Eva BERTRAND, Lorynda LOUFOUA
BUT3 SD-VCOD, Groupe 33

IUT de Paris - Rives de Seine
Université Paris Cité

---

*Situation d'Apprentissage et d'Evaluation*

# Migration d'une base de données relationnelle en NoSQL

*24 Novembre 2024*

---

## *Contexte :*

Nous sommes chargées d'opérer la migration d'une base de données relationnelle vers une base de données NoSQL pour une entreprise de voiture.

En effet, des limites commencent à se faire présentes sur la base de données actuelle, comme des pertes de données dues à des défaillances serveurs, des temps de requêtes plus long, etc.

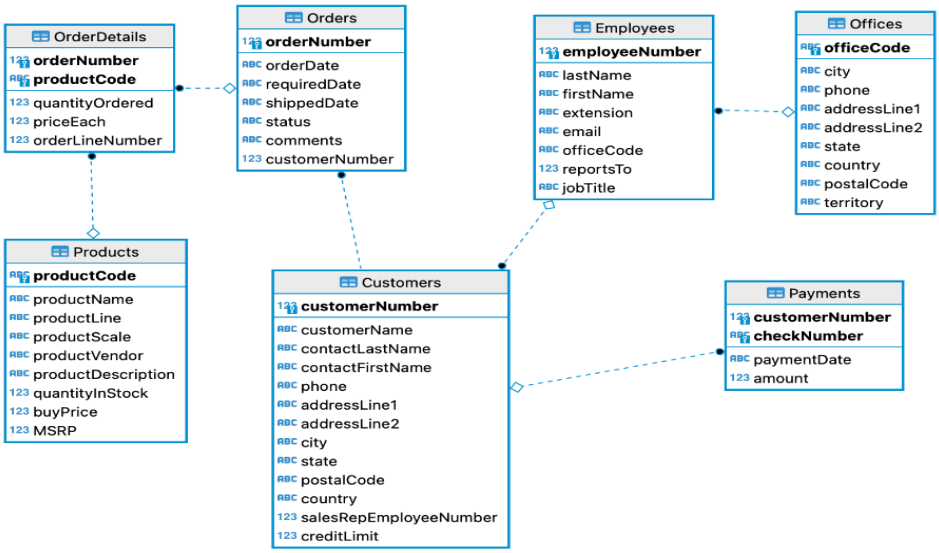
C'est une tâche complexe qui nécessite une planification minutieuse et une exécution rigoureuse.

Ce rapport détaille les étapes majeures de la migration, de l'analyse initiale à la validation des données migrées :

1. Création de requêtes SQL sur la base de données relationnelle.
2. Définition du schéma cible et du type de base NoSQL adapté.
3. Développement du script Python pour effectuer la migration.
4. Création et vérification des requêtes dans le nouveau format NoSQL.
5. Analyse des défis rencontrés et des solutions apportées.

## *Présentation des données d'origine :*

Pour ce projet, nous avons à disposition la base de données *ClassicModel*, une base contenant des données de Ventes, composées de 7 tables.




## *1 - Requêtes SQL*

La première étape consiste à créer des requêtes SQL sur la base de données initiale ClassicModel.

Ces requêtes permettent de comprendre la structure des données et d'extraire les informations nécessaires pour la migration.

Nous les comparerons plus tard avec nos sorties de requêtes NoSQL.

Pour se connecter à la base de données SQLite, nous utilisons le module **Sqlite3** en Python.

```python
# Importation des modules utilisés
import sqlite3
import pandas

# Création de la connexion
conn = sqlite3.connect("ClassicModel.sqlite")

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

# Fermeture de la connexion : IMPORTANT à faire dans un cadre professionnel
#conn.close()

###  ###
#1 - Lister les clients n'ayant jamais effecuté une commande
q1 = pandas.read_sql_query(
    """
    SELECT c.customerName, c.customerNumber
    FROM Customers c
    LEFT JOIN Orders o ON c.customerNumber = o.customerNumber
    Where o.orderNumber IS NULL;
    """,
    conn
)
print(q1)
print()
```

## *2 - Réflexion format*

L'objectif de cette étape se sépare en 2 tâches :

- Réflexion du type de base de données NoSQL à utiliser (colonne, document, clé ou graphe)
- Création du schéma cible des données dans la base de données NoSQL correspond au choix précédent

### 2.1 - Type de base de données

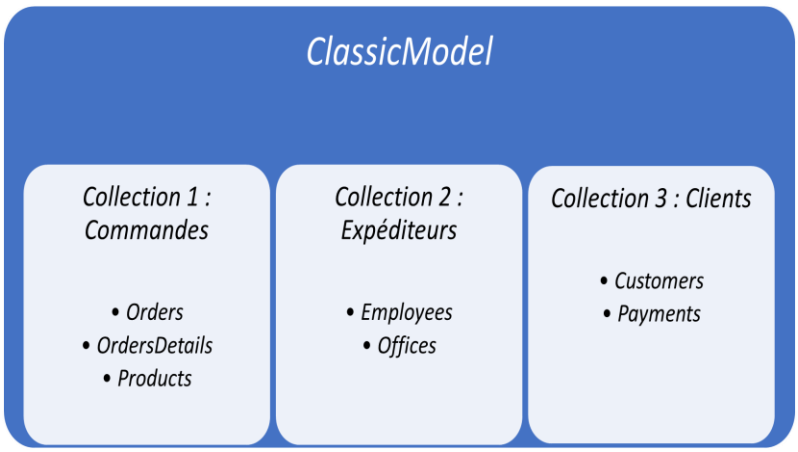
Nous avons choisi un modèle de données de type document car ce modèle est plus flexible que le modèle relationnel, chaque document peut avoir une structure différente.

Il y a la possibilité de stocker des informations de manière plus souple, sans nécessiter un schéma rigide, pour simplifier la gestion des données.

Notamment ici avec un modèle de base de données NoSQL comme MongoDB, il y a souvent une meilleure gestion des pannes et des sauvegardes.

Ce qui répond directement à l'une de nos problématiques majeures : la perte de données due à des défaillances serveur.

### 2.2 - Schéma cible du ClassicModel



## *3 - Écriture du script Python permettant le passage de SQLite à NoSQL*

### 3.1 - Pseudo-Algorithme

Pour établir la migration entre les données SQL à NoSQL, nous devons procéder à l'écriture du pseudo-algorithme.

Tout d'abord on importe sqlite3 et pymongo (MongoDB) pour se connecter à la base de données ClassicModel.

```
## Connexion à la base de données

import sqlite3
import pandas
import pymongo

URI = "mongodb+srv://user-mongo:-5D3ii.GbETV_qy@cluster-but-sae.8cviz.mongodb.net/?retryWrites=true&w=majority&appName=Clust
client = pymongo.MongoClient(URI)
db = client.NoSQL

# Création de la connexion
conn = sqlite3.connect("ClassicModel.sqlite")

```

On procède ensuite à la récupération des données en récupérant le contenu de chaque table de la base de données SQLite avec un (SELECT*)

### 3.2 Modèle de Données MongoDB (Architecture Cible)

Avec MongoDB les données sont stockées sous forme de documents JSON.

Chaque document peut avoir une structure différente, ce qui offre une grande flexibilité. On va alors créer les collections principales de la base de données.

Nous avons la collection :

- Clients afin de regrouper les paiements associés à chaque client
- Commandes où l'on regroupe les commandes ainsi que leurs détails et le détail des produits
- Expéditeurs afin d'associer chaque employé à son office

Nous transformons par la suite les données relationnelles en documents JSON et les insérons dans les collections créer.

Nous répétons cette opération pour toutes nos collections.

Une fois les données migrées, nous avons utilisé **MongoDB Compass** pour inspecter visuellement les collections créées et vérifier que les relations imbriquées correspondaient aux attentes.

## *4 - Création des requêtes initiales au nouveau format NoSQL*

A présent, après avoir réalisé la migration des données nous sommes alors en mesure de pouvoir restituer les mêmes résultats que les requêtes faites en SQL.

Nous convertissons les résultats en DataFrame pandas pour les afficher et les vérifier.

Voici un exemple de requête d'agrégation, ici on tente de lister les clients n'ayant jamais effectuer une commande :

```
## 1- Lister les clients n'ayant jamais effectué une commande ##

print('Question 1:')
no_comm = db.Clients.aggregate([
    { "$match": { "Payments": { "$eq": []}}},

    { "$project": {
            "_id": 0,
            "customerName": 1,
            "customerNumber": 1}},

    { "$sort": { "customerNumber": 1 } }  #tri par ordre croissant (customerNumber)
])

pandas.DataFrame(list(no_comm))
```

## *5 – Enjeux et difficultés de la mission*

Pour cette mission, les enjeux et les difficultés auxquels nous avons fait face sont :

- La connexion sur MongoDB Compass : Plusieurs erreurs de configuration ont nécessité des recherches pour corriger les paramètres de connexion.

- Durée de la migration : L'étape de la migration peut s'avérer être longue, surtout avec des volumes de données importants.

- Complexité des requêtes : Refaire les requêtes SQL en NoSQL peut être complexe en raison des différences de modèles de données et de syntaxe.
  Par exemple :
  Les jointures explicites en SQL (`JOIN`) nécessitent des opérations d'agrégation complexes en MongoDB (`$lookup`, `$unwind`).

- Vérification de l'intégrité des données : Assurer que toutes les données ont été correctement migrées et qu'aucune information n'a été perdue ou corrompue.

- Adaptation aux nouveaux modèles de données : Passer d'un modèle relationnel à un modèle de documents nécessite une compréhension approfondie des deux systèmes et une planification minutieuse.