Bloc 1
Moran HANANE

Archives numérisées de revues françaises

dépot GitHub: https://github.com/MoranHanane/Projet_Bloc_1.git
Fichier d'execution: Bloc1_Moran_HANANE.py

TABLE DES MATIÈRES

Introduction: contexte et intérêt

J'ambitionne éventuellement de travailler dans la numérisation et le traitement de données culturelles et historiques et ce projet était pour moi l'occasion de mettre en pratique ce que nous avons étudié en cours avec deux de mes passions: la presse et l'Histoire.

J'ai en effet décidé de compiler des données de revues françaises publiées depuis le 18è siècle, puis numérisées dans une base de données afin de pouvoir les exploiter, trier l'information… Ce travail contribue également à la **sauvegarde du patrimoine historique et culturel à l'ère du numérique.**

Mon objectif initial était d'extraire et de présenter tant les métadonnées des revues (titre, auteur, numéro d'article, etc..) que des imprimés d'article à proprement parler. J'ai revu mes ambitions à la baisse en constatant les difficultés rencontrées pour extraire les données que je souhaitais spécifiquement lors du scrapping du site web concerné (https://gallica.bnf.fr). Nous aborderons en détail ces difficultés de scrapping.

Extraction des données

Précautions préalables

J'ai consulté les sites de documentation de l'API et le fichier robot.txt (https://gallica.bnf.fr/robots.txt) avant le scrapping et l'application de mes requêtes API.
La documentation de l'API m'a été indispensable pour spécifier les paramètres de ma requête API, comme on peut le voir ci-dessous:

```python
BASE_URL = "https://gallica.bnf.fr/SRU"
QUERY = '(dc.type all "fascicule") and (ocr.quality all "Texte disponible")'
MAX_RECORDS = 50      # Limite imposée par l'API
TOTAL_RESULTS = 5000  # Nombre d'URL que je souhaite extraire
start_record = 0
results = []

while len(results) < TOTAL_RESULTS:
    params = {
        "operation": "searchRetrieve",
        "version": "1.2",
        "startRecord": start_record,
        "maximumRecords": MAX_RECORDS,
        "collapsing": "true",
        "exactSearch": "false",
        "query": QUERY
    }
```

Sans consulter la documentation il m'aurait été impossible de déterminer le **terme "fascicule" dans mon champ "Query"**, par exemple, car l'API regroupe toute sorte de collection d'objets culturels et les termes "journaux" et "magazines" n'avaient pas fonctionné.

La consultation du fichier robots.txt m'a permis de déduire des règles à respecter pour mon scrapping et mon interrogation d'API:
"*User-agent: FacebookBot*
Crawl-delay: 1

User-agent: PetalBot
Crawl-delay: 1"
" m'a indiqué par exemple que les robots FacebookBot et PetalBot avaient un délai de une seconde à respecter entre chaque requête… mais qu'a priori aucune règle de ce type ne s'appliquait aux autres robots.

J'ai malgré tout instauré un délai de une secondes entre chaque requête dans mon script avec la **commande "time.sleep(1)**" pour m'assurer que j'allais scrapper sans aucun risque de saturer les serveurs lors de mon extraction. C'est une bonne pratique.

Méthode d'extraction des données

J'ai rapidement rencontré des obstacles en essayant d'effectuer un scrapping "classique" sur le site de Gallica en utilisant Beautifulsoup ou Selenium avec les Xpaths tel que nous l'avions vu en TD car un dropdown qui ne s'affichait qu'après quelques secondes contient l'ensemble des métadonnées que je souhaite extraire, et ce pour chacune des revues!

J'ai donc décomposé mon script d'extraction des données en plusieurs étapes afin de procéder méthodiquement:

Extraction et stockage des URL contenant les métadonnées des revues

J'ai défini une fonction qui extrait les URLs des notices détaillées de la réponse XML de Gallica , puis les stocke dans une liste.

J'ai ensuite appliqué cette fonction avec mes paramètres spécifiés dans l'API de recherche afin de cataloguer une liste d'URLs pour l'étape suivante. Je me suis notamment assuré de ne stocker que les URL qui retournaient un code "200" (requête fructueuse).

Scrapping de la liste des URL afin d'en extraire toutes les métadonnées récupérables

Une fois les URL stockées dans une liste, puis dans un fichier, Selenium va scrapper les pages renvoyées par chacune des URL de la liste afin d'extraire toutes les métadonnées possibles.

J'ai rajouté pour cela deux éléments dans mon code: **"driver.set_page_load_timeout(15)"** et **"wait.until(EC.element_to_be_clickable((By.CSS_SELECTOR, "div#moreInfosRegion"))).click()** " qui m'a permis de m'assurer que la simulation de clic par Selenium avait lieu uniquement quand le bouton du dropdown devenait réellement interactif (j'ai fixé une marge de sûreté à 15 secondes pour le timeout); je n'avais pas cela dans ma première version et c'est pour cela que je n'arrivais pas à extraire de métadonnées. Toutes mes données scrappées avec succès ont été stockées dans une liste "metadata", laquelle a été exportée en local au format

Précautions en cours et en fin de traitement des données

Ayant eu la mauvaise surprise les deux premières fois de constater que mon scrapping s'était interrompu en cours de route alors qu'il tournait depuis plusieurs heures, j'ai instauré un **mécanisme de sauvegarde partielle des métadonnées en local toutes les 100 URL traitées**.

```
if i % 100 == 0:
    try:
        with open("metadata_gallica_partial.json", "w", encoding="utf-8") as f:
            json.dump(metadata_list, f, indent=4, ensure_ascii=False)
        pd.DataFrame(metadata_list).to_csv("metadata_gallica_partial.csv", index=False, encoding="utf-8")
        print(f" Sauvegarde partielle après {i} URL.")
    except Exception as e:
        print(f"Erreur lors de la sauvegarde partielle : {e}")
```

J'ai également **stocké toutes les erreurs rencontrées** lors du scrapping si besoin d'analyse complémentaire dans une **liste error_log**. Cela m'a été utile pour partager mes problèmes d'extraction des données sur le forum "stack overflow".

Développer des requêtes de type SQL d'extraction des données

Commandes via MySQLShell

L'utilisation de MySQL Shell est relativement intuitive une fois que l'on est habitué à utiliser d'autres outils d'invite de commande comme Anaconda, l'invite de commande native windows, etc…:

1. **\sql**
    - l'invite de commande est en JavaScript par défaut;
2. **\connect root@localhost**
3. **create database [NOM DE LA BDD]**
    - Cette commande est à effectuer seulement lors de la création de la base de données; elle correspond au CREATE de CRUD
4. **use [NOM DE LA BDD]**

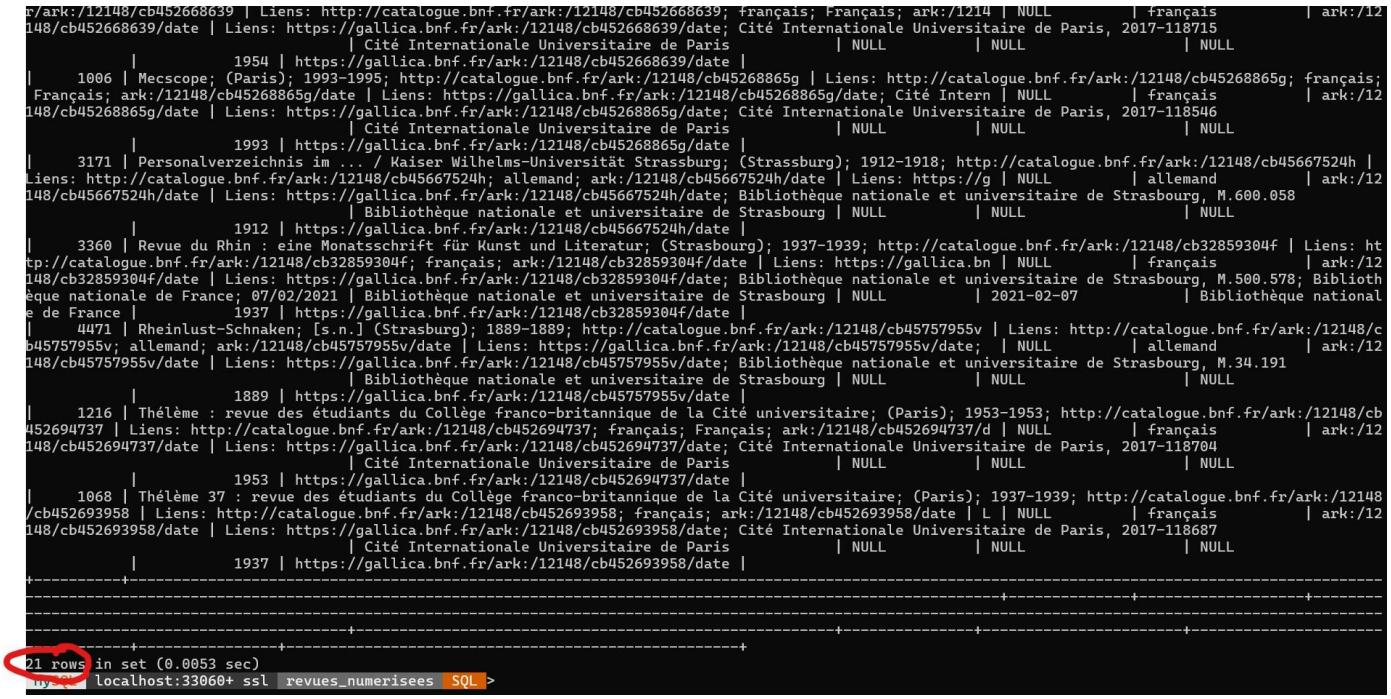
Requêtes d'extraction de donnée

J'ai mis en place les requêtes READ suivantes afin d'illustrer leur potentiel pour ma BDD SLQ:

1. Execution: sql_query_1 = """

```
SELECT * FROM Revue
WHERE Source IN ('Cité Internationale Universitaire de Paris', 'Bibliothèque nationale et universitaire de Strasbourg')
ORDER BY Titre ASC;
"""
mysql_cursor.execute(sql_query_1)
result_1 = mysql_cursor.fetchall()
print("Revues des sources universitaires :", result_1)
```

Bloc 1 - Moran HANANE - Archives numérisées de revues françaises



Cette requête aboutit à 21 sorties, ce qui correspond aux 21 entrées cumulées du CIU de Paris (13) et de la BNU de Strasbourg (8).

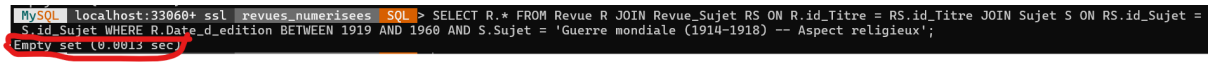
```
print(df["Source :"].value_counts())
```

```
Source :
                                                  4519
Bibliothèque nationale de France                   399
Musée de la Résistance nationale / Champigny        35
Cité Internationale Universitaire de Paris          13
Bibliothèque nationale et universitaire de Strasbourg  8
Musée Air France                                    6
Croix Rouge Française                               2
Bibliothèque interuniversitaire Cujas               1
Bibliothèque de la Cour de cassation                1
Espace Air Passion                                  1
Bibliothèque de l'Académie nationale de médecine    1
AgroParisTech / Centre de Nancy                     1
Bibliothèque municipale d'Abbeville                 1
Réseau Canopé                                       1
Fédération Française de BasketBall/Musée du Basket  1
Archives départementales des Hautes                 1
Bibliothèque du Musée national de la Marine         1
Les Amis de Masques et de Persona                   1
Musée Air France num en l'état                      1
Médiathèques de Plaine Commune                      1
Archives départementales de l'Ardèche               1
Ville de Paris / Bibliothèque de l'Hôtel de Ville (BHdV)  1
Centre d'Études Alexandrines (CEAlex)               1
Name: count, dtype: int64
```

Cela démontre que les données ont été correctement insérées dans la base de données SQL à partir de mon Dataframe et que la requête est correctement effectuée.
Une telle requête permet de filtrer , par exemple, les sources universitaires.

2. “SELECT * FROM Revue
WHERE Date_d_edition BETWEEN 1939 AND 1960
AND sujet = 'Guerre mondiale (1914-1918) -- Aspect religieux';“

Cette requête était destinée à me permettre de filtrer toutes les revues publiées entre 1919 et 1960 portant sur le sujet des aspects religieux de la première guerre mondiale. Elle n'aboutit à aucun résultat, ce qui montre qu'une requête combinant plusieurs filtres donnant chacun des résultats fructueux individuellement peut aboutir à un résultat qui, lui, est infructueux.



3. Requête MongoDB:

"# Connexion MongoDB
mongo_client = pymongo.MongoClient("mongodb://localhost:27017/")
mongo_db = mongo_client["bibliotheque_mongo"]
notices_collection = mongo_db["Notices"]

# Requête pour trouver les revues contenant "ISSN" dans "Notice du catalogue"
query = { "Notice_catalogue": { "$regex": "ISSN", "$options": "i" } }
result = notices_collection.find(query)

# Affichage des résultats
print(" Revues contenant 'ISSN' dans la Notice du catalogue :")
for doc in result:
print(doc)"

Cette commande permet d'afficher toutes les revues qui contiennent un identifiant ISSN au sein du champ "Notice de catalogue" de ma base de donnée Mongo DB (données semi-structurées).

Un identifiant ISSN est un identifiant unique pour toutes les revues publiées. Le problème ici est que l'extraction des métadonnées n'a pas pu permettre d'obtenir un nombre suffisant d'entrées afin de remplir le champ "Notice_catalogue" de la base de données Mongo DB et donc d'exploiter éventuellement l'identifiant ISSN des revues.

Commandes de nettoyage de données (ex: TRUNCATE)

Ces commandes m'ont été utiles lors de l'insertion de données: il s'est avéré que j'ai commis des erreurs dans l'intitulé des colonnes de mon dataframe que j'ai appelées dans mon script, ce qui a aboutit à l'erreur suivante: KeyError: "Notice d'ensemble".

Après avoir renommé correctement la colonne concernée dans mon script, il m'a fallu vider toutes mes tables avant de relancer mon script python afin de ne pas insérer les données en double. J'ai procédé à cette manoeuvre ainsi:

- en désactivant la vérifications de clés étrangères via la commande: SET FOREIGN_KEY_CHECKS = 0;
- en vidant toutes les tables de leur contenu :
    - TRUNCATE TABLE Revue;
    - TRUNCATE TABLE Auteur;
    - TRUNCATE TABLE Sujet;
- en réactivant la vérification de clés étrangères: SET FOREIGN_KEY_CHECKS = 1;

Cette commande correspond au DELETE de CRUD.

Nettoyage et agrégation des données

Les données sont effectivement agrégées, nettoyées et normalisées en un seul jeu de données à l'issue de l'exécution du script.

- Traitement de données et Nettoyage de données
- Programmation
- Scripts de Nettoyage de Données
- Agrégation de Données
- Manipulation de Données Multi-sources
- Homogénéisation des Formats de Données
- Gestion des Erreurs et des Exceptions
- Tests et Validation des règles et processus de nettoyage

df.drop_duplicates

Il s'agit d'une étape indispensable à tout nettoyage de base de données afin de vérifier s'il y a des entrées en double (avec la commande `df[df.duplicated()]`) et, s'il y a des doublons, de les supprimer avec la commande `df.drop_duplicates(inplace=True)`.

print(df["INTITULÉ DE LA COLONNE"].value_counts())

J'effectue un print(df["INTITULÉ DE LA COLONNE"].value counts()) sur les colonnes a priori classifiantes.

L'analyse des résultats me permet de prendre les décisions suivantes pour chaque colonne: homogénéisation, classification, drop column (en dernier recours car perte d'information):

- print(df[""].value_counts()): me donne:
    - Valeur vide 4983

Je décide donc simplement de faire un `df.drop[""]`

- `print(df["Conservation numérique :"].value counts())` me donne :
    - `Conservation numérique :`
    - `4891`
    - `Bibliothèque nationale de France 92`
    - `Bibliothèque nationale de France; 22/09/2014 2`
    - `Bibliothèque nationale de France; 06/03/2017 2`
    - `Bibliothèque nationale de France; 16/02/2020 1`
    - `Bibliothèque nationale de France; 29/06/2020 1`
    - `Bibliothèque nationale de France; 09/08/2020 1`
    - `Bibliothèque nationale de France; 27/09/2020 1`
    - `Bibliothèque nationale de France; 30/05/2018 1`
    - `Bibliothèque nationale de France; 23/06/2019 1`
    - `Bibliothèque nationale de France; 07/02/2021 1`
    - `Bibliothèque nationale de France; 25/03/2019 1`
    - `Bibliothèque nationale de France; 30/05/2016 1`
    - `Bibliothèque nationale de France; 09/02/2020 1`
    - `Bibliothèque nationale de France; 18/04/2021 1`
    - `Bibliothèque nationale de France; 08/04/2021 1`

Je décide donc d'uniformiser la colonne en ne conservant que "Bibliothèque nationale de France" les entrées correspondant à des dates sont en effet les dates correspondant à l'entrée "Date de mise en ligne", colonne déjà présente dans le dataframe.
Il n'y aura donc pas de perte d'information en effectuant l'opération:

```
df["Conservation numérique :"] = df["Conservation numérique :"].apply(
    lambda x: "Bibliothèque nationale de France" if x != "" and x !=
"Bibliothèque nationale de France" else x)
```

- `print(df["Auteur :"].value counts())` me donne :

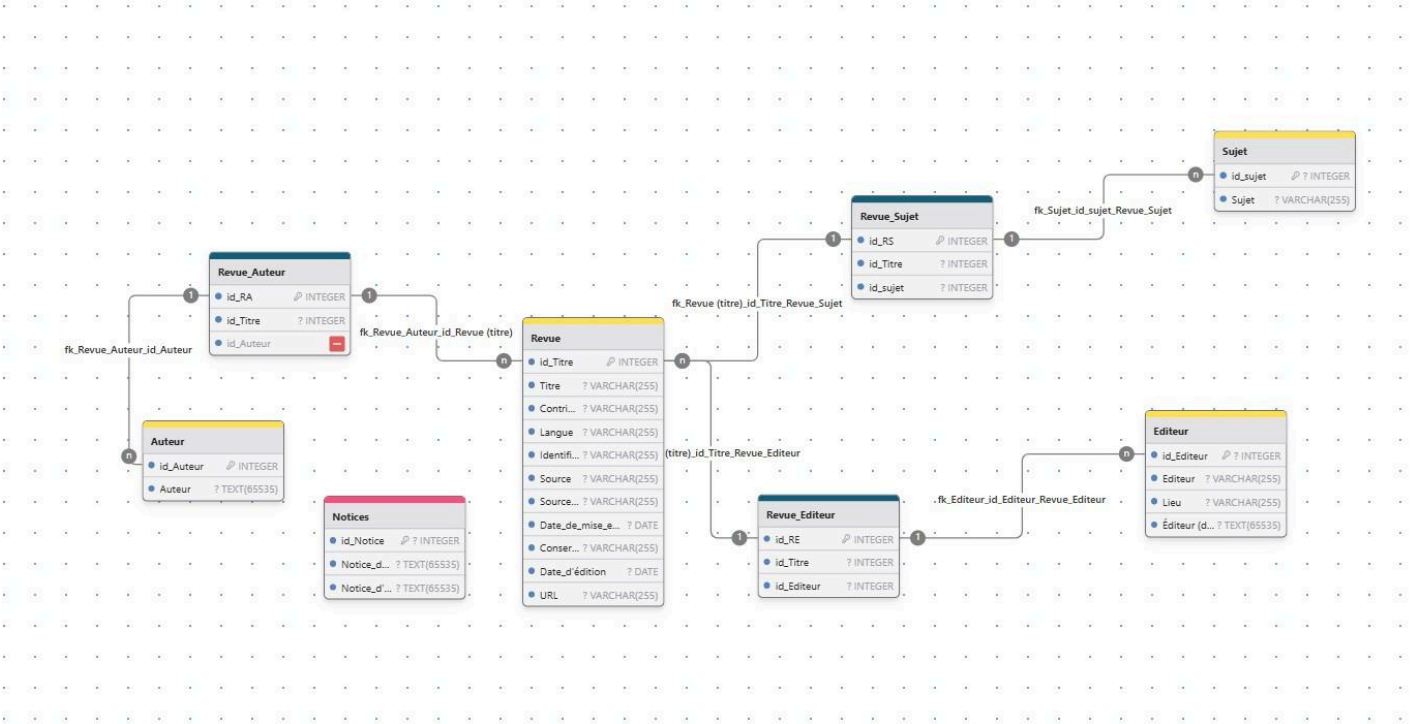
```
1  print(df["Auteur :"].value_counts())
✓ 0.0s
```

```
Auteur :

Sillon de la Drôme, de l'Isère et de la Loire. Auteur du texte Ne voir que les résultats de cet au
Bibliothèque royale de Belgique. Auteur du texte Ne voir que les résultats de cet auteur | Liens:
Comité national de l'enfance (France). Section (Rhône). Auteur du texte Ne voir que les résultats
Elsass-Lothringische wissenschaftliche Gesellschaft. Auteur du texte Ne voir que les résultats de

Fédération régionale des groupes d'études. Auteur du texte Ne voir que les résultats de cet auteur
Belle, Gabriel-Alexandre (1782-18..). Auteur du texte | Liens: https://gallica.bnf.fr/services/eng
Union fédérale des associations françaises de blessés, mutilés, anciens combattants de la Grande g
Amicale de l'École internationale des détectives (Paris). Auteur du texte Ne voir que les résultat
Église catholique. Diocèse (Montpellier). Auteur du texte Ne voir que les résultats de cet auteur
Name: count, Length: 1936, dtype: int64
```

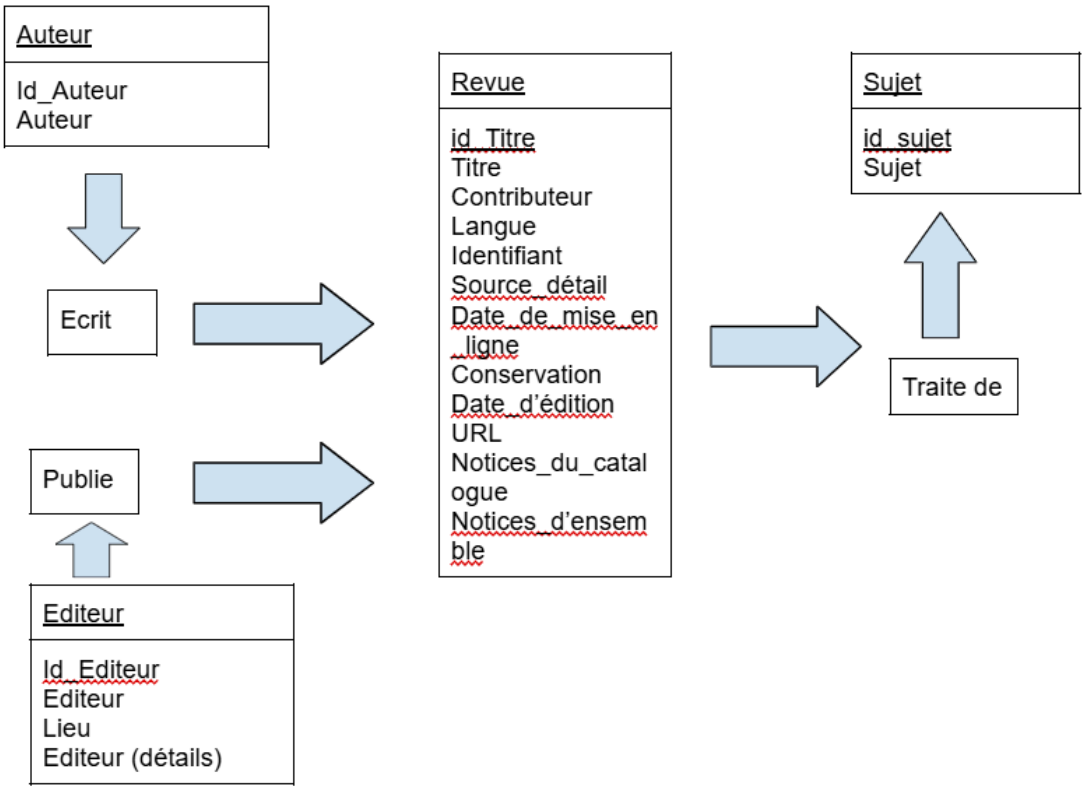
  - J'ai remarqué que l'entrée "Auteur" correspondait à chaque ligne au string à gauche de l'entrée ". Auteur du texte"
  - J'applique le code suivant afin d'extraire, pour chaque ligne, les données correspondantes et toutes les remplacer dans la colonne "Auteur":
    `df["Auteur :"] = df["Auteur :"].str.split(". Auteur du texte").str[0]`
- etc…
- `print(df["Relation :"].value_counts())`
  - J'ai remarqué que chaque ligne affichait une donnée chiffrée après "Première de couverture :", ce qui m'a amené à me demander si cela correspondait aux nombres de numéros parus pour chaque revue.
  - J'ai appliqué le code suivant pour étudier cette hypothèse:
    `df["Relation :"] = df["Relation :"].str.extract(r"Première de couverture : (\d+);").astype(float).astype("Int64")`
  - Puis en appliquant `df.sort_values(by=['Relation :'], ascending=False)`, j'ai pu m'apercevoir que les revues qui apparaissaient en haut de tableau étaient des revues peu connues.
    J'en ai déduit que le nombre après "Première de couverture :" ne correspondait pas à un nombre de revueexploitable numériquement dans le cadre de mon projet, mais à un identifiant que je n'ai pas su identifier.

Création de la base de données

Schémas de représentation des données



Modèle Physique de Données (MPD) réalisé avec DrawDB



Modèle Conceptuel de Données (MCD)

Explication de ma démarche

Parmi l'ensemble des données que j'ai structurées, après nettoyage, elles me semblaient toutes destinées à être contenues dans une table SQL, hormis peut-être les notices car les données contenues dans ces deux champs contenaient des descriptions bien plus longues qui s'apparentaient à de la donnée semi-structurée.
C'est donc pour cela que j'ai isolé ma table "notices" dans une collection MongoDB et que le reste a été établi sur la même BDD SQL.

Par ailleurs, j'espérais au départ pouvoir extraire plusieurs sujets ou plusieurs auteurs pour une seule et même revue à l'issue de mon nettoyage et de mon extraction de données. C'est pour cela que je me suis décidé à établir des tables de jonction basées sur une **relation many-to-many entre ma table "Revue" et toutes les autres** de ma BDD SQL. Il s'est avéré que **l'extraction de métadonnées n'a pas pu aboutir à cela**, mais toute jonction de table dans le cadre d'une relation many-to-many est de toute façon exploitable dans le cadre d'une relation one-to-many.

Développement de l’API mettant à disposition le jeu de données

Création d’un fichier .env

Il s’agit d’un fichier qui contient les variables d'environnement nécessaires pour lancer l’API tout en protégeant des données confidentielles sur un fichier distinct. J’ai créé un fichier nommé .env dans le même répertoire que mon fichier .py contenant le script d'exécution de mon API.

Exécution de l’API

Mon script met en place une API web avec Flask qui permet d’interagir avec ma base de données MySQL. Il commence par charger les variables d’environnement de mon fichier .env grâce à dotenv, ce qui sécurise les informations sensibles comme le nom d’utilisateur et le mot de passe de la base de données. Une fois ces données récupérées, il les utilise pour établir la connexion à ma base de données SQL ("Revues_numerisees"). La fonction get_db_connection() est définie pour créer cette connexion de manière réutilisable tout en gérant d’éventuelles erreurs.

L’API expose une **seule route "/revues**", accessible via une **requête GET**. Cette route permet de récupérer les entrées de la table Revue, en filtrant les résultats selon une **plage de dates définie par date_start et date_end**. Si ces paramètres sont fournis, une requête SQL est construite dynamiquement pour **afficher les revues publiées entre ces deux dates.** Le résultat est récupéré sous forme de dictionnaire et retourné au client sous format JSON. En cas d'erreur, l'exception est capturée et imprimée.

Enfin, le script exécute l’application Flask en mode debug si le fichier est lancé directement. Cela signifie que les erreurs seront affichées en temps réel dans la console, ce qui facilite le développement.