IUT de Paris - Rives de Seine
Université Paris Cité

Compte rendu SAE NO SQL

2024

Thomas BELAID
Mohammed BOUKOUIREN

Rendu: 10 Novembre 2024

SOMMAIRE

CONTEXTE

Dans le cadre de l'amélioration des performances et de la fiabilité du système d'information de son entreprise, Paula Dupont, directrice d'une entreprise spécialisée dans la production et la vente de véhicules, a pris la décision de migrer la base de données existante vers une solution NoSQL. En effet, la base de données relationnelle actuellement en place présente des limitations croissantes, notamment des temps de réponse élevés lors de certaines requêtes et une perte de données due à des pannes de serveurs, qui se multiplient avec le temps. Ces dysfonctionnements impactent directement les activités de l'entreprise et compromettent la fluidité des opérations. Afin de résoudre ces problématiques, Mme Dupont nous a confié la mission de mener à bien cette migration. À cette fin, elle a mis à disposition le jeu de données "ClassicModel", qui servira de point de départ pour l'adaptation et la mise en œuvre de la nouvelle base NoSQL.

A notre disposition :
La base de donnée ClassicModel sous le format sqlite.

Cette base de données relationnelle est composée de plusieurs tables interconnectées :

1. Orders (Commandes) : Contient les informations sur les commandes passées, liées aux Clients via customerNumber.
2. OrderDetails : Détaille chaque commande en se référant aux Produits via productCode.
3. Customers (Clients) : Regroupe les informations sur les clients, et est reliée aux Payments (Paiements) et aux Employés via salesRepEmployeeNumber.
4. Products (Produits) : Contient des détails sur les produits.
5. Employees (Employés) : Enregistre les informations des employés, associés à un Office (bureau) via officeCode.
6. Offices (Bureaux) : Stocke les données des bureaux de l'entreprise.
7. Payments (Paiements) : Liste les paiements effectués par les clients.

Les relations principales relient les commandes, les clients, les employés, et les produits.

METHODOLOGIE

La méthodologie que nous adoptons pour cette démarche de migration repose sur plusieurs étapes clés. Dans un premier temps, nous créons des requêtes SQL sur la base de données relationnelle initiale afin d'extraire les données nécessaires. Ensuite, nous réfléchissons au format des données à obtenir dans la base NoSQL, ainsi qu'à l'algorithme permettant de structurer efficacement ces données. Une fois cette phase préparatoire achevée, nous rédigeons un script Python pour automatiser la conversion des données de SQLite vers NoSQL. Enfin, la dernière étape consiste à valider la migration, en nous assurant que toutes les données ont été correctement transférées et que le système fonctionne comme prévu.

Etapes :

1. Création de requêtes SQL sur la BD initiale ;
2. Réflexion sur le format des données à obtenir et l'algorithme à réaliser ;
3. Écriture du script Python permettant le passage de SQLite à NoSQL ;
4. S'assurer que la migration s'est bien passée.

DIFFICULTÉS RENCONTRES

Réflexion sur le format des données à obtenir et l'algorithme à réaliser :

- L'une des premières difficultés consiste à adapter la structure des données relationnelles (tables, relations, clés étrangères) à un modèle NoSQL.
- Afin que la migration se fasse dans les meilleurs conditions nous savons du reflechir sur quel types de base de donnée nous pourrions uitiliser dans notre contexte. Ce qui a été un choix évident à faire car pour chaque type nous avions des points possitifs et des pointfs negatifs
- Gestion des transactions : Dans une base relationnelle, la gestion des transactions (ACID) est intégrée. En NoSQL, il faut réfléchir à des alternatives pour garantir la cohérence des données.

CRÉATION DE REQUÊTES SQL SUR LA BD INITIALE

Lors de cette étape, nous avons dû répondre à plusieurs questions qui ont orienté nos réflexions sur la structure des données et l'algorithme de migration, tout en nous assurant que la migration serait réalisée sans perte de données. Afin de valider cette intégrité des données, nous avons mis en place un script Python en utilisant les bibliothèques pandas et sqlite3. Ces outils nous ont permis de formuler et d'exécuter des requêtes SQL directement sur la base de données SQLite avant migration.

Voici un exemple de requête SQL que nous avons utilisée pour répondre à l'une des questions clés et vérifier la cohérence des données :

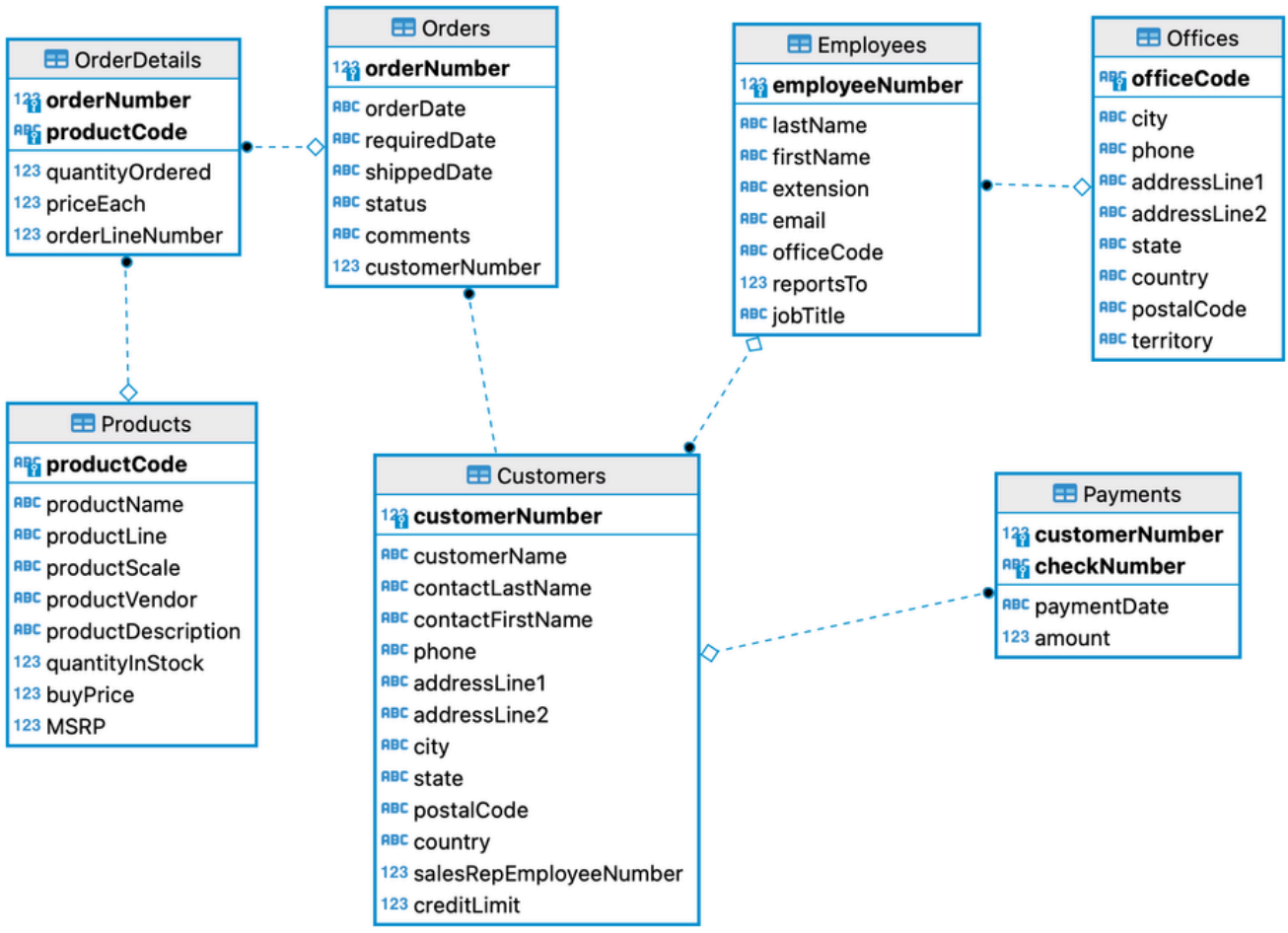
(Exemple de requetes qui réponds à une question)
query = '''
  SELECT c.customerNumber, c.customerName
  FROM Customers c
  LEFT JOIN Orders o ON c.customerNumber = o.customerNumber
  WHERE o.customerNumber IS NULL;
'''

Cette requête, écrite en langage SQL, permet d'identifier les clients n'ayant jamais passé de commande en effectuant une jointure externe gauche (LEFT JOIN) entre les tables Customers et Orders. L'objectif de cette vérification est de s'assurer que toutes les données de la table Customers sont correctement reliées à leurs commandes correspondantes, ou de repérer les clients orphelins (sans commande associée).

RÉFLEXION SUR LE FORMAT DES DONNÉES À OBTENIR ET L'ALGORITHME À RÉALISER

Comme mentionné précédemment, le choix du type de base de données NoSQL est une étape cruciale dans le processus de migration. Il s'agit de sélectionner un modèle de données qui répond à la fois aux besoins de l'entreprise et aux exigences techniques.
Le type de base de données NoSQL choisi impactera directement la manière dont les données seront stockées, indexées et récupérées.

Etat de la base avant Migration :



Nous avons choisi d'utiliser une base de données NoSQL au format document, car elle s'avère être la solution la plus adaptée à nos besoins. Ce type de base de données permet une flexibilité et une évolutivité accrues, facilitant la gestion de données non structurées et semi-structurées. De plus, il offre une performance optimale pour les opérations de lecture et d'écriture, ce qui est essentiel pour notre application. En permettant un schéma dynamique, nous pouvons facilement adapter la structure des données au fur et à mesure de l'évolution de nos exigences, ce qui nous permet de gagner en agilité et en réactivité.

Afin de réaliser au mieux la migration nous devons “restructuré” notre base de données

Pour cela nous avons décidé de reformer la base en 4 collections:

Customer qui comporte :

- customerNumber
- customerName
- contactLastName
- contactFirstName
- phone
- addressLine1
- addressLine2
- city
- state
- postalCode
- country
- salesRepEmployeeNumber
- creditLimit
- **Type_Payment** (sous dictionnaire qui comporte les variables)
    - Payement
    - Amount

Order qui comporte :

- orderNumber
- orderDate
- requiredDate
- shippedDate
- status
- comments
- customerNumber
- **Order_Detail (sous distionnaire qui comporte les variables)**
  - QuantityOrdered
  - PriceEach
  - OrderLineNumber

Product qui comporte :

- ProductName
- ProductLine
- ProductScale
- ProductVendor
- QuantityInstock
- buyPrice

Employee qui comporte :

- employeenumber
- LastName
- FirstName
- Extension
- Email
- OfficeCode
- reportsTO
- JobTitle
- **Offices (sous distionnaire qui comporte les variables)**
  - city
  - phon
  - state
  - country
  - postalCode
  - territory

ÉCRITURE DU SCRIPT PYTHON PERMETTANT LE PASSAGE DE SQLITE À NOSQL ;

Afin de pouvoir faire la migration nous avons du crée les trois différentes collections via Jupiter note book en utilisant du python et les librairies/ Packages sqlite3,panda et python mais surtout crée un cluster via MongoDB Compass et Atlas afin d'avoir les information sur ce que contient notre table et faire des vérifications .

Il y a plusieurs étapes pour chauqe collections nous avons poceder plus ou moints de la même façon.

Etape 1 : Impot des librairties, connection à notre cluster et connection à la base SQLite

```
import sqlite3
import pandas
import

# Connection à notre cluster

URI = "mongodb+srv://user_mongo:c4x03xqwLARoqlPK@sae.bkqxr.mongodb.net/?retryWrites=true&w=majority&appName=SAE"
client = pymongo.MongoClient(URI)
db = client.sae

# Création de la connexion avec la base Classic model
conn = sqlite3.connect("ClassicModel.sqlite")
```

Etape 2 : Selectioner des tables qu'on va utiliser pour crée la collection
Dans notre exemple (

```
# Récupération du contenu de Customers avec une requête SQL
customers = pandas.read_sql_query("SELECT * FROM Customers;", conn)
customers
```

ÉCRITURE DU SCRIPT PYTHON PERMETTANT LE PASSAGE DE SQLITE À NOSQL ;

Etape 3 : Création du sous dictionnaire,

```
Transac = [
    payments.query('customerNumber == @id')
        .drop(columns=["customerNumber", "customerNumber"])
        .to_dict(orient = "records")
        for id in customers.customerNumber
]
```

Etape 4 : ajout de la colonne dans la table dédiée exemple ici les paymements dans la table customers

```
customers = customers.assign(Transaction = Transac)
customers.head()
```

Etape 5 : insertion des données dans la collection qu'on vient de crée en même temps. Ici on insert dans la collection Customer

```
db.Customers.insert_many(
    customers.to_dict(orient = "records")
)
```

On a donc refait plus ou moins les mêmes etapes pour les deux autres collections : Employees et Orders

VALIDATION DE LA MIGRATION

Afin de pouvoir voir si les données ont bien été migrer il a fallu faire des requetes via pymongo. Comme nous avions fait des requetes Sql sur la table de base on a pû faire la comparaison avec quelques requetes.

Exemple première requete :

Version SQL :

```sql
# 1 ; Lister les clients n'ayant jamais effecuté une commande

query = '''
    SELECT c.customerNumber, c.customerName
    FROM Customers c
    LEFT JOIN Orders o ON c.customerNumber = o.customerNumber
    WHERE o.customerNumber IS NULL;
'''

df = pd.read_sql_query(query, conn)

print(df)
```

Version pymongo

```python
# Requete 1

# Pipeline d'agrégation pour lister les clients sans commandes
# Exécution de la requête
result = list(db.Customers.aggregate([
    {
        "$lookup": {
            "from": "orders",
            "localField": "customerNumber",
            "foreignField": "customerNumber",
            "as": "orders"
        }
    },
    {"$match": {"orders": {"$size": 0}}},
    {"$project": {"customerNumber": 1, "customerName": 1}},
    {"$sort": {"customerName": 1}}
]))

# Création du DataFrame avec les résultats
df = pd.DataFrame(result)

# Affichage du DataFrame
print(df)
```

VALIDATION DE LA MIGRATION

Resultat :

	_id	customerNumber	customerName
0	67161b75395f3172cdd2ff95	237	ANG Resellers
1	67161b75395f3172cdd2ff7d	168	American Souvenirs Inc
2	67161b75395f3172cdd2ffd6	465	Anton Designs, Ltd.
3	67161b75395f3172cdd2ff8d	206	Asian Shopping Network, Co
4	67161b75395f3172cdd2ffb7	348	Asian Treasures, Inc.
5	67161b75395f3172cdd2ffa4	293	BG&E Collectables
6	67161b75395f3172cdd2ffb3	335	Cramer Spezialitäten, Ltd
7	67161b75395f3172cdd2ffa8	307	Der Hund Imports
8	67161b75395f3172cdd2ffcc	443	Feuer Online Stores, Inc
9	67161b75395f3172cdd2ff9f	273	Franken Gifts, Co
10	67161b75395f3172cdd2ff6f	125	Havel & Zbyszek Co
11	67161b75395f3172cdd2ffbc	361	Kommission Auto
12	67161b75395f3172cdd2ffdb	480	Kremlin Collectables, Co.
13	67161b75395f3172cdd2ffbf	369	Lisboa Souveniers, Inc
14	67161b75395f3172cdd2ff99	247	Messner Shopping Network
15	67161b75395f3172cdd2ffda	477	Mit Vergnügen & Co.
16	67161b75395f3172cdd2ff92	223	Natürlich Autos
17	67161b75395f3172cdd2ff7e	169	Porto Imports Co.
18	67161b75395f3172cdd2ffc0	376	Precious Collectables
19	67161b75395f3172cdd2ffdc	481	Raanan Stores, Inc
20	67161b75395f3172cdd2ffba	356	SAR Distributors, Co
21	67161b75395f3172cdd2ffa7	303	Schuyler Imports
22	67161b75395f3172cdd2ffc8	409	Stuttgart Collectable Exchange
23	67161b75395f3172cdd2ffd4	459	Warburg Exchange

Nous pouvons voir que le resultat de la requête est le même autant en pymongo apres migration que en SQL.

Conclusion :

Nous avons observé que le résultat de la requête reste identique, que ce soit en utilisant PyMongo après migration ou en SQL. Grâce aux nombreuses étapes réalisées, incluant la conception, la réflexion approfondie, ainsi que les vérifications minutieuses, nous pouvons conclure que la migration a été réussie. Le processus mis en place a permis d'assurer la cohérence et la validité des données migrées, garantissant ainsi une transition fluide vers le nouveau système.
De plus, le choix d'une base NoSQL orientée documents s'est avéré judicieux pour notre cas, car il répond aux besoins de flexibilité et d'évolutivité nécessaires pour notre application.