Université Paris Cité

BUT SD FA EMS - 2025

SAE

Migration de données vers un environnement NoSQL

NoSQL

Réalisé par

Rachid SAHLI, Ayoub ERRAHMANI

Réalisé en

Novembre - Decembre 2024

Table des matières

1 Introduction

L'objectif de la présente situation d'évaluation et d'apprentissage est de migrer des données issues d'une base de donnée relationnelle (SQL) vers une base au format NoSQL.

Les bases de données relationnelles permettent de stocker des données structurées dans des relations, sous forme de tables. Ces tables sont organisées en ligne et en colonnes, où chaque ligne représente un enregistrement et chaque colonne un attribut de cet enregistrement. Le langage SQL (Structured Query Language) permet d'interagir avec ces bases de données. Il permet d'effectuer diverses opérations. Par exemple, la création de nouvelles tables, la mise à jour, l'insertion ou la suppression de données. Mais aussi de l'analyse de données.

Les bases de données NoSQL (Not Only SQL) ne sont pas des bases de données relationnelles, mais peuvent compléter ces dernières. Elles peuvent prendre diverses formes selon le besoin spécifique, mais permettent toutes de stocker des données sous une forme non structurée. Il est possible d'interagir avec ces bases sans utiliser de langages de requête complexe.

De manière plus concrète, on imagine ici, Paula DUPONT, directrice d'une entreprise de voitures. Elle souhaite migrer sa base de donnée relationnelle vers une base de données NoSQL. En effet, la base de données actuelle commence à montrer ses limites : les requêtes présentent une forte latence, des pertes de données surviennent en raison de défaillances serveur de plus en plus fréquentes, entre autres problèmes. Notre but est donc d'opérer la migration vers une base de données NoSQL.

Nous procéderons selon les quatre étapes suivantes. Dans un premier temps, nous créerons des requêtes SQL sur la base de données initiale. Ensuite, nous définirons le format des données à obtenir et concevrons l'algorithme qui guidera le processus de migration. Une fois cette étape définie, nous développerons le programme chargé d'effectuer la migration des données. Enfin, pour évaluer la qualité de la migration, nous créerons de nouvelles requêtes sur la base de données NoSQL et analyserons les résultats obtenus.

2 Base de données relationnelle (SQL)

2.1 Présentation de la base de données

Nous disposons initialement de la base de donnée relationnelle de l'entreprise de voitures. Elle se nomme ClassicModel et contient 7 tables. Ces tables incluent des informations sur les clients,

les véhicules, les commandes, les employées... Cette base de données présente des problèmes de performance, ce qui peut nuire à l'efficacité opérationnelle de l'entreprise.

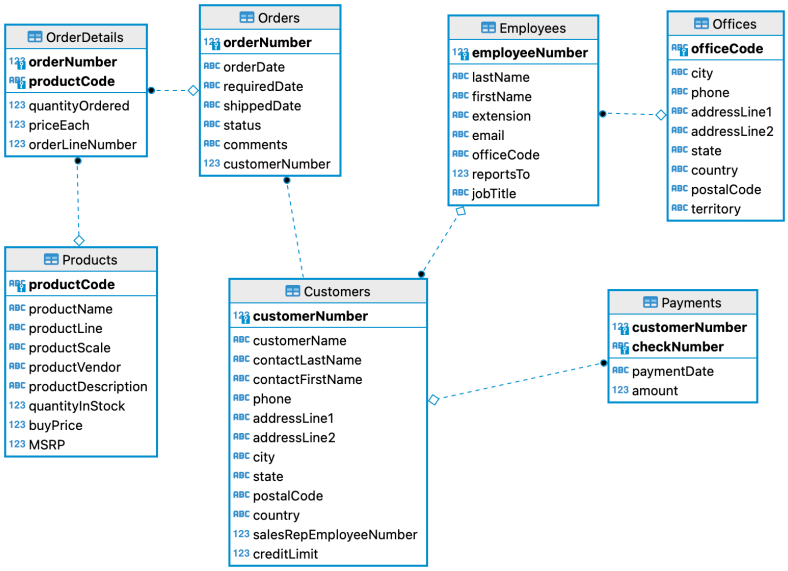

Figure 1: Modèle SQL

Le schéma relationnel ci-dessus illustre la structure de la base de données SQL. Bien que cette dernière présente certains problèmes de performance, la représentation des données reste relativement claire et organisée. Cette structure permet de facilement extraire des informations, notamment grâce à l'utilisation de jointures entre les tables.

2.2 Création de requête SQL

Nous avons dans un premier temps utiliser le modèle sqlite3 en Python pour créer la connexion à la base de données. Cela nous permet d'interagir avec notre base de données SQLite. Ce dernier est un système de gestion de bases de données relationnelle léger. Ensuite, nous utilisons la bibliothèque pandas en Python. Cette bibliothèque est très utile pour la manipulation et l'analyse de données. Ici, elle nous sert à exécuter nos requêtes et à récupérer le résultat dans un DataFrame.

Nous créons 10 requêtes SQL sur la base de données ClassicLite. Celles-ci sont contenues dans un fichier joint et serviront de tests pour évaluer la réussite (ou non) de la migration. En exécutant ces requêtes avant et après la migration, nous pourrons comparer les résultats obtenus dans la base de donnée relationnelle et la base NoSQL. Cela nous permettra de vérifier la cohérence, l'intégrité des données et s'assurer que la migration a été effectuée correctement, sans perte d'information. Cette

partie du travail nous a permis de mieux comprendre la structure de la base de donnée relationnelle ainsi que les informations qu'elles contenaient.

3 Définition du format des données et conception de l'algorithme de migration

Dans le cadre de notre projet et du contexte de l'entreprise automobile, l'utilisation d'une base de données NoSQL est possible car :

- L'entreprise dispose d'une grande quantité de données en constante croissance, qui sont structurées.
- L'adoption du NoSQL permettrait d'améliorer significativement les performances d'accès aux données, en optimisant le traitement de volumes de données plus importants, en réduisant les temps de latence et en augmentant le débit.

Il existe plusieurs types de base de données NoSQL présentant chacune des caractéristiques différentes, adaptées à des cas d'usages spécifiques. Il en existe 4 grandes familles (Clé-valeur, Colonne, Document et Graphe) . Ces quatre types de bases de données sont adaptées à différents cas d'usage, allant de la gestion simple de données à des analyses complexes de relations entre objets. Le choix du type de base de données NoSQL dépend donc de plusieurs facteurs clé, tels que la structure des données, les exigences de performance, la scalabilité et les cas d'utilisation spécifiques.

3.1 Choix du type de base de données NoSQL

Dans un premier temps, nous définissons la structure de nos données ainsi que les objectifs de la migration. Nous partons d'une base de donnée relationnelle, organisée sous forme de tables interconnectées par des relations, avec des clés primaires et étrangères, et un schéma rigide. L'objectif est de migrer ces données vers une solution NoSQL, qui offre plus de flexibilité et d'évolutivité, tout en préservant leur intégrité et en répondant aux exigences de performance et de scalabilité. En effet, l'une des limitations du modèle relationnel, en particulier, lorsqu'il s'agit de volumes de données croissants, est la difficulté à gérer efficacement des ensembles de données très volumineux. Par exemple, si le nombre de commandes de véhicules venait d'augmenter de manière exponentielle, la gestion de ces données dans une base relationnelles pourrait devenir complexe en raison des limitations en matière de scalabilité horizontale. Dans ce contexte, une base NoSQL, avec son

architecture distribuée, permettrait de mieux gérer cette croissance rapide tout en optimisant les performances. Nous prenons également en compte la nature de nos 10 requêtes, qui impliquent des agrégations et des jointures complexes.

Ensuite, nous comparons les différents formats de base de données NoSQL.

3.1.1 Format document

Le format document est bien adapté à notre base de données, car il permet de structurer naturellement les entités de manière hiérarchique. Par exemple, un client peut être représenté par un document contenant ses commandes, chaque commande incluant les produits associés. Ce format facilite également la gestion des données non structurées ou semi-structurées, comme les commentaires ou les statuts de commande. Il offre une grande flexibilité, permettant de traiter des données variables sans modification complexe du schéma. Enfin, il permet une scalabilité horizontale facile grâce à la partition des documents.

3.1.2 Format clé-valeur

Le modèle clé-valeur est très simple et rapide pour les recherches directes par clé. Cependant, il est limité pour les requêtes analytiques complexes, comme les agrégations ou les jointures. Par exemple, pour récupérer des informations clients, il faut connaître la clé spécifique, ce qui rend ce format inadapté à notre projet, où plusieurs relations entre entités doivent être gérées sans redondance. Ce modèle est donc trop restrictif pour répondre à nos besoins.

3.1.3 Format colonne

Le format colonne est efficace pour les écritures intensives et les agrégations simples, car il permet de traiter directement les colonnes concernées sans parcourir l'ensemble des données. Cependant, il montre des performances limitées pour les jointures complexes et les analyses nécessitant un accès global aux données. Dans notre cas, où les requêtes impliquent des jointures et des agrégations complexes, ce modèle est partiellement adapté mais insuffisant pour répondre à nos besoins.

3.1.4 Format graphe

Le format graphe est optimisé pour les relations complexes entre entités, comme celles des réseaux sociaux. Toutefois, dans notre cas, les relations sont simples (par exemple, client-commandes-

produits), et le modèle graphe nécessiterait de reconfigurer entièrement notre schéma de données, ce qui ajouterait complexité et redondance. Par conséquent, ce format n'est pas adapté à notre contexte.

Nous choisissons de migrer nos données vers le format document, qui est le mieux adapté à notre objectif. Il offrira à Paula Dupont une base de données flexible, bien adaptée à la gestion des données hiérarchiques et permettant une scalabilité horizontale grâce à la partition des documents.

3.2 Schéma cible des données

À présent, nous allons établir le schéma cible des données dans la base de données document. Ce schéma fait référence à la manière dont les données sont structurées et organisées dans le modèle document de la base de données NoSQL. Contrairement aux bases de données relationnelles où un schéma fixe (tables, colonnes, relations) est défini au préalable, le schéma dans une base de données document peut être plus flexible, en particulier si les documents peuvent avoir des structures variables. Cependant, il doit quand même être planifié pour assurer une gestion cohérente et efficace des données.

Nous avons décidé de structurer nos données autour de quatre collections principales : customers, payments, orders, et employees.

- Collection **Orders** : D'un point de vue métier, une commande et ses détails sont toujours traités ensemble, et il est essentiel de connaître les produits achetés, leurs quantités et leurs prix. Nous avons donc regroupé ces entités car elles suivent toutes un "processus d'achat". De plus, regrouper ces informations dans un seul document permet d'éviter les jointures complexes et d'améliorer les performances des requêtes, en accédant à toutes les données nécessaires en une seule lecture. Nous avons regrouper ces trois tables pour ces raisons.
- Collection **Employees** : Les informations relatives aux employés (nom, rôle, bureau, etc.) sont étroitement liées à celles contenues dans la table offices, car chaque employé est affecté à un bureau spécifique. En regroupant les entités employees et offices dans un même document, nous évitons les jointures complexes, ce qui améliore les performances des requêtes. Ainsi, lors de la récupération des données d'un employé, les informations sur son bureau sont immédiatement accessibles dans le même document, ce qui accélère l'accès et réduit le temps de réponse. De plus, si un employé change de bureau ou si de nouveaux bureaux sont ajoutés, ces mises à

jour peuvent être effectuées facilement sans perturber la structure globale des données.

- Collection *Customers* : Nous avons créé une collection customers contenant uniquement l'entité customers. Bien que chaque commande soit associée à un client, les informations relatives aux clients (nom, adresse, etc.) changent moins fréquemment que celles des commandes ou paiements. De plus, cette séparation permet d'optimiser les requêtes ciblant spécifiquement les données clients, sans avoir à inclure systématiquement les détails des commandes ou paiements.
- Collection *Payments* : Cette collection ne contient que l'entité payments. En gardant cette collection distincte, nous simplifions notre modèle et notre migration.

En effet, notre expériance étant limitée en MongoDB et modèles non relationnels nous avons préféré laisser les collections customers et payments indépendantes afin de simplifier la gestion des données et éviter des complications inutiles.

3.3 Pseudo-algorithme de migration

Avant de développer notre programme de migration des données, nous établissons un pseudo-algorithme. Ce dernier nous permet de décrire de manière claire et structurée les différentes étapes du processus de migration. En exposant la logique du programme sans se soucier de la syntaxe spécifique d'un langage de programmation, il simplifie la compréhension de la solution et facilite sa planification. Le pseudo-algorithme suivant présente les étapes de notre migration des vers un format document (MongoDB) :

1 - Connexion à la base de données SQL

2 - Connexion à la base de données MongoDB

3 - Pour chaque table SQL de Classiclite :

3.1 - Extraction des données de la table SQL

3.2 - Pour chaque ligne de données :

- Transformation des données en format JSON compatible avec MongoDB
- Insertion des données transformées dans la collection MongoDB appropriée

4 - Vérification de la réussite de la migration (validation des données insérées dans MongoDB)

5 - Fermeture des connexions aux bases de données

6 - Fin

En élaborant ce pseudo-algorithme, nous avons maintenant une feuille de route claire qui nous permet de passer à l'étape suivante : le développement du programme de migration.

4 Développement du programme de migration des données

Pour la migration de nos données, nous avons utilisé SQLite comme base de données source et MongoDB comme base de données cible, en tirant parti des bibliothèques sqlite3, pymongo et pandas pour effectuer les opérations nécessaires.

Le processus de migration se déroule en plusieurs étapes clés : extraction des données de SQLite, transformation des données en format compatible avec MongoDB (en particulier le format document), et insertion des données dans les collections appropriées de MongoDB. Après chaque étape, des vérifications rigoureuses ont été effectuées afin de s'assurer que les données ont été correctement manipulées et insérées. Les résultats des requêtes et des transformations sont affichés pour garantir la précision de chaque opération et valider l'intégrité des données migrées.

5 Évaluation de la qualité de la migration

Nous avons évalué la qualité de notre migration en exécutant les mêmes dix requêtes SQL que celles effectuées avant la migration, cette fois sur notre base de données NoSQL. Pour cela, nous avons utilisé le langage de requêtes MongoDB. Toutes les requêtes ont donné les mêmes résultats, à l'exception de la requête 10 (bonus), où nous n'avons pas pu obtenir les mêmes calculs en raison de notre niveau de familiarité limité avec MongoDB.

6 Conclusion

En conclusion, la migration des données vers MongoDB a été un succès. Nous avons pu valider la précision de notre travail en comparant les résultats de nos neuf requêtes SQL avec celles exécutées sur notre base NoSQL, et les résultats se sont révélés identiques.

L'étape de conception du schéma cible a été la plus difficile pour nous lors de ce projet. Notamment en raison des choix à faire pour structurer les données de manière optimale dans le modèle NoSQL. Cette phase a nécessité une réflexion approfondie et des ajustements constants pour concilier la

flexibilité de MongoDB avec les exigences fonctionnelles du projet. Contrairement à la première étape, où la création des requêtes SQL a été relativement facile grâce à notre expérience préalable, l'utilisation de MongoDB a demandé davantage d'efforts, car il s'agissait d'un langage nouveau pour nous.

Néanmoins, ce projet nous a permis de découvrir une approche innovante des bases de données, plus flexible et évolutive. Nous avons particulièrement apprécié cette expérience, car elle nous a donné l'opportunité de mettre en pratique les connaissances théoriques acquises dans le module NoSQL, tout en nous confrontant à des situations concrètes et réelles.