Institut universitaire de
Paris Rives de Seine

SAE NoSQL
Migration de données

Auteur :
Mehdi BENAYED
Bastien EBELY
Cheick GUEYE

10 novembre 2024

Partie 1

Introduction et définition de l'objectif finale

Dans ce projet, nous avons pour objectif de migrer une base de données d'un format relationnel SQLite vers un format NoSQL. Pour y parvenir, nous allons suivre une démarche structurée en quatre étapes :

1. **Création des requêtes SQL** pour extraire les données dont nous avons besoin. C'est essentiel pour avoir une bonne base d'informations.
2. **Définition du format des données** que nous voulons dans le système NoSQL et élaborer l'algorithme nécessaire pour cette migration.
3. **Écrire un script Python** qui facilitera ce transfert de données. On veut que ce soit fluide et sans accrocs permettant le passage de SQLite à NoSQL.
4. **Vérification de la migration** en formulant des requêtes dans le nouveau format NoSQL, pour s'assurer que tout est en ordre.

Nous commencerons par analyser les données pour bien comprendre ce dont nous avons besoin, puis nous choisirons le type de base de données NoSQL le plus adapté, qu'il s'agisse d'une base clé-valeur, de documents ou de graphes.

Ce processus nous permettra de garantir une transition réussie vers le modèle NoSQL tout en respectant les spécificités de nos données.

Partie 2

Migration vers une base de données NoSQL

2.1 Description des jeux de données

Les données proviennent de la base de données **SQLite ClassicModel**, qui stocke des informations de gestion des commandes. Notre groupe a proposé **deux modèles** différents pour la migration vers **MongoDB**, avec pour objectif de limiter le nombre de collections tout en optimisant l'organisation des données. Bien que nous soyons encore en discussion sur le choix final, MongoDB semble être l'option privilégiée. Voici les collections que nous envisageons de créer dans la nouvelle base NoSQL pour garantir la clarté et l'efficacité des structures de données.

2.2 Modèle 1 : Structure imbriquée

2.2.1 Orders

Cette collection regroupe toutes les informations relatives aux commandes passées par les clients, ainsi que les détails sur les produits et les informations client. Chaque document contient :

- OrderDetails : détails des produits commandés (quantité, prix, etc.).
- Produits imbriqués : informations spécifiques aux produits (code produit, nom, stock, etc.).
- Customers imbriqué : informations du client ayant passé la commande (nom, contact, adresse).
- Paiements imbriqués dans Customers : les paiements associés aux clients (montant, date, numéro de chèque).
- Détails de la commande : numéro de commande, date de livraison, statut de la commande, etc.

Ce modèle permet de regrouper toutes les informations pertinentes dans un seul document, simplifiant l'accès et la gestion des données relatives à une commande.

2.2.2 Employees

Cette collection contient toutes les informations sur les employés, avec un sous-document pour les bureaux où ils travaillent. Chaque document contient :

- Détails employés : nom, prénom, extension, email, etc.
- Offices imbriqué : informations du bureau associé à l'employé (ville, téléphone, adresse).

Nous avons choisi le modèle 1 afin de regrouper toutes les informations relatives à une commande dans un seul document. Cela permet de simplifier l'accès et la gestion des données en évitant des requêtes multiples entre plusieurs collections. En un seul appel, on peut accéder aux détails de la commande, aux produits associés, aux informations client, ainsi qu'aux paiements, ce qui est particulièrement avantageux pour des opérations de lecture fréquentes. Nous avons aussi réfléchi un deuxième modèle semi-séparée avec 3 tables : « Orders », « Customers » et « Employees », mais nous avons décidé de garder le modèle 1.

2.3 Définition des requêtes à utiliser

Après analyse des données, le type de base de données NoSQL choisi est **MongoDB**, une base orientée documents. MongoDB est très adaptée aux besoins de ce projet en raison de sa capacité à gérer des données semi-structurées et à modéliser les relations entre entités via des documents imbriqués et des références entre collections. Il est souple, sur les schémas, il est plus optimisable et permet d'imbriquer les tables les unes sur les autres pour faire moins de requête et il nous permet de faire des tests plus facilement sur Jupyter Notebook.

2.4 Difficultés rencontrées

Dans le cadre de l'étape de migration des données de SQLite vers MongoDB, nous avons rencontré plusieurs défis, notamment l'établissement de connexions à MongoDB avec des URI correctes et la gestion des autorisations d'accès. Nous avons dû veiller à supprimer les documents existants dans les collections pour éviter les doublons, nécessitant une attention particulière pour s'assurer que toutes les données étaient correctement effacées avant l'importation, donc nous avons utilisé à l'état initial ***.delete_many({})***.

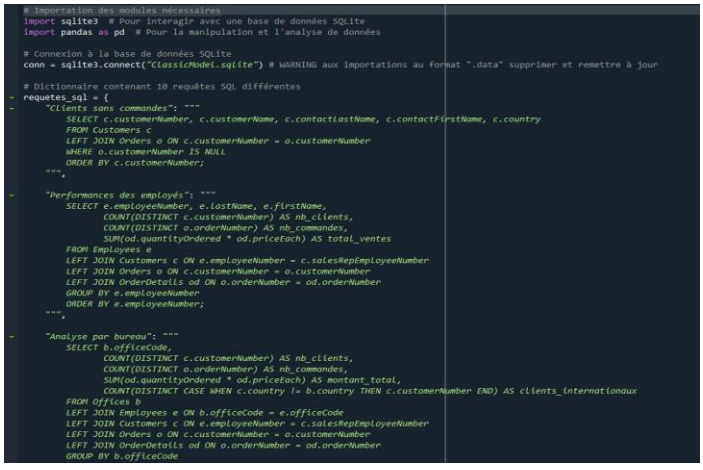
De plus, la transformation des données de format relationnel à un format document a exigé des ajustements minutieux, en particulier lors de la jointure des tables et de la structuration des données en listes. Enfin, la manipulation des données dans MongoDB, notamment avec l'utilisation de $lookup pour joindre des collections, a demandé des efforts supplémentaires pour garantir la cohérence et l'intégrité des informations. Ces défis ont mis en évidence la nécessité d'une collaboration efficace au sein de notre groupe afin d'assurer une migration réussie.

Partie 3

Les requêtes

3.1 Comment nous avons exécuter les requêtes

Voici les requêtes SQL qui nous permettront de vérifier si la migration vers MongoDB s'est déroulée correctement. L'objectif est de comparer les résultats des requêtes avant et après migration pour s'assurer que les données et leurs relations sont fidèles.



```
# Importation des modules nécessaires
import sqlite3  # Pour interagir avec une base de données SQLite
import pandas as pd  # Pour la manipulation et l'analyse de données

# Connexion à la base de données SQLite
conn = sqlite3.connect("ClassicModel.sqlite") # WARNING aux importations au format ".data" supprimer et remettre à jour

# Dictionnaire contenant 10 requêtes SQL différentes
requetes_sql = {
    "Clients sans commandes": """
        SELECT c.customerNumber, c.customerName, c.contactLastName, c.contactFirstName, c.country
        FROM Customers c
        LEFT JOIN Orders o ON c.customerNumber = o.customerNumber
        WHERE o.customerNumber IS NULL
        ORDER BY c.customerNumber;
    """,

    "Performances des employés": """
        SELECT e.employeeNumber, e.lastName, e.firstName,
               COUNT(DISTINCT c.customerNumber) AS nb_clients,
               COUNT(DISTINCT o.orderNumber) AS nb_commandes,
               SUM(od.quantityOrdered * od.priceEach) AS total_ventes
        FROM Employees e
        LEFT JOIN Customers c ON e.employeeNumber = c.salesRepEmployeeNumber
        LEFT JOIN Orders o ON c.customerNumber = o.customerNumber
        LEFT JOIN OrderDetails od ON o.orderNumber = od.orderNumber
        GROUP BY e.employeeNumber
        ORDER BY e.employeeNumber;
    """,

    "Analyse par bureau": """
        SELECT b.officeCode,
               COUNT(DISTINCT c.customerNumber) AS nb_clients,
               COUNT(DISTINCT o.orderNumber) AS nb_commandes,
               SUM(od.quantityOrdered * od.priceEach) AS montant_total,
               COUNT(DISTINCT CASE WHEN c.country != b.country THEN c.customerNumber END) AS clients_internationaux
        FROM Offices b
        LEFT JOIN Employees e ON b.officeCode = e.officeCode
        LEFT JOIN Customers c ON e.employeeNumber = c.salesRepEmployeeNumber
        LEFT JOIN Orders o ON c.customerNumber = o.customerNumber
        LEFT JOIN OrderDetails od ON o.orderNumber = od.orderNumber
        GROUP BY b.officeCode
```

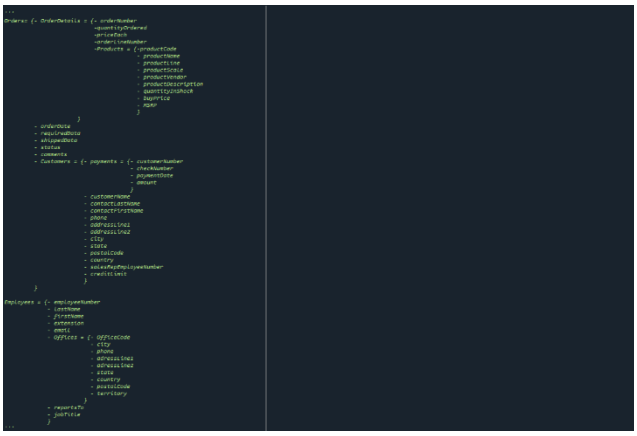
Sortie 1 *– extrait de validation des données par requêtes*

Index	customerNumber	total_achats	total_paiements
0	114	200995	195365
1	119	180125	136340
2	124	654858	647596
3	141	912294	793051
4	145	145042	119029
5	148	172990	172990

Sortie 2 *– extrait du dataframe*

Key	Type	Size	Value
Analyse par bureau	str	704	SELECT b.officeCode,
Clients sans commandes	str	268	SELECT c.customerNumber, c.customerName, c.contac…
Commandes par produit	str	449	SELECT p.productCode, p.productName,
Les Clients sont effectivement en retard de paiement	str	812	WITH AchatsTotaux AS (
Performances des employés	str	563	SELECT e.employeeNumber, e.lastName, e.firstName,…
Produits vendus à perte	str	300	SELECT p.productCode, p.productName, o.customerNu…
Tables de Contingence des commandes en fonction pays du client	str	399	SELECT p.productLine, c.country,
Tables de Contingence des commandes sur les produits achetés et le pays du client	str	409	SELECT p.productLine, c.country,
Top 10 produits à forte marge	str	286	SELECT p.productCode, p.productName,
Ventes par pays	str	502	SELECT c.country,

Sortie 3 *– extrait des requête sql*



Sortie 4*– schéma ciblé des données NoSQL choisie*

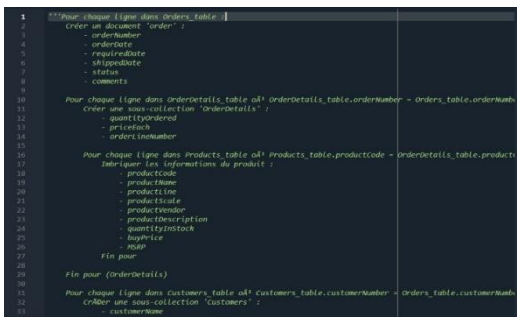
Partie 4

Schéma ciblé et conception du Pseudo-algorithme

L'algorithme parcourt **Orders_table** pour créer un document de commande contenant des informations comme le numéro de commande et les dates. Pour chaque commande, il rassemble les détails des articles dans une sous-collection **OrderDetails**, incluant la quantité et le prix, et imbrique des informations sur les produits depuis **Products_table**.

Ensuite, il recherche les informations client dans **Customers_table** pour former une sous-collection **Customers** contenant les détails du client, puis extrait les paiements associés de **Payments_table** pour créer une sous-collection **payments**. Après avoir structuré ces données, le document de commande est ajouté à la collection NoSQL **Orders**.

Le processus est similaire pour **Employees_table**, où chaque employé est créé avec ses détails, accompagnés des informations de bureau, et ajouté à la collection NoSQL **Employees**.



Sortie 5 *– extrait de code pour le Pseudo algorithme*