RAPPORT DE PROJET BDDR

Groupe TERNAT Clément - RICHARD Paul-Antoine

Notre rapport se divise en 3 parties majeures, le contexte, le peuplement et l'application web, et dans chacune de ces parties nous verrons quelles sont les choses qui ont pu nous poser problème dans le cadre de notre réflexion.

PARTIE 1. CONTEXTE

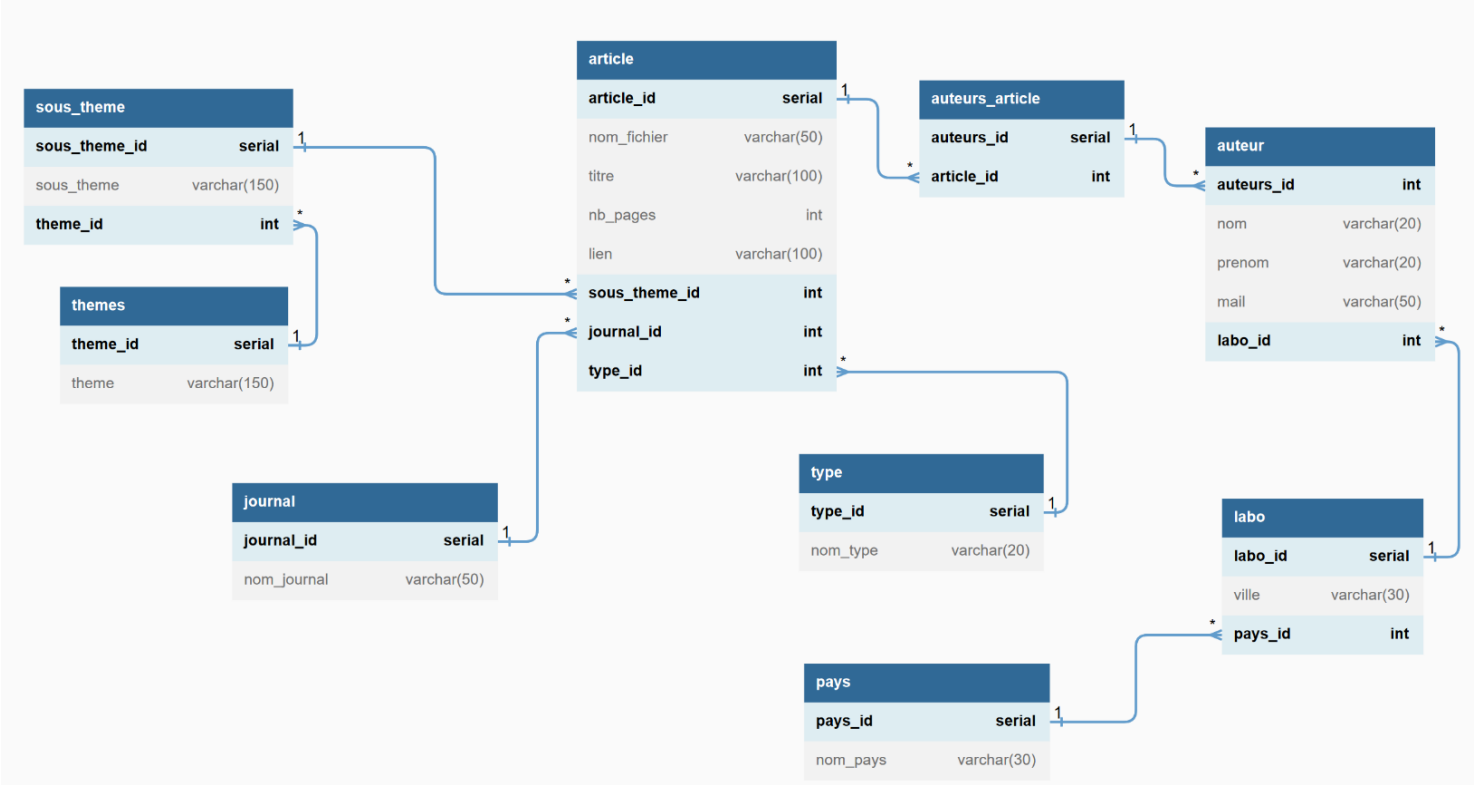
D'abord, nous avons pris connaissance des contraintes et des objectifs de ce projet. Nous devions récupérer des informations sur des articles à partir d'une archive contenant les éléments suivants :

- un fichier *metadata.csv* qui contient l'essentiel des informations de chaque article tel que les doi, titres, dates, url, journaux, etc...

- un dossier *target_tables* contenant des fichiers *csv* classés par thèmes et sous-thèmes, on y trouve également les types de publications de certains articles.

- deux dossiers *pdf_json* et *pmc_json* contenant des fichiers *json* dans lesquels on retrouve toutes les informations sur les auteurs des articles.

A partir de ces informations, nous avons pu commencer à concevoir notre premier modèle de données, représenté par le schéma ci-dessous. Avec du recul, nos intuitions étaient correctes, mais nous manquions d'informations sur les données auxquelles nous avions accès.



PARTIE 2. PEUPLEMENT

Dans cette partie nous allons revoir chacune des étapes du peuplement, du nettoyage des données à la création de notre base de données.

1) Nettoyage des données

Nous nous sommes rendu compte que le dossier *0_table_formats_and_column_definitions* du dossier *target_tables* était un document purement informatif et donc inutile dans notre cas, de même pour *unsorted_tables* qui n'est qu'un simple doublon des fichiers *csv*.

Le reste du nettoyage de données se déroule dans les parties pythons que nous allons voir ci-dessous.

2) ART_JOUR.py

L'objectif de cette sous-partie est de filtrer les données du document *metadata.csv* ainsi que de créer des listes qui nous permettront de remplir nos tables ultérieurement. C'est pourquoi nous avons commencé par supprimer toutes les données que nous pensions inutiles du document, dont la colonne correspondant au doi de chacun des articles.

Afin de retirer les articles en double, nous avons décidé d'identifier chaque article par son titre, mais nous nous sommes rapidement rendu compte qu'il pouvait y avoir des majuscules ou des symboles qui différaient entre un même article ce qui posait problème pour l'identification.

C'est pourquoi nous avons décidé de faire un tri des articles par doi, chaque doi étant l'identifiant exact d'un article. Il restait alors un problème, les articles qui n'avaient pas de doi ne pouvaient pas être traités correctement selon cette méthode, nous avons donc décidé d'attribuer à ces articles les dois *no_doi_i*. On a alors su qu'il y avait exactement 1 055 359 articles à étudier et traiter.

Enfin, nous avons créé les listes des titres, dates et liens de chacun des articles qui nous serviront dans la création de la table *article* plus tard, ainsi que la liste qui formera plus tard la table *journaux*.

3) THEMES_TYPES.py

Dans cette sous-partie notre objectif est de récupérer les thèmes, sous-thèmes, et types des articles grâce aux informations contenues dans le dossier *target_tables*.

Study Link
https://www.sciencedirect.com/science/article/pii/S2468266720300736
https://doi.org/10.1002/14651858.CD011621.pub4
https://onlinelibrary.wiley.com/doi/pdfdirect/10.1111/jgs.16490
https://doi.org/10.1016/j.jaip.2020.04.012
https://doi.org/10.1101/2020.04.10.20061267
https://doi.org/10.31219/osf.io/bv28d
https://doi.org/10.1377/hlthaff.2020.00382
https://doi.org/10.1002/14651858.cd013574
https://doi.org/10.1101/2020.04.01.20049528
https://dx.doi.org/10.2139/ssrn.3569098
https://link.springer.com/content/pdf/10.1007/s40615-020-00756-0.pdf
https://doi.org/10.31234/osf.io/jkfu3

Pour se faire, nous avons ouvert l'intégralité des documents contenus dans ce dossier. Il se trouve que dans chacun des documents .csv qui sont dans les sous-dossiers il existe une colonne *Study Link*. Dans cette colonne nous trouvons une chaîne de caractère qui peut contenir un identifiant de type doi, pubmed ou PMC, et c'est grâce à cette information que l'on pourra associer à chacun des articles uniques les nouvelles informations obtenues.

C'est à ce stade que nous nous sommes rendu compte qu'il pouvait y avoir plusieurs types, thèmes et sous-thèmes par articles. C'est pourquoi nous avons rajouté des tables de transition sur notre schéma et plus globalement dans notre base de données. C'est donc assez logiquement ce que va mettre en place la suite du THEMES_TYPES.py.

Doi	Sous_theme	Type
10.1002/jmv.25674	['Diagnosing SARS-COV-2 with Nucleic-acid based tech', 'Development of a point-of-care test and rapid bed-side tests']	['Systematic Review', 'Investigative Study']
10.1002/jmv.25727	['Diagnosing SARS-COV-2 with antibodies', 'Development of a point-of-care test and rapid bed-side tests']	['Clinical trial', 'Investigative Study']
10.1002/jmv.25762	['What regional genetic variations (mutations) exist']	['Genomic Study']

Une fois cette partie terminée, nous avons donc cinq nouvelles tables prêtes à être remplies à l'aide des listes et des dataframes que nous avons créés.

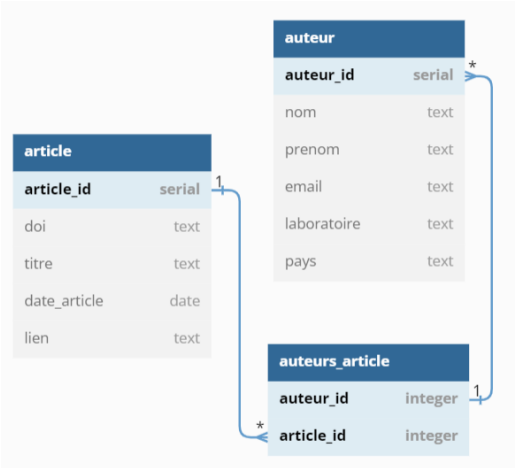
4) CSVAUTEURS.py et AUTEURS_LABOS.py

Dans ces deux dernières sous-parties notre objectif est de créer les listes qui contiendront toutes les informations sur les auteurs, c'est-à-dire les dois des articles auxquels ils sont associés, leurs nom et prénom, leurs emails, les institutions / laboratoires auxquels ils sont associés, ainsi que les pays liés à chacun de ces laboratoires. Toutes ces informations sont contenues dans les fichier *json* des dossiers *pdf_json* et *pmc_json.*

Pour cela nous allons parcourir toutes les lignes du dataframe que nous avions créé dans la partie ART_JOUR.py, et puis nous allons comparer les colonnes *pdf_json_files* et *pmc_json_files* de ce dataframe à tous les noms des fichiers présents dans les dossiers cités précédemment. Si un fichier existe, on récupère les informations sur les auteurs et les laboratoires à l'aide de la fonction filedotjson et on les rajoute en tant que nouvelle ligne du dataframe *auteurs*. Etant donné que l'on vérifie chaque *pdf.json* et *pmc.json*, il y aura de nombreux doublons.

Nous avons donc récupéré toutes les informations nécessaires, nous devons donc faire en sorte de créer les listes et dataframes qui rempliront les tables lors du peuplement.

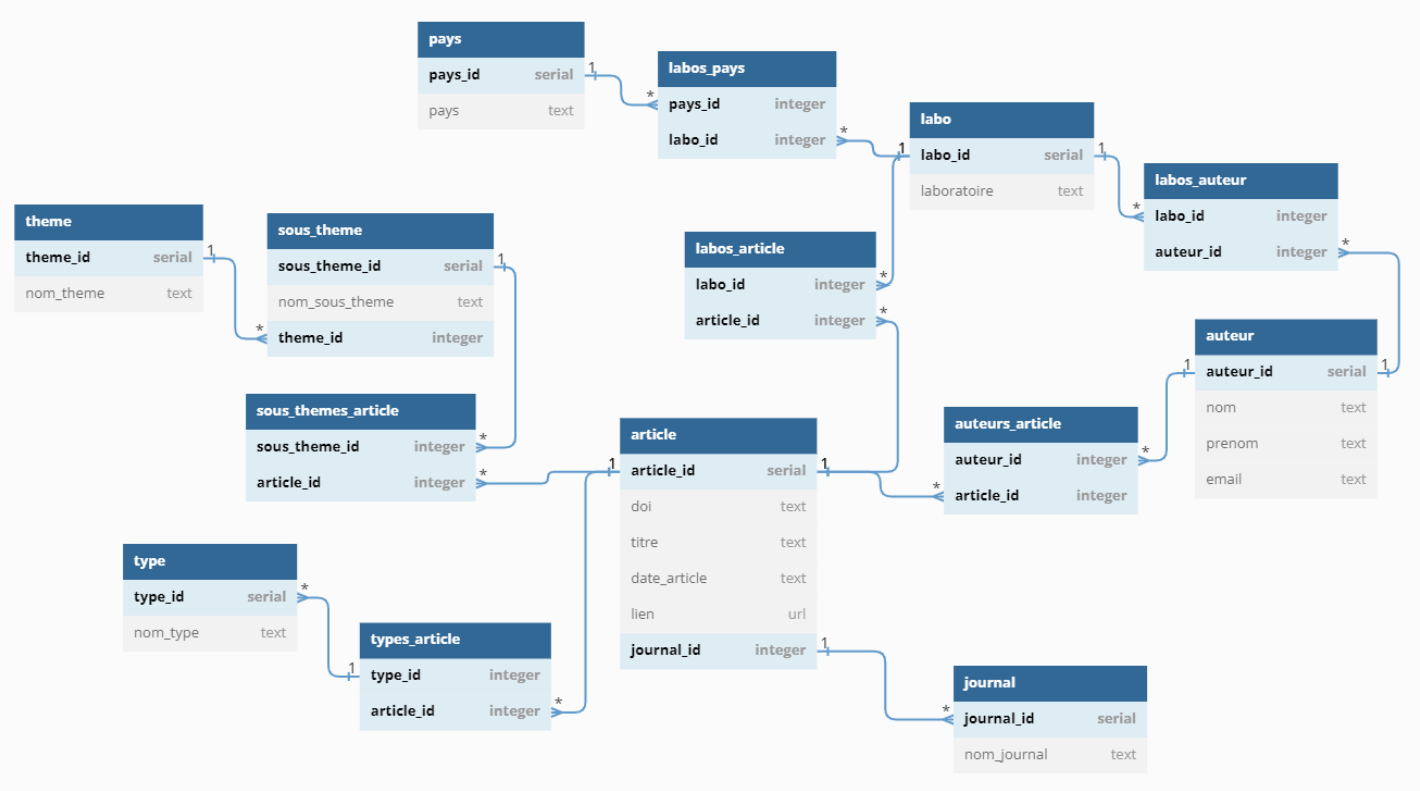
Au départ nous souhaitions former des listes afin de créer les tables suivantes :



Nous sommes donc partis du dataframe *auteurs* de la section précédente. La première étape a logiquement été de *groupby* le dataframe par combinaisons Prénom-Nom, en veillant à ne pas inclure de doublon sur chaque colonne (doi, email, labo…).

On obtient donc pour chacun des auteurs la liste de tous ses emails, les articles qu'il a rédigé grâce aux doi et de tous les laboratoires auxquels il est affilié. On s'est alors rendu compte qu'il pouvait y avoir plusieurs laboratoires associés à chaque auteur. Cela veut donc dire qu'une table de transition supplémentaire est nécessaire entre les tables *labo* et *auteur*.

Il en est de même pour les pays, nous nous rendons compte qu'il peut y en avoir plusieurs pour un même laboratoire, et que de plus ces pays peuvent être mal orthographiés.



Ci-dessus notre schéma de tables final.

5) Création de la base de données

Pour la dernière étape du peuplement nous avons commencé par créer une application, puis à créer ses modèles. C'est donc logiquement que nous avons créé une classe par table dans le *models.py*, puis nous avons effectué les migrations afin de créer les tables sur PostgreSQL. En ce qui concerne la base de données, nous avons fait le choix de l'héberger directement sur nos ordinateurs (méthode localhost). Enfin nous avons rempli nos tables grâce au module *psycopg2* de python.

PARTIE 3. APPLICATION DJANGO

Dans cette partie nous allons voir comment créer des *Views*, des Templates et tout ce qui est nécessaire pour pouvoir afficher nos données à l'écran.

1) URLs, Views, Templates

D'abord, nous avons décidé de créer une vue permettant de récupérer toutes les infos liées à un article en particulier. Spécifié dans une url, on récupère le doi de l'article, puis la vue *page_article* nous récupère les informations à l'aide de *class.objects.filter* . Nous avons répété le même processus pour afficher les informations sur les auteurs, laboratoires, pays, journaux.

Ensuite, nous avons décidé de créer plusieurs types de recherche par type d'information. Nous avons donc créé une vue par type de recherche.

- Concernant les recherches par titre, laboratoire et journal, il suffit d'effectuer une recherche par mots-clés du type *covid france children* pour afficher tous les articles contenant ces mots-clés, *ou univ paris* pour afficher tous les laboratoires contenant ces mots-clés. Cela fonctionne grâce à un *filter(titre__icontains=…)* . De même pour les journaux.

- On peut effectuer une recherche par doi exact, grâce à un *get(doi=doi)*

- Etant donné que les pays ne sont pas bien renseignés, il nous a semblé intéressant d'inclure du javascript nous permettant d'afficher en temps réel les pays contenant le mot-clé que l'on entre.

- Nous avons fait une recherche par thème/sous-thème avec la possibilité de chercher par titre. On renseigne un thème et un sous-thème et on obtient tous les articles correspondants, de même en renseignant un titre pour affiner les résultats.

- En ce qui concerne les types des articles, il suffit de sélectionner un type pour afficher dans l'ordre décroissant les journaux ayant publié le plus d'articles de ce type.

- Les recherches par dates sont celles qui ont été les plus difficiles à mettre en place, nous avons dû créer deux nouvelles classes afin de différencier les articles dont la date est complète (année-mois-jour) des articles n'ayant que l'année. De ce fait, nous pouvons convertir nos dates au format date grâce au module *datetime*, et effectuer des requêtes *filter* de manière beaucoup plus optimisée. Nous avons fait le choix de permettre des recherches par date exacte et par année. Concernant les histogrammes, nous pouvons afficher un histogramme entre deux dates, en spécifiant un choix de tri : date exacte, semaine, mois, année.