5);
21 INSERT INTO family (id, family_name, id_parent) VALUES (21, 'ssDNA_viruses', 1);
22 INSERT INTO family (id, family_name, id_parent) VALUES (22, 'Geminiviridae', 21);
23 INSERT INTO family (id, family_name, id_parent) VALUES (23, 'Begomovirus', 22);
24 INSERT INTO family (id, family_name, id_parent) VALUES (24, 'Parvoviridae', 21);
25 INSERT INTO family (id, family_name, id_parent) VALUES (25, 'Parvovirinae', 24);
```

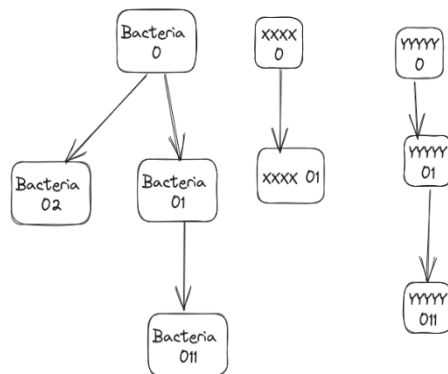
❖ Script organism.sh :

```
#!/bin/bash

directory="/data/org/BDD/organism"
extension=".sql" # Change this to your desired extension

for file in "$directory"/*"$extension"; do
    psql -U postgres -w -d gb_tfe -f "$file"
done
```

❖ Notes prises lors de ma réflexion



Famille
ID, Nom, ID_Parent

ORGANISME
ID, FAMILLE, FAMILLE_PATH, DATE_MISE_AJOUR

MONITORING
email, NOM_FAMILLE, ID_FAMILLE, DATE_MISE_AJOUR

Pour construire la base de données
- parcourir l'arbre existant récupérer les IDs dans les fichiers GB et faire de simples requêtes SQL d'insertion
- Pour les mise à jour parcourir le fichier de mise à jour, récupérer l'ID et le nom de famille et faire une requête update soit un insert si l'ID n'existait pas avant

Pour vérifier les changements sur une famille donnée
- prendre tous les noms de familles dont les IDs sont inférieurs à l'ID de la famille à surveiller

```

SELECT id, famille FROM organisme
WHERE famille IN (SELECT famille FROM FAMILLE where id < id_parent)
AND date_mise_a_jour > date
  
```

Envoi du rapport
Mise à jour de la table Monitoring

je veux monitorer la famille Asgard
marica@gmail.com, Asgard, 50, 2023-02-10

Solution 1

je cherche toutes les familles enfants de la famille à surveiller (F1, F2...)
je recherche tous les id avec une date de mise à jour supérieure au 2023-02-10
je vérifie si la famille des id appartient à la liste des sous familles

je peux rapporter et à la fin je met à jour la table monitoring avec la date du 2023-02-11

Solution 2

recherche dans la table organisme tous les organismes ayant une date de mise à jour supérieure à 2023-02-10 et dont le family_path contient le nom de la famille à surveiller

Solution 1

```

select * from organism where update_date > 2023-02-10 and
family_name in (
  select WITH recursion family_name from family where parent_id == 50 UNION ('Asgard')
)
  
```

Solution 2

```

select * from organism where update_date > 2023-02-10 and
family_path like '%Asgard%'
  
```

❖ Programme utilisé pour faire les captures d'écran : **SCROT**

❖ Programme utilisé pour représenter graphiquement l'arbre : **TREE**