الجـمهورية الجـزائرية الديمقراطـية الشـعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العـالي و البحث العـلمي
Ministère de L'Enseignement Supérieur et de la Recherche Scientifique
جـامعة عمار ثـليجي بالأغـواط
Université Ammar Telidji Laghouat

كـليـة العلـوم
Faculté des Sciences
قـسـم الإعـلام الآلـي
Département d'Informatique

Thèse de Master

Domaine : Mathématique et Informatique

Filière : Informatique

Option : Systèmes d'information et de décision

Présenté par : Gueffaf Nafissa

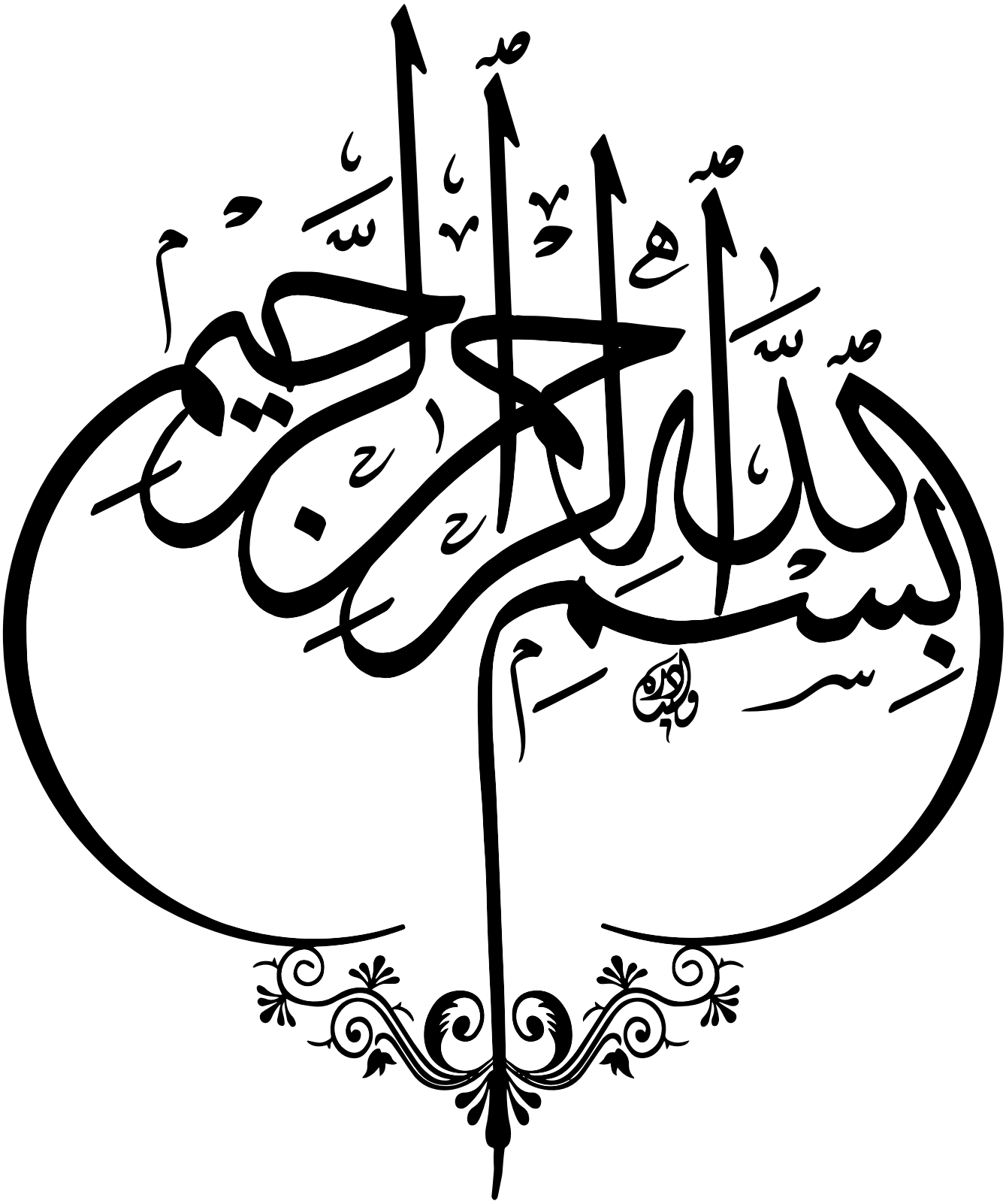
Thème

Prototype de transformation de schémas SQL vers NoSQL avec intégration de stratégies de dénormalisation

Soutenue publiquement, le 01/07/2025, devant le jury composé de :

Mr.	OUINTEN Youcef	Prof	(Université de Laghouat)	Président
Mr.	CHELLAMA Laradj	MAA	(Université de Laghouat)	Examinateur
Mr.	BOUAKKAZ Mustapha	Prof	(Université de Laghouat)	Examinateur
Mr.	MAICHA Mohamed EL Habib	MCB	(Université de Laghouat)	Encadreur

Année Universitaire 2024/2025


بسم الله الرحمن الرحيم

Dédicaces

Le voyage n'a pas été court, et il ne devait pas l'être. Le rêve n'était pas proche, et le chemin n'était pas facile, mais je l'ai accompli et j'ai réalisé ce rêve.

Louange, gratitude et remerciement à Dieu : c'est par Sa grâce seule qu'aujourd'hui je vois un rêve longtemps attendu devenu une réalité dont je suis fière.

Je dédie ce travail de fin d'études avec amour : À moi-même, qui malgré les obstacles ne s'est pas brisée, et malgré les difficultés ne s'est pas arrêtée, mais a toujours été plus forte qu'elle ne le croyait. À celle qui fut pour moi un refuge quand le monde s'est refroidi, une voix qui m'a appelée quand tout est devenu silence.

À celle qui m'a donné la vie deux fois, le jour de ma naissance et le jour de ma patience : toute la gratitude ne suffirait pas, ma chère mère.

À mon père, qui m'a appris que la dignité ne se donne pas, et que la connaissance est l'arme des patients. À toi qui as été

mon soutien quand les jours se sont faits durs, et ma fierté quand les mots m'ont manqué.

À mon reflet dans la faiblesse, et ma compagne dans la patience, tu as été l'étreinte inconditionnelle et la main toujours tendue sans attendre en retour : ma sœur Imane.

À mes piliers, sur lesquels je m'appuie : mes frères "Mohammed Oussama", "Abdelkader", et "Ibrahim", qu'Allah les garde comme une bénédiction dans ma vie, et qu'Il ne me prive jamais d'eux.

À toute ma famille, j'offre le fruit de mon parcours.

À mon encadrant, Dr. MAICHA Mohammed El Habib, dont le savoir et les conseils ont eu un grand impact sur ce travail. Il ne m'a jamais refusé son expertise, et son soutien bienveillant a été un véritable moteur pour achever ce projet avec assurance.

Et à tous mes enseignants du département d'informatique, toute ma reconnaissance pour leurs efforts et leurs enseignements.

Remerciements

Je rends toute ma gratitude à Allah Tout-Puissant, qui m'a accordé la force, la rigueur et la persévérance pour mener à bien ce travail.

Avant tout, j'adresse mes remerciements les plus sincères à mon encadrant, le Dr. MAICHA Mohammed El Habib, pour ses orientations, son soutien constant et ses conseils précieux tout au long de ce parcours. Son expertise et ses encouragements ont joué un rôle fondamental dans ma compréhension du sujet et le développement de mes compétences en recherche.

J'exprime également ma profonde reconnaissance aux membres du jury de soutenance...

Mes remerciements vont aussi à l'ensemble des enseignants du département d'informatique, ainsi qu'à toutes les personnes qui ont contribué, de près ou de loin, à mon parcours académique et à la réalisation de ce travail.

Abstract

The transformation of relational database schemas into document-oriented models suitable for NoSQL represents a key step in the transition toward more flexible and scalable data systems. Although the traditional structure of SQL databases is precise and well-organized, it does not align with the flexibility and denormalization strategies required by NoSQL systems. In this context, an approach has been developed that analyzes relational schemas based solely on "CREATE TABLE" statements, with the goal of automatically extracting entities and relationships, and defining transformation rules that guide the migration process while considering the type of relationships and applying appropriate denormalization strategies. To demonstrate the effectiveness of this approach, a simple application tool with a graphical interface was developed to convert relational schemas into a document-oriented representation in JSON format. The tool was validated through a case study representing a car sales management system, and the results confirmed the adequacy of the generated model and its ability to simplify the transition concepts from SQL to NoSQL (MongoDB).

ملخص

يُعد تحويل مخططات قواعد البيانات العلائقية إلى نماذج وثائقية ملائمة لـ NoSQL خطوة محورية في مسار الانتقال نحو أنظمة بيانات أكثر مرونة وقابلية للتوسع. وبالرغم من أن بنية SQL التقليدية دقيقة ومنظمة، الا انها لا تتلاءم مع متطلبات NoSQL من حيث مرونة النموذج وايضا استراتيجيات إلغاء التطبيع. في هذا السياق، تم تطوير مقاربة تعتمد على تحليل المخطط العلائقي انطلاقًا فقط من تعليمات "CREATE TABLE"، بهدف استخراج الكيانات والعلاقات بشكل تلقائي، وايضا وضع قواعد تحويل توجه عملية الانتقال، مع مراعاة نوع العلاقات وتطبيق استراتيجيات الغاء التطبيع. ولإثبات فعالية هذا النهج، تم تطوير أداة تطبيقية بسيطة بواجهة رسومية بسيطة، تقوم بتحويل المخططات إلى تمثيل وثائقي على شكل JSON. وقد تم التحقق من صحة هذه الأداة من خلال دراسة حالة تمثل نظامًا لإدارة مبيعات السيارات، حيث أظهرت النتائج ملاءمة النموذج الناتج وقدرته على تبسيط مفاهيم الانتقال من SQL إلى NoSQL (MongoDB).

الكلمات المفتاحية: مخطط SQL، NoSQL (MongoDB)، التضمين، المرجعية، إلغاء التطبيع، أداة بسيطة.

Résumé

La transformation des schémas de bases de données relationnelles en modèles documentaires adaptés à NoSQL constitue une étape essentielle dans la transition vers des systèmes de données plus flexibles et évolutifs. Bien que la structure traditionnelle des bases SQL soit précise et bien organisée, elle ne s'adapte pas aux exigences des bases NoSQL en termes de flexibilité de modèle et de stratégies de dénormalisation. Dans ce contexte, une approche a été développée, reposant sur l'analyse du schéma relationnel à partir des seules instructions "CREATE TABLE", afin d'extraire automatiquement les entités et les relations, et de définir des règles de transformation guidant le processus de migration, tout en tenant compte du type de relation et de l'application de stratégies de dénormalisation. Pour démontrer l'efficacité de cette approche, un outil applicatif simple avec une interface graphique a été conçu, permettant de convertir les schémas en représentation documentaire au format JSON. La validation de cet outil a été réalisée à travers une étude de cas représentant un système de gestion des ventes automobiles, dont les résultats ont montré l'adéquation du modèle généré et sa capacité à simplifier les concepts de transition de SQL vers NoSQL (MongoDB).

Mots Clée : Schéma SQL, NoSQL (MongoDB), Imbrication, Référencement, Dénormalisation, Outil simple.

Table des matières

liste des figures **1**

liste des tables **2**

Introduction Générale **3**

1 Généralité et Fondement de base **7**

1.1 Bases de données relationnelles (SQL) 7

1.1.1 Historique 7

1.1.2 Points forts 7

1.1.3 Points faible 8

1.2 Bases de données NoSQL 8

1.2.1 Bref aperçu sur NewSQL 8

1.2.2 Définition de NoSQL : 9

1.2.3 Histoire de NoSQL 9

1.2.4 Points forts 9

1.2.5 Types de NoSQL 10

1.3 Techniques de dénormalisation 15

1.3.1 La normalisation 15

1.3.2 La dénormalisation 15

1.4 Comparaison SQL vs NoSQL 17

2 Revue de la littérature **21**

2.1 Étude des travaux antérieurs 21

2.1.1 Transformations de Modèles Conceptuels vers NoSQL 21

2.1.2 Approches de Modélisation Directe pour les Bases de Données NoSQL 22

2.1.3 Outils et Méthodes de Migration Automatisée de SQL vers NoSQL . . . 22
2.1.4 Stratégies de Dénormalisation dans le Contexte de la migration . . . 23
2.1.5 Évaluation de la Faisabilité et Cadres méthodologiques pour la migration . . . 23
2.2 Comparaison des travaux antérieurs . . . 23
2.3 Transformations de Modèles Conceptuels vers NoSQL . . . 25
2.4 Approches de migration SQL vers NoSQL . . . 26
2.5 Stratégies de dénormalisation dans le contexte NoSQL . . . 26

3 Conception du prototype de transformation 29
3.1 Architecture du prototype . . . 29
3.1.1 Définition du prototype . . . 30
3.1.2 Description des composants architecturaux . . . 30
3.1.3 Présentation de MongoDB . . . 31
3.2 Principe de fonctionnement du prototype . . . 32
3.2.1 Définition des règles de transformation . . . 32
3.2.2 Étude de cas . . . 34
3.3 Representation de schéma MongoDB . . . 40
3.4 Analyse et discussion des résultats . . . 41
3.4.1 Développement de l'outil . . . 42
3.4.2 Fonctionnement de l'interface . . . 42

Conclusion Générale 46

Bibliographie 48

liste des figures

1.1 Organisation d'une collection dans une BD orientée-documents.(1) 11
1.2 Organisation d'une famille de colonnes dans une BD orientée colonnes.(1) . 12
1.3 Organisation des données dans une BD orientée-graphes.(1) 13
1.4 Organisation des données dans une BD clé-valeur.(1) 14

3.1 Architecture de la conversion SQL vers MongoDB 30
3.2 Modèle Conceptuel de Données (MCD) du système de gestion des ventes automobile 35
3.3 Modèle Logique de Données (MLD) dérivé du Modèle Conceptuel de Données 36
3.4 Interface de l'outil de transformation SQL vers MongoDB 42
3.5 Interface de l'outil en cours d'exécution 43
3.6 Interface de l'outil affichant les types de relations détectées (1 :1, 1 :N, N :M) 43
3.7 Résultat final de la conversion affiché en format "JSON" 44

liste des tables

1.1 Petites données vs Big Data 10
1.2 Bases de données populaires Documents et cas d'utilisation 12
1.3 Bases de données populaires colonnes et cas d'utilisation 13
1.4 Bases de données graphiques populaires et cas d'utilisation 14
1.5 Bases de données populaires Clé-Valeur et cas d'utilisation 15
1.6 SQL vs NoSQL 17

2.1 Étude comparative des approches de modélisation NoSQL 24

3.1 Création des structures MongoDB selon le type de relation 33
3.2 Comparaison simplifiée entre le modèle relationnel SQL et le modèle MongoDB obtenu 45

Introduction Générale

Contexte

L'évolution rapide des technologies de l'information a profondément transformé la manière dont les données sont produites, stockées et exploitées. Pendant plusieurs décennies, les systèmes de gestion de bases de données relationnelles (SGBDR), reposant sur le langage SQL, ont constitué la norme dominante pour structurer et interroger les informations. Ces bases relationnelles, fondées sur le modèle proposé par Edgar Codd dans les années 1970, se sont imposées grâce à leur rigueur, leur capacité à garantir l'intégrité des données et leur efficacité pour les systèmes transactionnels. Elles ont été largement adoptées dans les domaines traditionnels tels que la gestion d'entreprise, les systèmes bancaires ou les applications de gestion.

Cependant, ces dernières années, l'émergence de nouvelles applications — caractérisées par des volumes massifs de données (Big Data), une diversité croissante de formats (données semi-structurées ou non structurées), et des exigences de performance en temps réel — a mis en évidence les limites du modèle relationnel classique. La rigidité des schémas fixes, la difficulté à faire évoluer les structures de données rapidement, et la complexité croissante des jointures dans des contextes distribués ont conduit les ingénieurs à explorer d'autres paradigmes de gestion de données.

C'est dans ce contexte qu'apparaissent les bases de données NoSQL, acronyme de "Not Only SQL". Ces systèmes proposent une approche plus souple, adaptée à la gestion de données hétérogènes, évolutives et massivement distribuées. En abandonnant la stricte normalisation relationnelle, les bases NoSQL permettent d'optimiser certaines opérations en favorisant la dénormalisation, c'est-à-dire en acceptant une certaine redondance des données afin de réduire les coûts des jointures et d'améliorer les performances.

Problématique

Face à cette évolution, de nombreuses entreprises et institutions se posent aujourd'hui la question de migrer une partie de leurs bases relationnelles existantes vers des solutions NoSQL. Cette migration n'est pas triviale : les modèles conceptuels diffèrent radicalement, les relations entre données ne se traduisent pas toujours directement, et les stratégies de dénormalisation doivent être soigneusement pensées pour ne pas compromettre l'intégrité ou la cohérence de l'information.

De plus, bien que la littérature scientifique ait proposé plusieurs approches et outils de transformation de schémas SQL vers NoSQL, ces solutions restent parfois complexes à mettre en œuvre, peu généralisables ou adaptées à des contextes bien spécifiques. Les ingénieurs manquent encore de repères clairs sur les meilleures pratiques de transformation et sur les impacts des différentes stratégies de dénormalisation dans un contexte concret.

La problématique centrale de ce mémoire est donc la suivante : Comment concevoir une méthode simple, compréhensible et applicable permettant de transformer un schéma relationnel SQL vers une structure NoSQL tout en intégrant les stratégies de dénormalisation adaptées ? Plus précisément : – Quelles règles de transformation appliquer pour garantir une correspondance efficace entre les modèles ? – Quelles décisions de dénormalisation adopter pour concilier performance, simplicité et intégrité ?

C'est à ces questions que ce mémoire s'efforce d'apporter des éléments de réponse.

Objectifs du mémoire

L'objectif général de ce mémoire est de concevoir un prototype simplifié de transformation de schémas SQL vers NoSQL intégrant des stratégies de dénormalisation.

Méthodologie adoptée

Pour atteindre les objectifs fixés, ce mémoire adopte une méthodologie en plusieurs étapes, combinant une approche théorique (analyse documentaire) et une approche pratique (conception d'un prototype et expérimentation).
L'enchaînement méthodologique retenu est le suivant :

1. Cette étape consiste à explorer les concepts fondamentaux nécessaires à la compréhension du sujet, notamment :
   — Les bases de données relationnelles (SQL) et leurs principes de modélisation ;
   — Les bases de données NoSQL, en générale ;
   — Les techniques de dénormalisation et leur utilité dans un contexte NoSQL ;
   — Une comparaison conceptuelle et structurelle entre les deux paradigmes (SQL vs NoSQL).
2. Étude documentaire approfondie :
   Une revue de la littérature scientifique et technique sera menée afin de :
   — Comprendre en détail les modèles de données SQL et NoSQL, ainsi que les concepts de normalisation et dénormalisation ;
   — Identifier et analyser les travaux existants relatifs à la transformation de schémas SQL vers NoSQL, qu'il s'agisse de méthodes, d'outils ou de cadres méthodologiques ;
   — Recenser les règles de transformation proposées et évaluer leur pertinence pour la construction d'un processus simplifié et applicable.
3. Synthèse et définition des règles de transformation :
   À partir de l'analyse bibliographique, un ensemble de règles de correspondance entre schémas relationnels et structures NoSQL sera défini. Ces règles prendront en compte les types de relations (1 :1, 1 :N, N :M), les cas d'usage courants, et les stratégies de dénormalisation les plus adaptées au contexte ciblé (par exemple, imbrication de documents dans MongoDB pour optimiser les lectures).
4. Conception du prototype de transformation :
   Sur la base des règles établies, un prototype simplifié sera conçu. Celui-ci permettra de prendre en entrée un schéma SQL représentatif (exprimé sous forme de tables et relations simples) et de produire une structure NoSQL correspondante (sous forme de collections et documents imbriqués).
5. Application à un cas d'étude concret
   Le prototype sera testé sur un cas d'étude réaliste : un petit schéma dédié à un système de gestion des ventes automobiles, comprenant plusieurs tables et rela-

tions. Ce cas permettra d'illustrer concrètement la démarche de transformation et d'évaluer la pertinence des choix de dénormalisation opérés.

6. Analyse et évaluation des résultats :
   Les résultats obtenus seront analysés afin de :
   — Vérifier la conformité de la transformation avec les règles définies ;
   — Apprécier les gains en termes de simplification d'accès aux données et de performances potentielles ;
   — Identifier les limites de l'approche et formuler des pistes d'amélioration pour des travaux futurs.

Cette méthodologie progressive et structurée vise à assurer la rigueur de l'analyse théorique tout en garantissant une application pratique pertinente et réaliste.

Organisation du mémoire

Cette mémoire est structurée en trois chapitres principaux :

— Le **premier chapitre** présente les concepts fondamentaux liés aux bases de données relationnelles et aux bases NoSQL, ainsi que les notions de modélisation, de normalisation et de dénormalisation.
— Le **deuxième chapitre** propose une revue récente des travaux de recherche portant sur la transformation des modèles relationnels vers des modèles NoSQL, en mettant en lumière les approches, outils et stratégies suggérés dans ce domaine.
— Enfin, le **dernier chapitre** décrit l'architecture du prototype, détaille les étapes de développement de l'outil, les règles de conversion appliquées, ainsi qu'une étude de cas visant à valider cette approche.

Chapitre 1

Généralité et Fondement de base

Introduction

Les bases de données constituent les piliers fondamentaux des systèmes d'information, évoluant au fil du temps pour répondre aux besoins des différentes applications. Ce chapitre offre un aperçu approfondi de l'évolution des bases de données, des avantages et inconvénients de chaque type, et compare SQL et NoSQL afin de comprendre dans quels cas chacun est le choix le plus approprié.

1.1 Bases de données relationnelles (SQL)

1.1.1 Historique

Dans les années 1970, Edgar Codd a proposé le modèle de bases de données relationnelles (RDBMS) alors qu'il travaillait chez IBM. Le projet System R a été lancé dans le laboratoire IBM de San José pour tester l'efficacité de ce modèle. Plus tard, Ray Boyce et Donald Chamberlin ont développé le langage SEQUEL pour simplifier les requêtes, mais il a été raccourci en SQL en raison de problèmes de marque déposée. Après la mort de Boyce en 1974, le développement de SQL s'est poursuivi dans le cadre du projet System R et est devenu plus tard la norme principale des bases de données relationnelles, contribuant à l'émergence de systèmes tels qu'Oracle, IBM DB2 et MySQL.(2)

1.1.2 Points forts

SQL(RDBMS) est un langage de programmation et un outil essentiel. Il permet de gérer et de contrôler les bases de données. Cette simple réalité rend SQL indispensable

dans notre monde moderne axé sur la technologie, en raison de ses avantages (3) :

- Cohérence des données et intégrité des transactions.
- Schémas bien définis et données structurées.
- Requêtes complexes et données relationnelles.
- Transactions multi-lignes.
- Langage standardisé, facile à apprendre et à utiliser.
- Portabilité et évolutivité.
- Les six formes normales, visant à réduire la redondance et à améliorer l'intégrité des données. (4)

1.1.3 Points faible

Cependant, il présente certaines limites, notamment (4) :

- SQL ne peut pas gérer de vastes ensembles de données non structurées ni une grande variété de modèles de données.
- Absence de traitement immédiat des données.
- Flexibilité limitée des données.
- Interface complexe.
- Certaines versions sont coûteuses.

• C'est ce qui nous mène à réfléchir sur une nouvelle façon de représenter la nouvelle structure de données :

1.2 Bases de données NoSQL

1.2.1 Bref aperçu sur NewSQL

NewSQL est une classe de systèmes de gestion de bases de données relationnelles. Le terme NewSQL est apparu pour la première fois en 2011 grâce à l'analyste Matthew Aslett du groupe 451, pour désigner une nouvelle génération de systèmes de gestion de bases de données visant à offrir les mêmes performances élevées que les bases NoSQL, tout en conservant le modèle relationnel et la prise en charge du langage SQL. Ces systèmes étaient conçus pour fournir une scalabilité horizontale et de meilleures performances, tout en garantissant la cohérence et en maintenant la facilité d'utilisation du SQL.
Cependant, NewSQL n'a pas connu la large adoption attendue en raison de la complexité de son implémentation et des défis liés à son évolutivité par rapport à d'autres systèmes. De plus, les bases de données relationnelles traditionnelles ont continué à améliorer leurs performances et à intégrer de nouvelles fonctionnalités répondant aux besoins des applica-

tions modernes. Par conséquent, de nombreuses entreprises ont préféré des solutions plus flexibles pour gérer des volumes de données massifs et variés.
Les bases de données NoSQL ont connu un développement rapide ces dernières années, devenant de plus en plus adaptées aux applications nécessitant le traitement de grandes quantités de données hétérogènes. Grâce à leurs caractéristiques, que nous verrons plus tard, elles ont contribué au déclin de l'intérêt pour les solutions NewSQL.(5)

1.2.2 Définition de NoSQL :

NoSQL, qui signifie "Not Only SQL", est une approche de système de gestion de base de données utilisée pour traiter des données non structurées et semi-structurées dans une base de données. Cela signifie que les données qui ne peuvent pas être analysées ou comptées par les bases de données relationnelles traditionnelles (par exemple, SQL) peuvent rester dans leur format d'origine et être intégrées dans une base de données NoSQL. La raison pour laquelle on l'appelle NoSQL est que ces bases de données peuvent gérer des modèles de données non tabulaires et non relationnels, tout en prenant en charge des langages de requête de type SQL.(6)
Contrairement aux bases de données SQL, NoSQL offre des schémas dynamiques permettant un stockage plus flexible des données, ce qui les rend idéales pour gérer d'énormes volumes de données provenant de sources variées. Ainsi, plusieurs bases de données NoSQL matures sont disponibles pour aider les entreprises à étendre leurs applications de big data.(4)

1.2.3 Histoire de NoSQL

La première utilisation du terme NoSQL remonte à 1998 par Carlo Strozzi pour une base de données relationnelle légère et open-source qui n'utilisait pas SQL.Le terme a été réutilisé en 2009 par Eric Evans et Johan Oskarsson pour décrire les bases de données non relationnelles. (7)

1.2.4 Points forts

Les bases de données NoSQL permettent (8) :

— Flexibilité du schéma : NoSQL prend en charge plusieurs types de données et permet de modifier facilement le schéma pour répondre aux exigences des données en évolution.

— Développement rapide : Offre un environnement flexible qui accélère le développement des applications en réduisant le besoin de transformation des données.

— Évolutivité : Permet d'augmenter facilement la capacité avec la croissance des données et du trafic sans interruption du système.
— Haute disponibilité : Basé sur une architecture distribuée qui réduit les points de défaillance et garantit la continuité du service.
— Performance des requêtes : Optimisé pour un accès rapide aux données en limitant le besoin de jointures complexes.
— Gestion des big data : Conçu pour traiter d'énormes volumes de données, idéal pour l'analyse des big data et l'Internet des objets, Analyses en temps réel.

Le tableau [1.1] regroupe les différente caractéristique (4V's) de big data volume, variété, vélocité et véracité, chaqu'une est comparé par la métrique de petite donnée vs big data(9) :

Caractéristique	**Petites données**	**Big Data**
Volume	Données de l'ordre de dizaines ou centaines de gigaoctets	Taille des données supérieure aux téraoctets
Variété	Données généralement structurées et uniformes	Données souvent non structurées et hétérogènes
Vélocité	Flux de données régulé et constant, agrégation lente des données	Les données arrivent à des vitesses extrêmement élevées, avec une agrégation massive en peu de temps
Véracité	Données généralement de haute qualité et fiables	La qualité et la fiabilité des données peuvent varier considérablement

Tableau 1.1 – Petites données vs Big Data

1.2.5 Types de NoSQL

Les bases de données NoSQL peuvent être classées en quatre types principaux, en fonction de leurs méthodes de stockage et de récupération des données, et chaque type possède des avantages et des cas d'utilisation spécifiques, faisant de NoSQL un choix privilégié pour les applications de big data, les analyses en temps réel, l'informatique en nuage et les systèmes distribués :

1.2.5.1 Bases de données orientées Documents :

C'est un type de base de données NoSQL qui stocke les données sous forme de documents, généralement en JSON (JavaScript Object Notation), BSON (Binary JSON) ou XML (eXtensible Markup Language), au lieu de tables relationnelles. Elle permet une organisation flexible des données et facilite leur récupération rapide.
Les documents sont regroupés en collections "Figure [1.1]", mais ils ne nécessitent pas un schéma strict, offrant ainsi une grande flexibilité. Chaque document peut être accédé rapidement grâce à un index attribué, ce qui accélère les requêtes (10).

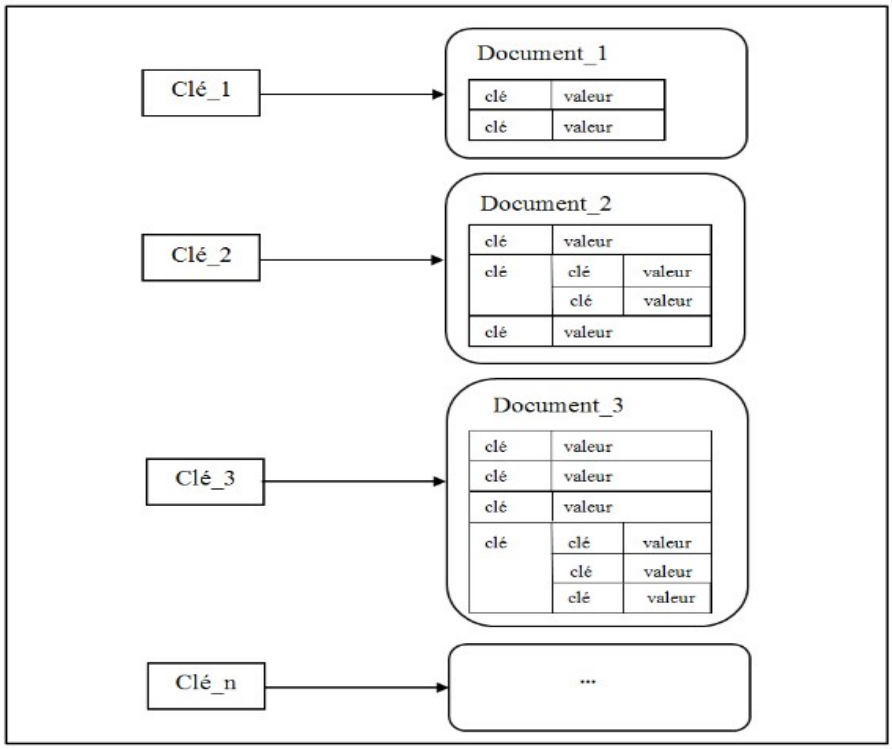


Figure 1.1 – Organisation d'une collection dans une BD orientée-documents.(1)

• Caractéristiques principales

Ce qui suit met en lumière les principales caractéristiques des bases de données orientées documents, en complément du Tableau [1.2] qui présente les bases de données documentaires courantes et leurs cas d'utilisation (10) :

- **Schéma flexible :** les documents peuvent avoir des structures différentes.
- **Création et maintenance rapides :** la gestion des documents est simple et nécessite peu d'efforts.
- **Pas de clés étrangères :** les documents sont indépendants, éliminant le besoin de relations complexes.
- **Formats ouverts :** prend en charge JSON, XML et d'autres formats.

Base de données	Cas d'utilisation
MongoDB	Gestion de contenu, catalogues de produits, profils utilisateurs
CouchDB	Applications hors ligne, synchronisation mobile
Firebase Firestore	Applications en temps réel, messagerie instantanée

Tableau 1.2 – Bases de données populaires Documents et cas d'utilisation

1.2.5.2 Bases de données orientées Colonnes :

C'est une base de données non relationnelle qui stocke les données en colonnes plutôt qu'en lignes. Cela permet de lire directement certaines colonnes sans charger de données inutiles "Figure [1.2]", améliorant ainsi l'efficacité de récupération et la vitesse d'analyse des données. Les bases de données orientées colonnes sont utilisées pour stocker de grandes quantités de données.(10)

Figure 1.2 – Organisation d'une famille de colonnes dans une BD orientée colonnes.(1)

• Caractéristiques principales

Ce qui suit met en lumière les principales caractéristiques des bases de données orientées colonnes, en complément du Tableau [1.3] qui présente les bases de données colonnaires courantes et leurs cas d'utilisation (10) :

- **Haute scalabilité :** Prise en charge du traitement des données distribuées.
- **Compression :** Le stockage en colonnes permet une compression efficace des données.
- **Performance des requêtes rapide :** Idéale pour les analyses de données.

Base de données	Cas d'utilisation
Apache Cassandra	Analytique en temps réel, applications IoT
Google Bigtable	Apprentissage automatique à grande échelle, séries temporelles
HBase	Écosystème Hadoop, stockage distribué

Tableau 1.3 – Bases de données populaires colonnes et cas d'utilisation

1.2.5.3 Bases de données orientées Graphes :

Et ils mettent l'accent sur les relations entre les éléments. Elles stockent les données comme des nœuds connectés par des liens ou des relations "Figure[1.3]", ce qui les rend idéales pour les requêtes complexes basées sur les relations, et les données sont représentées sous forme de nœuds (objets) et d'arêtes (connexions), où cela permet aux algorithmes de parcours de graphe rapide de récupérer les relations efficacement, et elles sont utilisées dans les cas où les relations sont aussi importantes que les données elles-mêmes.(10)

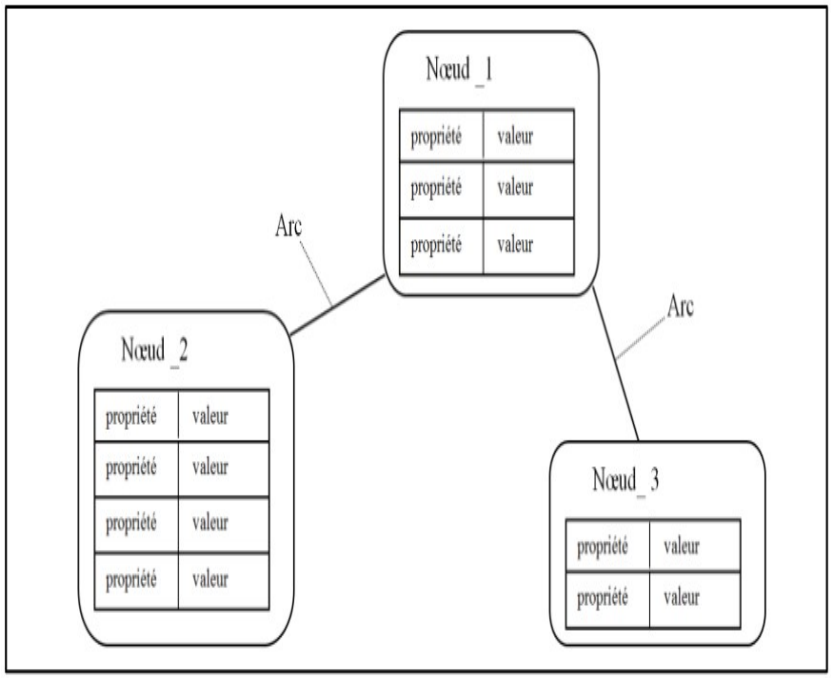


Figure 1.3 – Organisation des données dans une BD orientée-graphes.(1)

• Caractéristiques principales

Ce qui suit est une clarification des principales caractéristiques des bases de données de graphes, en plus du tableau [1.4] qui présente les bases de données de graphes courantes et leurs cas d'utilisation (10) :

— **Stockage centré sur les relations :** Idéal pour les réseaux sociaux, la détection de fraude, et les moteurs de recommandation.

— **Traitement des requêtes en temps réel :** Les requêtes renvoient des résultats quasi-instantanément.
— **Flexibilité du schéma :** S'adapte facilement aux évolutions des structures relationnelles.

Base de données	Cas d'utilisation
Neo4j	Détection de fraude, réseaux sociaux
Amazon Neptune	Graphes de connaissances, recommandations IA
ArangoDB	Base de données multi-modèle, cybersécurité

Tableau 1.4 – Bases de données graphiques populaires et cas d'utilisation

1.2.5.4 Bases de données Clé-Valeur :

C'est un type de base de données NoSQL le plus simple qui stocke les données sous forme de paires clé-valeur "Figure[1.4]", où une clé unique est utilisée pour récupérer rapidement les données. Les valeurs peuvent être simples ou complexes, ce qui la rend idéale pour la mise en cache et les applications en temps réel.(10)

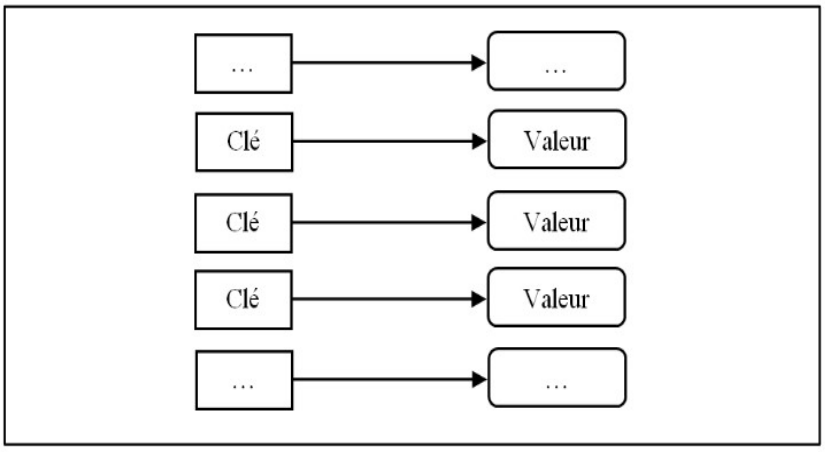


Figure 1.4 – Organisation des données dans une BD clé-valeur.(1)

• Caractéristiques principales

Ce qui suit met en lumière les principales caractéristiques des bases de données clé-valeur, en complément du tableau [1.5] qui présente les bases de données clé-valeur courantes et leurs cas d'utilisation (10) :

— **Simplicité :** Récupération des données très rapide grâce à l'accès direct via la clé.
— **Évolutivité :** Conçue pour le scaling horizontal et le stockage distribué.
— **Vitesse :** Idéale pour la mise en cache et les applications en temps réel.

Base de données	Cas d'utilisation
Redis	Mise en cache, classements en temps réel, stockage de session
Memcached	Mise en cache en mémoire à haute vitesse
Amazon DynamoDB	Applications cloud évolutives

Tableau 1.5 – Bases de données populaires Clé-Valeur et cas d'utilisation

1.3 Techniques de dénormalisation

1.3.1 La normalisation

La normalisation des données dans les bases de données est un processus couramment utilisé en statistiques, en science des données et en apprentissage automatique (machine learning) pour mettre à l'échelle les valeurs de différentes variables dans un même intervalle. L'objectif principal de là est de rendre les données comparables entre elles et plus facilement interprétables par les algorithmes d'analyse et de modélisation. Parmi les raisons ayant conduit à l'émergence de la dénormalisation, on retrouve principalement : le manque de flexibilité, le frein à l'innovation, le coût élevé d'adaptation aux normes, et l'approche « taille unique » qui ne tient pas compte de la diversité des besoins utilisateurs.(11)

1.3.2 La dénormalisation

Est le processus de simplification de sa structure en combinant les données pour accélérer l'accès à l'information. Cette démarche réduit l'intégrité des données en abaissant les niveaux de normalisation, mais améliore les performances en limitant les opérations JOIN coûteuses. Bien qu'elle puisse entraîner des anomalies et des redondances, celles-ci peuvent être contrôlées par des contraintes logicielles lors de la saisie des données.
La dénormalisation est utilisée pour améliorer la vitesse de récupération des données. Elle est utile pour améliorer les performances des requêtes en réduisant les JOIN, faciliter la gestion en fournissant des données directement disponibles, et accélérer la production de rapports analytiques.(12)
Il existe diverses techniques de dénormalisation de bases de données utilisées en fonction du cas d'utilisation. Voici cinq techniques clés couramment utilisées (13) :

1.3.2.1 Table pré-jointe

Une table pré-jointe est une technique que vous pouvez utiliser lorsque les opérations de jointure sont coûteuses en termes de calcul. Dans cette approche, vous pouvez dupliquer

des champs de données spécifiques (colonnes) à travers plusieurs tables normalisées afin de minimiser les jointures ou les recherches fréquentes lors des requêtes.(13)

1.3.2.2 Tables en miroir

Cette technique implique la création d'une copie complète ou partielle de tables existantes. Dans une approche de tables miroirs, vous pouvez consolider des données logiquement liées dans une seule table, même si cela entraîne une duplication. Cela vous permet de simplifier le schéma en réduisant la fragmentation de données similaires entre les tables.(13)

1.3.2.3 Division de table

Le partitionnement de table fait référence à la décomposition d'une grande table en tables plus petites et plus faciles à gérer pour un traitement des requêtes plus rapide. Vous pouvez effectuer le partitionnement de table de deux manières(13) :

• **Division horizontale de la table :** Dans le partitionnement horizontal de table, vous pouvez distribuer les lignes d'une table sur plusieurs tables tout en conservant les mêmes colonnes. Cette technique est utile lorsque vous pouvez diviser logiquement les données en fonction de certains critères, tels que les régions, les départements ou les périodes de temps.

• **Division verticale de la table :** Le partitionnement vertical implique la division d'une table en fonction des colonnes tout en appliquant la clé primaire à chaque partition. Il est bénéfique lorsque certains champs de données sont interrogés plus fréquemment que d'autres, permettant un accès optimisé à ces colonnes.

1.3.2.4 Ajout de colonnes dérivées

L'ajout de colonnes dérivées est une technique qui vous aide à améliorer les performances des requêtes en stockant des valeurs pré-calculées dans la table elle-même. Une colonne dérivée est une nouvelle colonne qui est créée en effectuant un calcul sur des données existantes. En ajoutant une colonne dérivée, vous pouvez éviter de recalculer la valeur pour améliorer l'efficacité.(13)

1.3.2.5 Vues matérialisées

Les vues matérialisées sont des résultats de requêtes pré-calculés stockés sous forme de table distincte dans la base de données. Elles sont conçues pour optimiser les performances des requêtes coûteuses, telles que celles impliquant des jointures et des agrégations. En

stockant les résultats de ces requêtes, les vues matérialisées facilitent la récupération rapide des données de la base de données sans exécuter la même requête à plusieurs reprises.(13)

1.4 Comparaison SQL vs NoSQL

Le passage d'une architecture de données traditionnelle basée sur SQL vers des solutions NoSQL soulève des questions importantes quant à leurs caractéristiques distinctes. Le Tableau [1.6] ci-dessous (14) offre une comparaison détaillée de ces deux types de systèmes de gestion de bases de données selon différents aspects cruciaux :

Aspect	**SQL (Relationnel)**	**NoSQL(Non-relationnel)**
Structure des données	Tables avec lignes et colonnes	Basé sur des documents, clé-valeur, famille de colonnes ou graphe
Schéma	Schéma fixe	Schéma flexible
Évolutivité	Évolutif verticalement	Évolutif horizontalement
Intégrité des données	Conforme à ACID (forte cohérence)	Conforme à BASE (plus disponible, moins cohérent)
Langage de requête	SQL (Structured Query Language)	Variable (ex. : MongoDB utilise son propre langage de requête)
Performance	Requêtes complexes et transactions	Big Data et opérations de lecture/écriture
Cas d'utilisation	Idéal pour les systèmes transactionnels	Idéal pour le Big Data, les applications web en temps réel et les Data Lakes
Exemples	MySQL,PostgreSQL, Oracle, MS SQL Server	MongoDB, Cassandra, CouchDB, Neo4j

Tableau 1.6 – SQL vs NoSQL

• Type

Les bases de données **SQL** sont appelées bases de données relationnelles (RDBMS), tandis que les bases de données **NoSQL** sont considérées comme non relationnelles ou distribuées.

• Langage

SQL utilise le langage de requête structuré (SQL) pour manipuler les données et relier les tables via des jointures (JOINs). **NoSQL** utilise différents langages de requête, certains similaires à SQL (comme Cassandra) et d'autres complètement différents, utilisant des formats orientés documents comme JSON.

• Structure

SQL organise les données sous forme de tables, ce qui la rend idéale pour les systèmes nécessitant une forte interconnexion des données.

— **Exemple d'une base SQL(14) :**

```
{
   "id": "101",
   "category": "food",
   "name": "Apples",
   "qty": "150"
}
```

NoSQL, en revanche, stocke les données sous forme de documents, paires clé-valeur, bases de données graphiques ou magasins de colonnes larges, offrant ainsi une plus grande flexibilité pour gérer des données non structurées.

— **Exemple d'une base NoSQL(14) :**

```
Produits = [
{
   "id": "101",
   "category": "food",
   "name": "California Apples",
   "qty": "150"
},
{
   "id": "102",
   "category": "electronics",
   "name": "Apple MacBook Air",
   "qty": "10",
   "specifications": {
      "storage": "256GB SSD",
```

```
      "cpu": "8 Core",
      "camera": "Caméra FaceTime HD 1080p" }}
]
```

• Scalabilité

SQL sont généralement scalables verticalement, c'est-à-dire en améliorant les performances d'un seul serveur (augmentation de la mémoire RAM, du processeur ou du stockage). En revanche, les bases **NoSQL** sont scalables horizontalement, permettant l'ajout de plusieurs serveurs pour répartir les données, ce qui les rend plus adaptées aux volumes massifs de données évolutives.

• Schéma

SQL : Schéma fixe (structure prédéfinie) avec une normalisation stricte des données.
NoSQL : Schéma flexible et adaptable aux données semi-structurées ou non structurées.

• Performance

SQL : Performant pour les requêtes complexes et les transactions, mais peut devenir lent avec de très grandes quantités de données.
NoSQL : Rapide pour les opérations massives de lecture/écriture, notamment avec des données non structurées.

• Cas d'utilisation

SQL : ERP, applications d'entreprise traditionnelles,Banque,CRM.
NoSQL : Applications mobiles et web en temps réel, Internet des objets (IoT), personnalisation et recommandations, gestion des stocks et catalogues, détection de fraude et vérification d'identité, publicité numérique.

• Propriétés suivies

SQL suit les propriétés **ACID** (Atomicité, Cohérence, Isolation, Durabilité), garantissant l'intégrité des transactions et la protection des données (15) :

- **Atomicité :** Une transaction est traitée comme une seule unité. Elle est soit entièrement validée, soit totalement annulée.
- **Cohérence :** Garantit que chaque transaction maintient l'intégrité des données.

— **Isolation :** Garantit que l'exécution simultanée des transactions ne compromet pas l'état de la base de données.
— **Durabilité :** Les données validées restent enregistrées même en cas de panne du système.

NoSQL suit le théorème **CAP** (Cohérence, Disponibilité, Tolérance au partitionnement), privilégiant la performance et la distribution, tout en assouplissant certaines garanties de cohérence stricte (15) :

— **Cohérence éventuelle :** Chaque lecture renvoie soit la dernière écriture, soit une réponse erronée.
— **Disponibilité :** Chaque requête reçoit une réponse, mais celle-ci peut ne pas contenir les dernières mises à jour.
— **Tolérance au partitionnement :**Le système continue de fonctionner même en cas de latence ou de perte de communication entre les nœuds.

L'un des défis fondamentaux est défini par le théorème **CAP**, qui stipule qu'une base de données distribuée ne peut garantir que deux des trois propriétés suivantes à la fois : cohérence, disponibilité et tolérance au partitionnement.

Conclusion

Ce chapitre présente un aperçu des bases de données relationnelles traditionnelles et de leur évolution vers des solutions modernes offrant de meilleures performances et une plus grande évolutivité. Alors que SQL reste le choix privilégié pour les applications nécessitant l'intégrité des données et la gestion de transactions complexes, NoSQL s'est avéré efficace pour le traitement des grandes quantités de données non structurées.

Chapitre 2

Revue de la littérature

Introduction

L'évolution des systèmes de gestion de bases de données a vu un intérêt croissant pour les bases de données NoSQL, en raison de leur capacité à gérer de manière efficace de grandes quantités de données variées. L'un des défis majeurs consiste à transformer les schémas SQL traditionnels en modèles NoSQL adaptés, tout en intégrant des stratégies de dénormalisation pour améliorer les performances.Dans ce chapitre, on vise à examiner les travaux de recherche existants sur la transformation des schémas de bases de données. La littérature académique présente une diversité d'études publiées. Nous avons analysé et classé ces travaux selon cinq techniques distinctes, qui ont ensuite été comparées dans le tableau [2.1]. Nous avons analysé et classé ces travaux selon cinq techniques distinctes, qui ont ensuite été comparées dans le tableau [**??**]. Une analyse approfondie a permis d'en dégager les résultats et d'identifier les techniques fondamentales, en mettant particulièrement l'accent sur les approches de migration de SQL vers NoSQL ainsi que sur les techniques de dénormalisation qui y sont associées.

2.1 Étude des travaux antérieurs

2.1.1 Transformations de Modèles Conceptuels vers NoSQL

Plusieurs études ont exploré comment transformer des modèles conceptuels, comme UML, en schémas NoSQL. Par exemple, (16) ont utilisé une approche appelée MDA pour convertir des modèles UML en modèles NoSQL, en créant un modèle logique qui fonctionne avec différents types de bases NoSQL. De même, la thèse (1) met l'accent sur l'utilisation de MDA pour cette transformation, tout en intégrant des règles métiers pour garantir la

cohérence des schémas finaux. (17) ont développé UML4NoSQL, qui utilise des extensions UML pour créer automatiquement des schémas adaptés aux bases de données orientées documents. Cependant, ces travaux partent tous d'un modèle UML, ce qui signifie qu'il pourrait être nécessaire de convertir d'abord des schémas SQL en UML pour notre projet, en considérant la dénormalisation et l'imbrication.Néanmoins, les étapes de conversion vers les structures NoSQL ont été mises en lumière par (18) proposent une méthode de conception de base de données NoSQL, basée sur un modèle conceptuel UML inspiré du cadre de Peter Chen. Bien que leur objectif principal soit la conception initiale dans un contexte de persistance polyglotte, leur travail souligne l'importance d'une modélisation conceptuelle rigoureuse avant la conception physique NoSQL.

2.1.2 Approches de Modélisation Directe pour les Bases de Données NoSQL

Face au manque de standards clairs pour modéliser les bases de données NoSQL, certains chercheurs ont exploré des méthodes directes. Par exemple, (19) ont adapté les diagrammes Entité-Association pour mieux visualiser et comprendre les structures des bases NoSQL orientées documents. De son côté, (20) a insisté sur l'importance de bien réfléchir à la modélisation, en utilisant des objets imbriqués et la dénormalisation pour optimiser l'accès aux données. Bien que ces études soulignent les particularités des bases NoSQL, elles n'abordent pas la transformation automatique à partir de schémas SQL existants.

2.1.3 Outils et Méthodes de Migration Automatisée de SQL vers NoSQL

Diverses recherches ont cherché à créer des outils pour faciliter la migration des bases de données traditionnelles vers des systèmes NoSQL.(21) ont développé DLoader, qui automatise le processus de transfert des données et des schémas vers MongoDB et Cassandra. De même, (22) ont proposé MigDB, qui prend en compte les relations entre les tables pour un passage fluide vers MongoDB. (23) ont introduit un modèle qui utilise des graphes pour pour imbriquer les tables liées lors de la migration vers des bases NoSQL orientées documents, optimisant ainsi les performances des requêtes. (24) a conçu un algorithme capable de migrer les données sans besoin de connaître le schéma à l'avance visant l'optimisation de la structure des données NoSQL. (25) a mis au point une approche en deux étapes pour migrer vers HBase, en automatisant la transformation des données et des schémas. Ces travaux se concentrent sur l'automatisation des migrations, un aspect

clé de notre étude.

2.1.4 Stratégies de Dénormalisation dans le Contexte de la migration

Il existe une stratégie visant à améliorer la performance dans les bases de données NoSQL, principalement en réduisant le besoin de jointures, connue sous le nom de dénormalisation. Dans ce contexte, (26) ont proposé une méthodologie indépendante pour la dénormalisation et la migration des schémas SQL vers des environnements NoSQL, ciblant spécifiquement les systèmes de gestion de contenu (CMS). Leur approche visait à adapter les structures de données existantes aux modèles NoSQL sans nécessiter une refonte complète, ce qui a entraîné des améliorations notables en termes d'efficacité. Parallèlement, les travaux de (27) et (28) ont exploré la transformation de schémas multidimensionnels en structures NoSQL, avec des gains de performance significatifs. (29) ont développé un ensemble de règles de conversion pour migrer des schémas relationnels vers différents modèles NoSQL, en tenant compte des relations essentielles des données et en cherchant à maintenir l'intégrité des informations. Collectivement, ces approches soulignent le rôle crucial de la dénormalisation dans le processus de transition vers les systèmes NoSQL.

2.1.5 Évaluation de la Faisabilité et Cadres méthodologiques pour la migration

Outre les approches techniques de transformation de schémas, ces études se sont concentrées sur l'aspect méthodologique de la migration des bases de données SQL vers NoSQL. Ainsi, (30) ont développé une approche pour évaluer la faisabilité de cette transition. Leur méthode reposait sur une analyse approfondie de la structure des schémas relationnels existants, ainsi que sur l'identification des transformations potentielles permettant la migration des données sans compromettre leur intégrité. De même,(31) ont proposé une méthode basée sur la rétro-ingénierie des schémas SQL pour effectuer la migration vers MongoDB tout en assurant la préservation des contraintes d'intégrité. Ces travaux fournissent des cadres méthodologiques importants pour aborder le processus de migration vers les environnements NoSQL.

2.2 Comparaison des travaux antérieurs

Une analyse comparative des études classées est présentée dans le tableau [2.1], mettant en évidence les objectifs, les points de départ et les aspects liés aux Big Data

Travail	Entrées (Point de Départ)	Sorties (Systèmes NoSQL Ciblés)	Caractéristiques du Big Data (Mentionnées)
(16)	Diagramme de classes d'UML	Orienté documents, colonnes, et graphe	Volume, variété et vitesse
(1)	diagramme de classes d'UML	Orienté documents, colonnes et graphe	-
(31)	Diagrammes Entité-Association(ER) adaptés	Orienté documents	-
(17)	cas d'utilisation et diagramme de classes	Orienté documents	Variété, volume, vélocité et véracité
(20)	Analyse des BDD relationnelles	Orienté documents (techniques de modélisation)	-
(21)	Schéma relationnel SQL	Orienté documents (MongoDB), Orienté Colonnes (Cassandra)	-
(23)	MySQL	Orienté documents (MongoDB)	-
(24)	BDD relationnelles SQL (sans schéma préalable)	NoSQL (optimisation générale)	-
(22)	Schéma relationnel SQL(RDB)	Orienté documents (MongoDB)	volume and Variété
(25)	BDD relationnelle SQL	Orienté colonnes (HBase)	-
(26)	Schémas relationnels SQL (CMS)	NoSQL	-
(30)	Schéma relationnel SQL(RDB)	Général (évaluation de la faisabilité)	-
(31)	Schéma relationnel SQL(RDB)	Orienté documents (MongoDB)	-
(29)	Schéma relationnel SQL(RDB)	Orienté documents, colonnes, graphes	-
(27)	Schémas multidimensionnels (OLAP)	Orienté documents, colonnes	-
(28)	Modèle conceptuel multidimensionnel	Orienté documents, colonnes	-
(18)	Modèle conceptuel UML (inspiré de Peter Chen)	Général (conception NoSQL)	-

Tableau 2.1 – Étude comparative des approches de modélisation NoSQL

À la suite de l'analyse des études présentées dans le tableau [2.1], nous avons pu dégager trois techniques principales récurrentes dans les travaux antérieurs. Afin de structurer notre étude de manière cohérente, les résultats du tableau seront examinés dans le cadre de chacune de ces techniques. Pour chaque approche identifiée, nous mettrons en lumière les contributions associées, en soulignant leurs points forts ainsi que leurs limites.

2.3 Transformations de Modèles Conceptuels vers NoSQL

Grâce à l'analyse effectuée, nous avons pu comprendre comment transformer des modèles conceptuels, tels que les diagrammes de classes UML, en schémas NoSQL. Les auteurs dans (16; 1; 17) proposent un processus de transformation d'un schéma de classes UML vers les modèles orientés colonnes, les modèles orientés documents et les modèles orientés graphes. Où (16) respectent, au plus, deux des caractéristiques du Big Data (Volume, Variété et Vitesse), et mettent également en lumière les aspects de Variété, Volume, Vitesse et Véracité du Big Data chez (17). Par ailleurs, (18) a proposé une méthode de conception d'une base de données NoSQL à partir d'un modèle conceptuel UML inspiré du cadre de Peter Chen. Ces études nous ont permis d'identifier les avantages et les limites de cette technique :

Points forts :

— L'utilisation d'UML offre une méthode visuelle et organisée pour la conception des structures de données NoSQL, ce qui peut faciliter la compréhension et la communication au sein de l'équipe.

— Cette technique vise à créer un modèle intermédiaire qui peut être transformé en différents types de bases de données NoSQL.

— L'intégration des règles métier lors de la transformation pour assurer la cohérence des données dans le système NoSQL final.

— L'importance d'une modélisation conceptuelle précise avant de passer à la conception physique de NoSQL, ce qui peut conduire à des structures de données plus efficaces.

Points faibles :

— Si l'utilisateur dispose déjà d'un système hérité basé sur des bases de données SQL, certains proposent une étape supplémentaire consistant à transformer d'abord le schéma SQL existant en un modèle UML, puis à convertir l'UML en NoSQL, ce qui peut être complexe et chronophage.

— La focalisation exclusive sur UML.

2.4 Approches de migration SQL vers NoSQL

Plusieurs études ont cherché à développer des outils facilitant la migration. Alors que d'autres approches dépendent de schémas SQL relationnels (21; 22; 23; 29; 31) transforment des schémas SQL pour cibler les modèles orientés documents, les modèles colonnes et les modèles orientés graphes, avec une note des caractéristiques de Volume et Variété du Big Data pour (22), quant à (26), ils se sont concentrés sur les schémas SQL relationnels pour des sorties NoSQL générales, ciblant spécifiquement les systèmes de gestion de contenu (CMS). Enfin, (25) a développé une approche en deux étapes pour la migration vers HBase, combinant automatisation des données et transformation du schéma. Ces études nous ont permis d'identifier les avantages et les limites de cette technique :

Points forts :

— Simplification et automatisation du processus de transformation, permettant un gain de temps et d'effort considérable.

— Prise en compte des relations entre les tables SQL afin d'assurer une transition fluide et une structure de données NoSQL optimisée.

— Méthodes (algorithmes) capables de traiter les données sans connaissance préalable du schéma.

— Préservation des contraintes d'intégrité lors de la migration grâce à l'ingénierie inverse des schémas SQL.

Points faibles :

— Tous les outils ne prennent pas en charge efficacement les schémas SQL complexes impliquant de nombreuses relations et contraintes.

— Limitations possibles à certains types spécifiques de bases NoSQL.

2.5 Stratégies de dénormalisation dans le contexte NoSQL

Il s'est avéré qu'une stratégie est adoptée pour améliorer les performances dans les bases de données NoSQL, consistant principalement à réduire la dépendance aux opérations de jointure, connue sous le nom de stratégie de dénormalisation. Dans ce contexte, l'étude (26) a proposé une méthodologie indépendante de dénormalisation et de migration des schémas SQL vers des environnements NoSQL, ciblant spécifiquement les systèmes de gestion de contenu (CMS). Leur approche vise à adapter les structures de données

existantes aux modèles NoSQL sans nécessiter une refonte complète, permettant ainsi des améliorations significatives en termes d'efficacité. Parallèlement, les travaux de (27; 28) ont proposé un processus de transformation avec un ensemble de règles de mapping des modèles multidimensionnels vers les modèles orientés colonnes et les modèles orientés documents. De même, l'étude (29) a développé un ensemble de règles de transformation pour migrer les schémas relationnels vers différents modèles NoSQL, en tenant compte des relations fondamentales entre les données. Ces approches, prises dans leur ensemble, soulignent le rôle essentiel que joue la dénormalisation dans la transition vers les systèmes NoSQL. Cette analyse nous a permis d'identifier les principaux avantages et inconvénients de cette technique :

Points forts :

— L'objectif principal de la dénormalisation est de réduire le besoin d'opérations de jointure, ce qui accélère les opérations de lecture et de requête dans NoSQL.

— Elle permet d'atteindre des gains de performance significatifs, notamment en réduisant les temps de réponse.

— En consolidant les données pertinentes dans un même document, elle facilite l'accès rapide aux informations, ce qui est essentiel dans les environnements Big Data.

Points faibles :

— La dénormalisation entraîne la duplication des données, ce qui peut augmenter le volume de stockage et poser des problèmes de cohérence.

— Elle peut conduire à des modèles de données moins flexibles et plus difficiles à maintenir.

— La mise à jour des données dupliquées devient plus complexe, car elle nécessite des mécanismes de synchronisation rigoureux.

Conclusion

À partir de l'analyse du tableau [2.1] et des trois techniques principales qui en ont émergé, plusieurs constats ont pu être formulés concernant les approches existantes de transformation vers les bases de données NoSQL.
Il apparaît que la plupart des travaux ne définissent pas clairement les règles de transformation à appliquer lors du passage des modèles relationnels vers les modèles NoSQL. De plus, ces contributions se concentrent souvent sur un seul type de modèle source, ce qui en limite la généralisation.

L'analyse a également révélé une prédominance des solutions orientées vers les bases documentaires ou en colonnes, au détriment des bases orientées graphes. Enfin, bien que la question de la dénormalisation soit essentielle dans les environnements NoSQL, elle reste peu approfondie et rarement intégrée de manière systématique.
Ces constats soulignent la nécessité d'une approche plus structurée, intégrant à la fois des règles de transformation explicites et des stratégies de dénormalisation adaptées, ce que ce travail vise précisément à proposer.

Chapitre 3

Conception du prototype de transformation

Introduction

Dans le chapitre précédent, une analyse approfondie des travaux de recherche a permis de dégager trois techniques principales de transformation des bases de données relationnelles vers les systèmes NoSQL. Cette étude a mis en lumière les approches adoptées, les types de modèles ciblés, ainsi que les limites méthodologiques observées, notamment l'absence de formalisation des règles de conversion et la sous-exploitation des stratégies de dénormalisation.

Dans ce contexte, le présent chapitre vise à concevoir et mettre en œuvre un prototype de transformation automatique d'un schéma SQL vers un modèle NoSQL, en particulier MongoDB. Ce prototype s'appuie sur les constats précédents et intègre explicitement des règles de dénormalisation afin d'optimiser les performances dans un environnement NoSQL.

3.1 Architecture du prototype

Dans cette partie, nous proposons un prototype représentant le mécanisme adopté pour la transformation des schémas de bases de données relationnelles (SQL) vers un modèle orienté document utilisant MongoDB. Ce prototype illustre les étapes fondamentales par lesquelles les données passent au cours du processus de conversion, depuis l'analyse structurelle du schéma relationnel jusqu'à la génération d'un schéma documentaire intégré.

3.1.1 Définition du prototype

La figure 3.1 illustre le fonctionnement du modèle proposé pour la conversion d'un schéma relationnel SQL vers MongoDB. Le processus commence par l'analyse du schéma relationnel afin d'en extraire un modèle logique intermédiaire. Ce modèle est ensuite enrichi par l'application de règles de dénormalisation. Enfin, un générateur dédié produit automatiquement le schéma physique compatible avec MongoDB, sous forme de documents JSON.

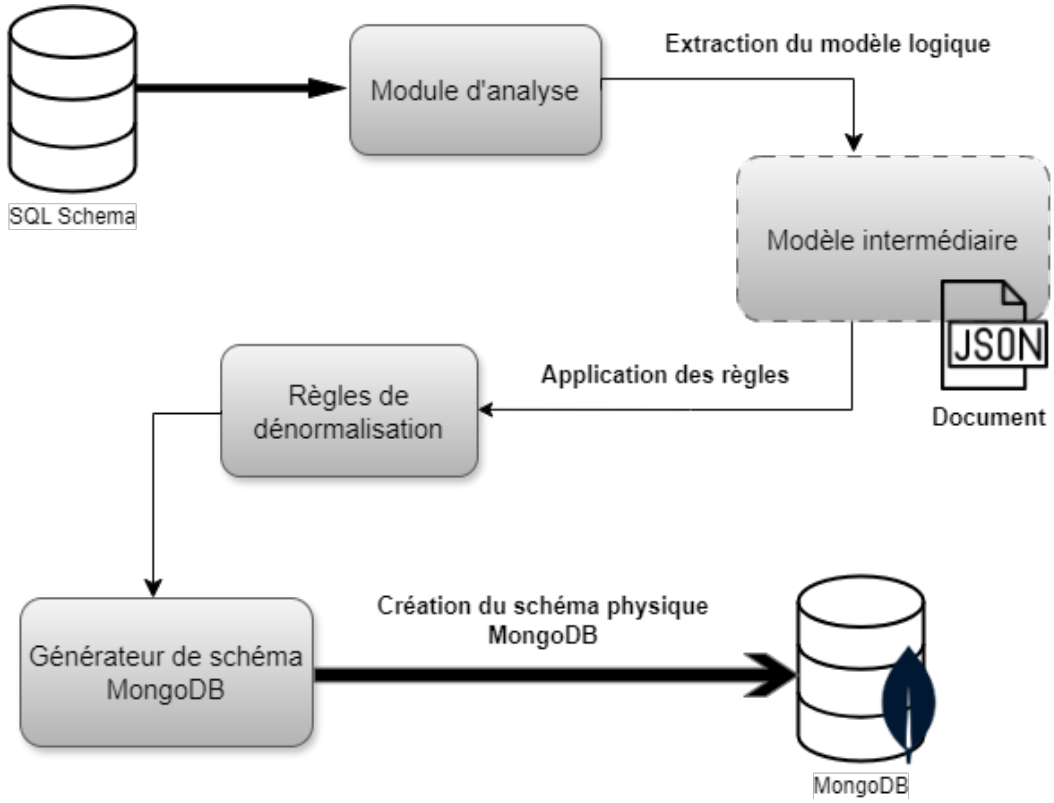


Figure 3.1 – Architecture de la conversion SQL vers MongoDB

3.1.2 Description des composants architecturaux

— **Script SQL (entrée utilisateur)**
L'utilisateur fournit initialement un schéma relationnel écrit en langage SQL, le plus souvent en utilisant les commandes "CREATE TABLE". Ce schéma contient les entités (tables), les attributs (colonnes), et les relations qui décrivent la base de données source.

— **Analyseur SQL (unité d'analyse syntaxique et sémantique)**
Les commandes SQL sont lues et comprises pour en extraire les informations essen-

tielles, telles que les tables, les clés primaires et étrangères, et les types de relations entre les entités. Cette étape constitue le point de départ pour la construction du modèle intermédiaire.

— **Modèle intermédiaire**
Après l'extraction des informations issues de l'analyse, l'étape du modèle intermédiaire consiste à structurer et organiser les données de manière uniforme et cohérente avant l'application de la dénormalisation. Ce modèle est représenté sous forme d'objets JSON décrivant les tables et les relations.

— **Unité de dénormalisation**
l'unité de dénormalisation correspond à chaque relation détectée entre des tables SQL qui peut être transformée en une structure plus adaptée au modèle NoSQL. À cette étape, l'outil identifie automatiquement les segments de données susceptibles d'être fusionnés (via l'imbrication) ou laissés séparés (via le référencement), selon la nature de la relation (1 :1, 1 :N, N :M), la taille des entités et leur fréquence d'accès, en réduisant le nombre de requêtes complexes et en augmentant la vitesse d'accès à l'information.

— **Générateur de schéma MongoDB**
Le générateur de schéma MongoDB est la dernière étape qui dédiée à la création du modèle physique de la base de données NoSQL. Il a pour rôle principal de formaliser les structures finales des collections MongoDB à partir d'un modèle dénormalisé, en assurant leur conformité avec les principes du stockage orienté document.

3.1.3 Présentation de MongoDB

— **Définition de MongoDB**
MongoDB est une base de données NoSQL open source, basée sur un modèle non relationnel orienté document. Elle permet de stocker différents types de données, même en grands volumes, grâce au format BSON, une version binaire de JSON qui prend en charge plus de types de données.(32)

— **Comment fonctionne MongoDB ?**
MongoDB fonctionne en stockant les données sous forme de documents organisés dans des collections, au lieu des tables et lignes utilisées dans les bases relationnelles. Chaque collection regroupe plusieurs documents, lesquels sont constitués de paires clé-valeur et représentent l'unité fondamentale des données. La structure des documents est flexible : on peut y ajouter ou supprimer des champs facilement. Chaque document peut avoir une clé primaire unique, et ses valeurs peuvent être de différents types, y compris d'autres documents, des tableaux ou des tableaux de

documents.(32)

— **Principales caractéristiques de MongoDB**

Flexibilité : MongoDB possède une architecture de schéma dynamique qui fonctionne avec des données et un stockage non structurés. Les données étant stockées dans des documents flexibles de type JSON, le schéma de base de données n'a pas besoin d'être prédéfini et peut être modifié de manière dynamique sans entraîner d'arrêt.(32)

Partage des données (sharding) : MongoDB permet une mise à l'échelle horizontale grâce à un processus appelé sharding. Le sharding permet de diviser les données d'un grand ensemble de données et de les répartir sur plusieurs serveurs. Si un serveur ne peut pas gérer une charge importante de données, celles-ci peuvent être automatiquement divisées et distribuées sans interrompre le traitement.(32)

Performances optimisées : Pour améliorer les performances, MongoDB charge les données en mémoire vive (RAM), ce qui accélère les opérations de lecture et d'écriture. Ce modèle non relationnel permet également une exécution des requêtes plus simple et plus rapide que dans les bases de données traditionnelles.(32)

3.2 Principe de fonctionnement du prototype

Afin de mieux cerner les fondements techniques de c'outil développé, il convient de présenter son architecture générale et la séquence des opérations qu'il effectue. L'outil proposé est composé de quatre modules principaux, fonctionnant selon une séquence claire et organisée. Le processus commence par le module d'analyse, qui reçoit le schéma SQL et en extrait le modèle logique. Par la suite, ce modèle est converti en un modèle intermédiaire représenté de manière flexible au format JSON. Vient ensuite le module d'application des règles de dénormalisation, qui adapte le modèle intermédiaire pour qu'il soit compatible avec la structure de MongoDB, en déterminant les relations à intégrer (embedding) ou à séparer (référencement). Enfin, le module de génération de schéma se charge de créer le modèle physique MongoDB prêt à être exécuté.
Dans cette section, on va illustrer le fonctionnement de notre prototype à travers un cas d'étude.

3.2.1 Définition des règles de transformation

En se fondant sur l'analyse comparative des travaux antérieurs, et sur la compréhension acquise par l'étude des caractéristiques du modèle relationnel et des bases de données orientées document, il est possible d'extraire un ensemble de règles générales qui guident

le processus de transformation d'un modèle relationnel (SQL) vers un modèle orienté document (MongoDB). Ces règles visent à garantir une structure flexible, efficace et évolutive, en prenant en considération la nature des relations entre les entités, le volume des données, ainsi que les motifs de requête et d'utilisation. Ci-après, nous présentons les plus importantes de ces règles(33) :

Règle 1 : Transformation d'une relation 1 :1

Lorsque deux entités entretiennent une relation un-à-un (1 :1), les données peuvent être intégrées (imbriquées) dans un seul document. Cette approche permet de simplifier l'accès aux données liées.

Exemple : une entité *Utilisateur* liée à une entité *Profil* peut être modélisée comme un seul document dans MongoDB où le profil est un sous-document de l'utilisateur.

Règle 2 : Traitement d'une relation de type 1 :N

Les relations un-à-plusieurs (1 :N) sont représentées par des *références*, sans prise en compte du contexte d'utilisation. Cela consiste à stocker dans la collection correspondant au côté « plusieurs » un identifiant pointant vers le document du côté « un ». **Exemple** : une entité *Client* liée à plusieurs *Commandes*. Deux collections distinctes sont générées :

- `clients` : { _id, nom, ... }
- `commandes` : { _id, produit, client_id, ... }

Le champ `client_id` établit la liaison avec la collection `clients`.

Règle 3 : Transformation d'une relation N :M

Les relations plusieurs-à-plusieurs (N :M) nécessitent souvent une structure intermédiaire dans le modèle relationnel. En MongoDB, plusieurs stratégies existent, dont :

- Références croisées : si les deux entités doivent rester indépendantes.

Exemple : une relation entre *Étudiants* et *Cours*. Chaque étudiant peut avoir une liste d'identifiants de cours suivis, ou inversement.

Voici le tableau [3.1] suivant qui représente la différence entre ces règles :

Type de relation	**Collections MongoDB créées**	**Structure attendue du document**
1 :1	Embedding (inclusion d'un document dans un autre)	`{ _id: <int>, nom: <string>, profil: { bio: <string> } }`
1 :N	Référence (stockage de l'identifiant dans un autre document)	`{ _id: <int>, commande_id: <int>, produit: <string>, quantite: <int> }`
N :M	Collection intermédiaire avec références croisées	`{ etudiant_id: <int>, cours_id: <int> }`

Tableau 3.1 – Création des structures MongoDB selon le type de relation

À partir de cette conception architecturale générale, nous allons, dans la partie suivante, appliquer ces étapes méthodiques à un cas d'étude simplifié, afin de démontrer le fonctionnement du prototype dans un contexte pratique.

3.2.2 Étude de cas

Dans cette partie, nous présenterons une étude de cas simplifiée d'un système d'information dédié à la gestion des ventes automobiles au sein d'une entreprise de distribution. Ce système vise à organiser les relations entre les clients, les véhicules, les dépôts et les commerciaux. Cet exemple reflète une réalité suffisante, permettant de tester les étapes de transformation d'un modèle relationnel vers un modèle orienté document de manière pratique et claire.
Nous débuterons par une analyse préliminaire, à partir de laquelle nous produirons un Modèle Conceptuel de Données (MCD), puis le transformerons en un Modèle Logique de Données (MLD). Enfin, nous appliquerons le modèle proposé pour convertir cette structure en un modèle documentaire utilisant MongoDB.

1- Script SQL :

Nous débuterons par une analyse préliminaire, à partir de laquelle nous produirons un Modèle Conceptuel de Données (MCD), puis le transformerons en un Modèle Logique de Données (MLD). Enfin, nous appliquerons le modèle proposé pour convertir cette structure en un modèle documentaire utilisant MongoDB.

• **Le Modèle Conceptuel de Données (MCD)**

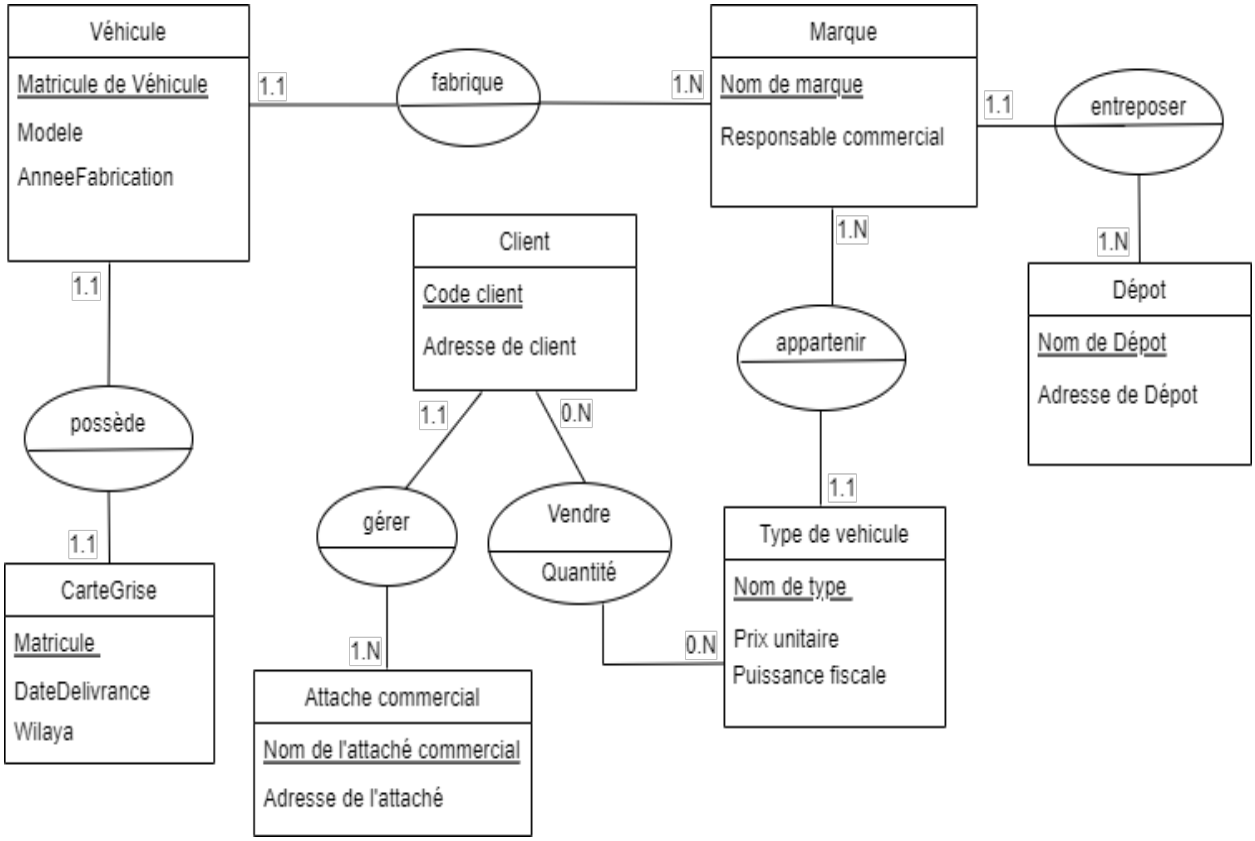


Figure 3.2 – Modèle Conceptuel de Données (MCD) du système de gestion des ventes automobile

Partant du Modèle Conceptuel de Données (MCD) précédent, qui définit les différentes entités et relations dans le domaine d'étude, nous le convertissons en un Modèle Logique de Données (MLD) représentant la structure logique de la base de données relationnelle. Cette conversion implique la spécification des tables, des clés primaires et étrangères, ainsi que la représentation des relations d'une manière compatible avec les systèmes de gestion de bases de données relationnelles.

• **Le Modèle Logique de Données (MLD)**

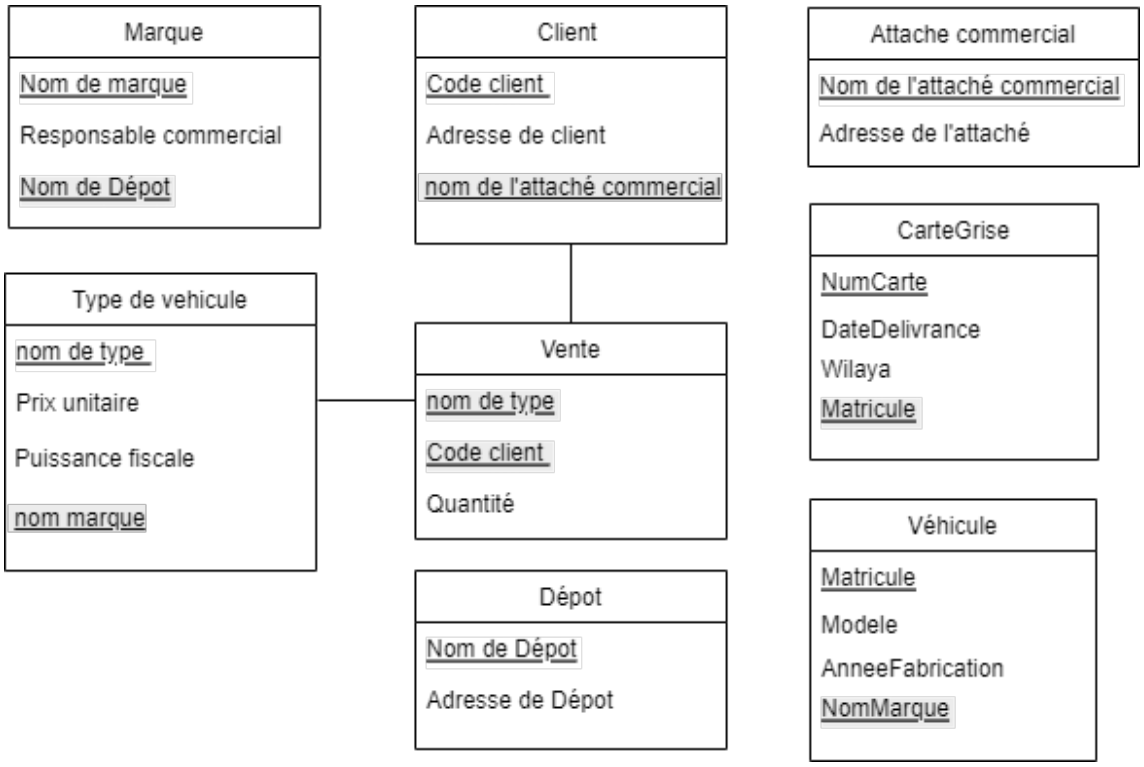


Figure 3.3 – Modèle Logique de Données (MLD) dérivé du Modèle Conceptuel de Données

Chaque client est associé à un attaché commercial par une clé étrangère. De même, chaque type de véhicule appartient à une marque spécifique, laquelle est liée à un dépôt. Les transactions de vente sont enregistrées dans la table, qui relie le client au type de véhicule vendu, en documentant la quantité.
Chaque véhicule est représenté par une table liée à sa marque, et il est associé à une seule carte grise. Ce modèle repose sur l'utilisation de clés primaires et étrangères afin d'assurer l'intégrité des données et la cohérence des relations entre les tables.

2- Analyseur SQL :

Le processus d'analyse commence par la lecture des commandes SQL brutes fournies en entrée. L'analyseur syntaxique examine d'abord la structure formelle de chaque instruction (par exemple : `CREATE TABLE`, `PRIMARY KEY`, `FOREIGN KEY`, etc.) en s'appuyant sur une grammaire définie du langage SQL. Cette étape permet de détecter la validité syntaxique et de segmenter les éléments constitutifs du schéma : noms des tables, attributs, types de données, et contraintes associées.

Ensuite, l'analyseur sémantique entre en jeu pour donner un sens contextuel aux éléments extraits. Il vérifie, par exemple, la cohérence des types de données, l'existence des

relations entre les tables via les clés étrangères, ou encore la conformité des contraintes définies. Cette double analyse aboutit à une représentation logique interne du schéma relationnel, sous forme de structure de données exploitable par les modules suivants.

Ce modèle logique contient non seulement la hiérarchie des entités et leurs attributs, mais aussi les relations cardinales implicites, ce qui facilitera l'application des règles de dénormalisation dans l'étape suivante.

3- Modèle intermédiaire :

Le modèle intermédiaire constitue une étape essentielle dans le processus de transformation. Il permet d'identifier plusieurs éléments fondamentaux tels que les entités (tables), leurs attributs, les types de données, ainsi que les clés primaires et étrangères.

Ce modèle ne se limite pas à une simple traduction des éléments SQL, mais joue également un rôle central dans la compréhension des dépendances entre les entités.

À partir de cette représentation, le système procède à appliquer les quatre règles fondamentales mentionnées précédemment. C'est sur la base de ces règles que l'unité de dénormalisation applique, à ce stade, les stratégies d'intégration ou de séparation des données.

4- Unité de dénormalisation

Pour chaque relation, l'unité détermine s'il est plus approprié d'opter pour une intégration des données (embedding) ou pour une séparation par référence (referencing), en se basant sur le type de relation identifié précédemment. Dans ce qui suit, nous présentons les particularités de chacune de ces deux approches :

• Documents Intégrés :

Les documents intégrés sont des documents stockés à l'intérieur d'autres documents, formant ainsi une structure imbriquée et hiérarchique. Cette approche permet à MongoDB de stocker les données connexes ensemble, ce qui facilite la récupération de l'ensemble des informations en une seule requête. Cette méthode utilise le support de MongoDB pour les structures de documents complexes.

- Ses avantages : on retrouve la possibilité de réaliser des opérations atomiques pour garantir la cohérence des données, une amélioration des performances en lecture grâce à la récupération de toutes les données pertinentes en une seule requête, ainsi qu'un renforcement de la localisation des données (Data Locality) qui optimise les performances lors d'accés fréquents aux données interconnectées.

- Cas d'utilisation :

Relations Un-à-Un : Lorsqu'un document est directement lié à un autre (par exemple,

le profil d'un utilisateur et ses paramètres utilisateur).

- Exemple :

Relation :

La relation 1 :1 entre cartegrise et vehicule utilise la stratégie d'imbrication, avec le champ "Matricule".

• Documents Référencé :

Les documents référencés gèrent les relations en incluant une référence (généralement un ObjectId) vers un autre document stocké dans une collection différente. Cette approche sépare les données liées en documents et collections distincts.

- Ses avantages : on retrouve leur capacité à offrir des modèles de données plus flexibles et normalisés, facilitant ainsi la gestion des relations complexes. Ils permettent également une meilleure gestion de la taille des documents, les maintenant plus petits et aidant à éviter la limite de 16 Mo, et contribuent à une réduction de la redondance dans les données en stockant les informations liées dans des documents séparés.

- Cas d'utilisation :

Relations Un-à-Plusieurs : Par exemple, stocker plusieurs commentaires sous un seul article de blog lorsque les commentaires ne sont pas trop volumineux.

Relations Plusieurs-à-Plusieurs : Par exemple, un étudiant peut s'inscrire à plusieurs cours, et chaque cours peut avoir de nombreux étudiants. Les références sont idéales pour ces types de relations.

- Exemple :

Relation :

La relation N :1 entre marque et depot utilise la stratégie de référencement, avec le champ "NomDepot".

...

Relation :

La relation N :M entre vente et client utilise la stratégie de référencement, avec le champ "CodeClient".

5- Générateur de schéma MongoDB

Le générateur de schéma MongoDB constitue l'unité chargée de produire la structure finale des collections à partir du modèle dénormalisé. Il interprète les décisions prises lors de l'étape de dénormalisation (embedding ou référencement) et génère, pour chaque entité, une représentation conforme aux exigences de MongoDB. Cette unité veille à organiser les documents JSON selon une logique cohérente, en tenant compte des relations, des types de données et de la structure hiérarchique optimale. **exemple d'un seul entité :**


```
{
  "marque": {
    "fields": {
      "NomMarque": {
        "type": "String",
        "required": true,
        "primaryKey": true
      },
      "ResponsableCommercial": {
        "type": "String",
        "required": false
      },
      "NomDepot": {
        "type": "String",
        "required": false,
        "foreignKey": true
      }
    },
    "relationships": [
      {
        "relationType": "N:1",
        "with": "depot",
        "foreignField": "NomDepot",
        "strategy": "referencing",
        "description": "Référence vers la collection depot via le champ NomDepot
        pour éviter la duplication et faciliter la mise à jour."
      }
    ]}
}
```

3.3 Representation de schéma MongoDB

La représentation finale du schéma MongoDB se fait sous forme de documents JSON structurés. Chaque entité du modèle relationnel est convertie en un document, associé à d'autres entités soit par une stratégie d'intégration (embedding), soit par une stratégie de référence (referencing).

Cette représentation permet de visualiser clairement la structure cible, en mettant en évidence les entités, les types de données, les clés, ainsi que les dépendances entre collections, tout en illustrant la logique globale de transformation. Nous présentons ci-dessous un extrait simplifié regroupant quelques entités clés. Chaque relation y est associée à une stratégie de transformation choisie selon le type de dépendance observée :

- **Fichier SQL :**

```
CREATE TABLE Marque (
  NomMarque VARCHAR(50) PRIMARY KEY,
  ResponsableCommercial VARCHAR(100),
  NomDepot VARCHAR(50),
  FOREIGN KEY (NomDepot) REFERENCES Depot(NomDepot)
);

CREATE TABLE Vehicule (
  Matricule VARCHAR(10) PRIMARY KEY,
  Modele VARCHAR(50),
  AnneeFabrication INT,
  NomMarque VARCHAR(50),
  FOREIGN KEY (NomMarque) REFERENCES Marque(NomMarque)
);

CREATE TABLE CarteGrise (
  NumCarte VARCHAR(10),
  DateDelivrance DATE,
  Wilaya VARCHAR(50),
  Matricule VARCHAR(10) PRIMARY KEY,
  FOREIGN KEY (Matricule) REFERENCES Vehicule(Matricule)
);
```

- **Fichier JSON :**

```
{
 "typedevehicule": {
    "fields": { ... },
    "relationships": [
      {
        "relationType": "N:1",
        "with": "marque",
        "strategy": "referencing",
        "description": "Référence vers la collection marque via le champ NomMarque
        pour éviter la duplication et faciliter la mise à jour."
      }
    ]
  },
  "cartegrise": {
    "fields": { ... },
    "relationships": [
      {
        "with": "vehicule",
        "strategy": "embedding"
        "description": "Les données complètes de vehicule sont intégrées dans
        le document cartegrise sous forme d'objet imbriqué."
      }
    ]
  }
}
```

3.4 Analyse et discussion des résultats

Après avoir défini et conçu l'architecture du prototype, cette section présente les résultats obtenus suite à la mise en œuvre de l'outil. Elle décrit l'interface utilisateur développée, les fonctionnalités proposées, ainsi que les résultats concrets de la transformation des schémas SQL vers MongoDB.

3.4.1 Développement de l'outil

Le prototype de transformation a été développé et testé dans un environnement web. Pour sa mise en œuvre, une interface utilisateur simple a été conçue en HTML/CSS/JavaScript, permettant de charger des fichiers SQL, d'analyser leur structure, puis de générer une représentation appropriée dans MongoDB. Le traitement principal a été assuré par un script écrit en Python.

Nous avons ainsi créé un outil simple de transformation des schémas SQL vers MongoDB, accessible via l'interface conviviale conçue.

3.4.2 Fonctionnement de l'interface

Les utilisateurs peuvent soit saisir manuellement le code du schéma SQL, soit importer un fichier au format `.sql` pour lancer le processus de transformation. Le système analyse les entrées, extrait les tables, les attributs, les clés primaires et étrangères, identifie les types de relations (1 :1, 1 :N, N :M) ainsi que la stratégie de dénormalisation choisie, puis génère automatiquement un schéma documentaire adapté à MongoDB.

Le résultat final met en évidence les types de données, les relations, ainsi que l'application des stratégies de dénormalisation dans un environnement NoSQL. Les captures d'écran suivantes illustrent les différentes étapes de fonctionnement de l'outil ainsi que les résultats générés après la transformation.

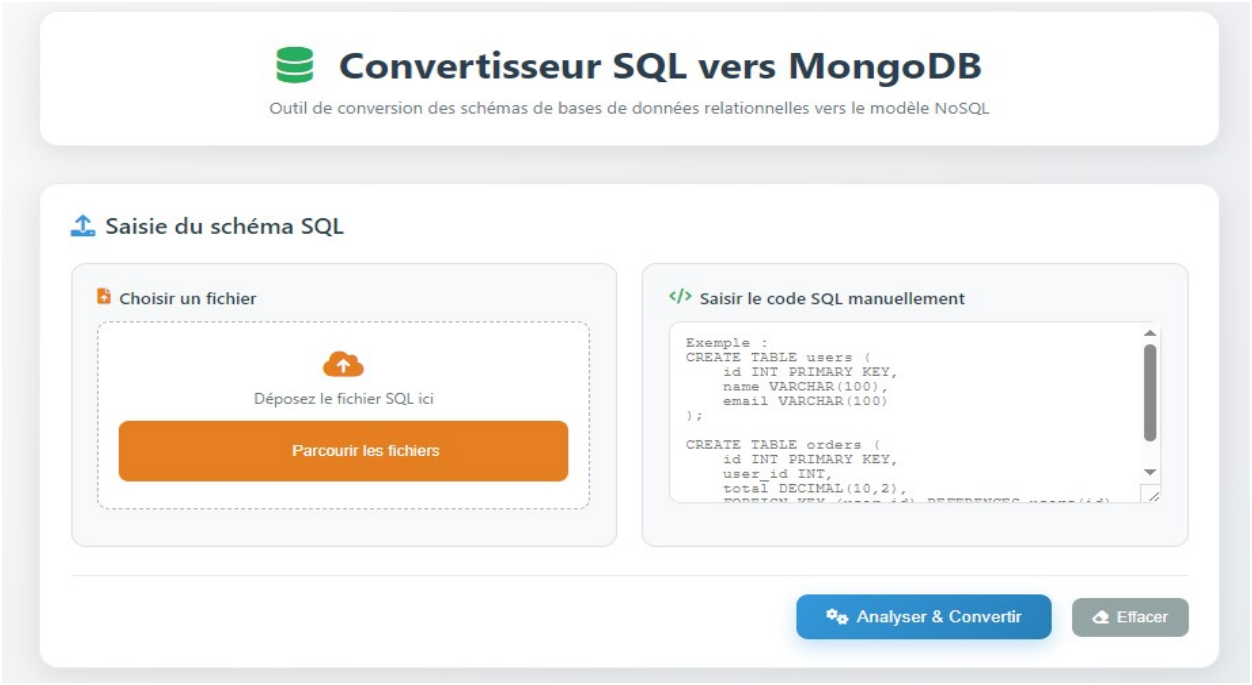


Figure 3.4 – Interface de l'outil de transformation SQL vers MongoDB

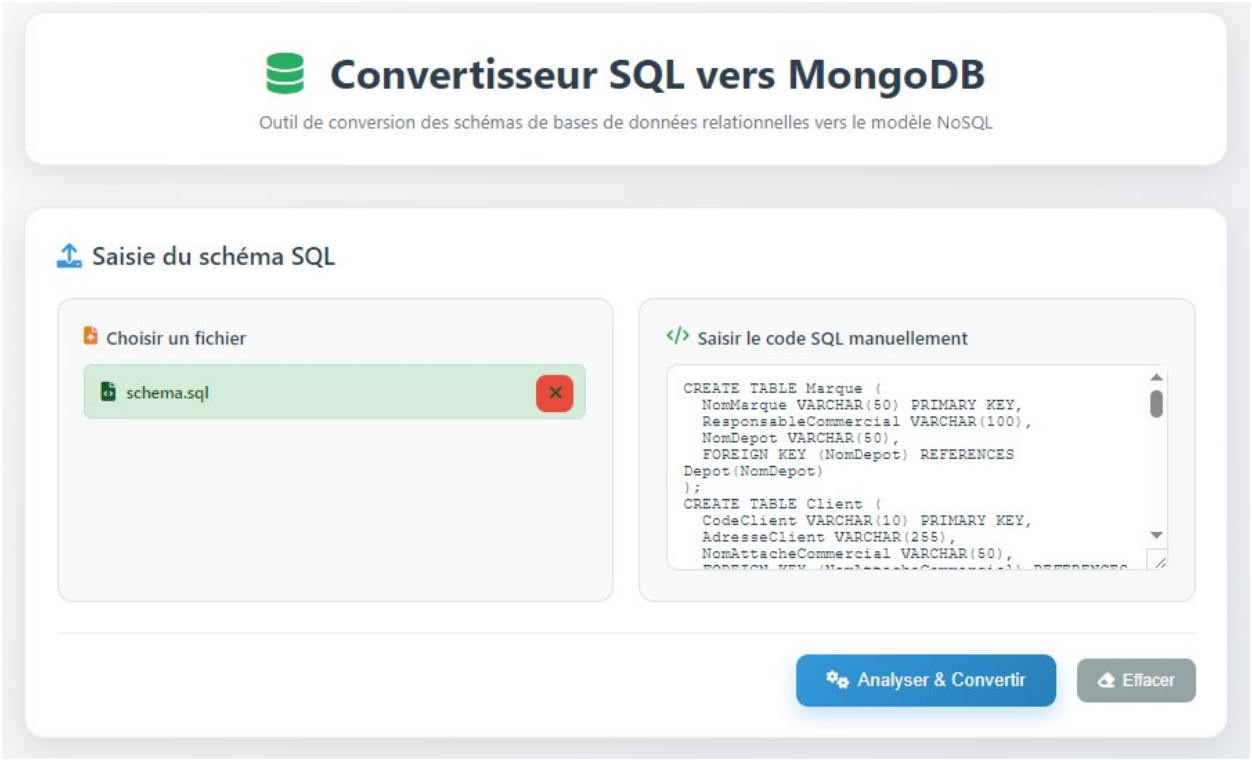


Figure 3.5 – Interface de l'outil en cours d'exécution

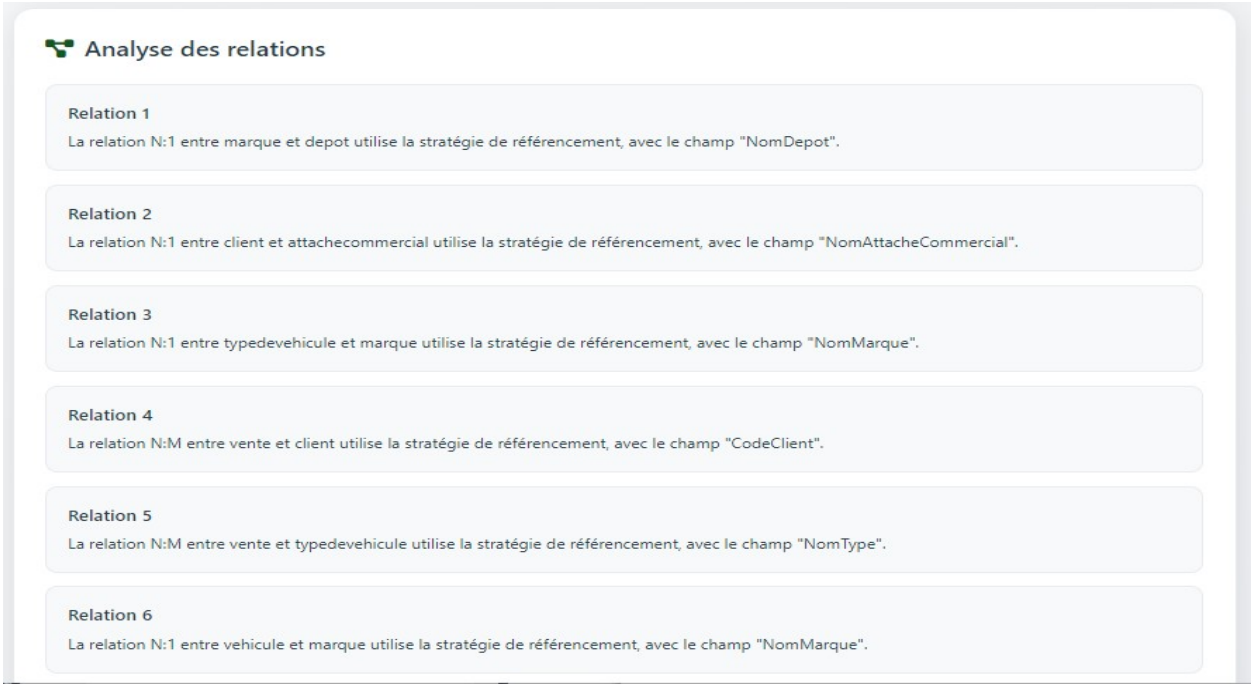


Figure 3.6 – Interface de l'outil affichant les types de relations détectées (1 :1, 1 :N, N :M)

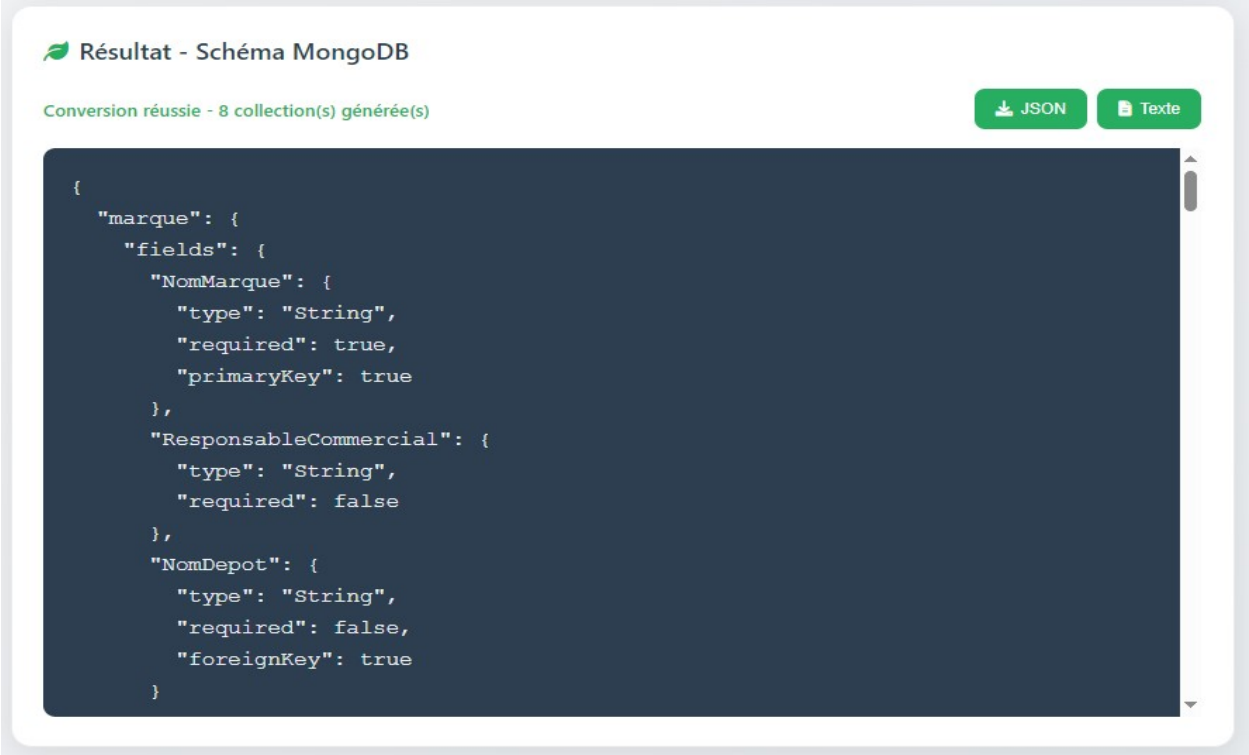


Figure 3.7 – Résultat final de la conversion affiché en format "JSON"

L'outil permet également de télécharger le schéma MongoDB généré au format `.json`, afin de faciliter son exploitation ou son intégration dans d'autres environnements.

En outre, l'outil offre une interactivité permettant aux développeurs, étudiants et chercheurs de comprendre et de visualiser clairement les différences entre les bases de données relationnelles (SQL) et les bases de données NoSQL.

Afin d'évaluer l'efficacité de la transformation, une comparaison simplifiée entre le modèle relationnel initial et le modèle MongoDB généré est présentée ci-dessous.

Critère	Modèle SQL	Modèle MongoDB
Structure des données	Tables (relations), lignes, colonnes	Documents JSON, collections
Nombre de tables / collections	6 tables	8 collections
Types de relations	1 :1, 1 :N, N :M	Imbrication ou Références simples et croisées
Structure des données	Tables (relations), lignes, colonnes	Documents JSON, collections
Données dupliquées	Aucune	Minimale (via stratégie d'optimisation)
Gestion des relations N :M	Table intermédiaire	Collection intermédiaire avec références croisées
Requêtes complexes (jointures)	Oui (JOIN)	Non, remplacées par des parcours de documents

Tableau 3.2 – Comparaison simplifiée entre le modèle relationnel SQL et le modèle MongoDB obtenu

- **Accès à l'outil :**

Le code source de l'outil, ainsi qu'une démonstration de son fonctionnement, est disponible à l'adresse suivante : https://github.com/nafgue2002/sql-to-mongodb-tool

Conclusion

Dans ce chapitre, nous avons présenté le prototype simple de transformation, développé pour automatiser la migration des schémas relationnels vers le modèle documentaire de MongoDB. Nous avons expliqué son architecture, son principe de fonctionnement, et illustré le processus à travers un exemple concret, mettant en évidence la capacité de l'outil à interpréter les relations et à générer des documents adaptés. Nous avons ensuite décrit l'interface utilisateur. Les résultats obtenus sont satisfaisants et confirment la faisabilité de l'approche proposée.

Conclusion Générale

Bilan du travail réalisé

Les résultats obtenus à travers le prototype développé sont conformes aux attentes définies au préalable. L'outil a permis de convertir efficacement des schémas relationnels simples en représentations documentaires adaptées à MongoDB, tout en prenant en compte les relations, les types de données et les stratégies de dénormalisation adoptées.

Cette étude a mis en évidence plusieurs résultats clés :

- La capacité de l'outil à extraire automatiquement les tables, les champs, les clés primaires et étrangères à partir de fichiers SQL.
- La reconnaissance précise des types de relations (1 :1, 1 :N, N :M) et leur modélisation selon des stratégies appropriées dans un environnement NoSQL.
- Un support efficace pour l'analyse et le traitement des structures relationnelles classiques.
- La génération automatique d'un schéma documentaire cohérent au format JSON, prêt à être utilisé dans MongoDB.
- Une interface utilisateur claire et conviviale permettant de visualiser en détail les résultats finaux.

Ainsi, cette mise en œuvre concrète confirme l'efficacité de l'approche proposée, tout en mettant en lumière son potentiel pédagogique et applicatif au profit des étudiants, chercheurs et développeurs intéressés par la migration des bases de données SQL vers des environnements NoSQL.

Limites de l'étude

Le prototype développé constitue une version initiale fonctionnelle, centrée sur la transformation des structures relationnelles classiques. Toutefois, certaines limites ont été identifiées :

- L'outil repose sur un ensemble de règles de transformation prédéfinies et rigides.
- Les stratégies de dénormalisation sont choisies selon des heuristiques simples, sans prendre en compte les objectifs métiers ni les particularités d'utilisation des données.

Perspectives de recherches futures

Dans le but de développer l'outil et de le rendre plus performant et flexible, plusieurs améliorations futures peuvent être envisagées, notamment :

- Rendre l'outil plus intelligent :
  - en lui permettant de gérer automatiquement toutes les règles de transformation, sans limitations, en fonction du contenu de la base de données et du contexte d'utilisation ;
  - en élargissant sa capacité à transformer des schémas SQL vers différents types de bases de données NoSQL ;
  - en rendant le choix de la stratégie de dénormalisation plus pertinent, en fonction des particularités d'usage des données.

Bibliographie

[1] Amal Ait Brahim. Approche dirigée par les modèles pour l'implantation de bases de données massives sur des sgbd nosql. https://theses.hal.science/tel-02073342. Accessed 12-05-2025.

[2] Donald D. Chamberlin. Early history of sql. https://ieeexplore.ieee.org/document/6359709?arnumber=6359709, 2012. Accessed 13-03-2025.

[3] Les formes normales des bddrelationnel. https://www.loxodata.com/post/formes-normales/. Accessed 13-03-2025.

[4] GeeksforGeeks. Difference between sql and nosql. https://www.geeksforgeeks.org/difference-between-sql-and-nosql/. Accessed 13-03-2025.

[5] Newsql - wikipedia. https://en.wikipedia.org/wiki/NewSQL?utm_source=chatgpt.com. Accessed 16-03-2025.

[6] Bases de données nosql vs sql | mongodb. https://www.mongodb.com/fr-fr/resources/basics/databases/nosql-explained/nosql-vs-sql. Accessed 13-03-2025.

[7] A brief history of non-relational databases - dataversity. https://www.dataversity.net/a-brief-history-of-non-relational-databases/?utm_source=chatgpt.com. Accessed 15-03-2025.

[8] Qu'est-ce que nosql ? explication des bases de données | google cloud | google cloud. https://cloud.google.com/discover/what-is-nosql?hl=fr. Accessed 16-03-2025.

[9] GeeksforGeeks. Difference between small data and big data. https://www.geeksforgeeks.org/difference-between-small-data-and-big-data/. Accessed 13-03-2025.

[10] GeeksforGeeks. Types of NoSQL Databases - GeeksforGeeks — geeksforgeeks.org. https://www.geeksforgeeks.org/types-of-nosql-databases/. Accessed 11-05-2025.

[11] Blent.ai. Normalisation de données : définitions, exemples. https://blent.ai/blog/a/normalisation-de-donnees-tout-savoir, 2025. Accessed 12-05-2025.

[12] FasterCapital. Avantages et inconvénients de la normalisation. https://fastercapital.com/fr/sujet/avantages-et-inconvÃ‰nients-de-la-normalisation.html. Accessed 14-06-2025.

[13] Airbyte Editorial Team. Data denormalization : What it is and why it's useful - airbyte. https://airbyte.com/data-engineering-resources/data-denormalization. Accessed 19-05-2025.

[14] Difference between SQL and NoSQL - GeeksforGeeks — geeksforgeeks.org. https://www.geeksforgeeks.org/difference-between-sql-and-nosql/. Accessed 12-05-2025.

[15] Sql to nosql : Architecture differences and considerations for migration. https://www.scylladb.com/wp-content/uploads/wp-sql-to-nosql-architectur-differences-considerations-migration-1.pdf. Accessed 12-05-2025.

[16] Fatma Abdelhédi, Amal Ait Brahim, Faten Atigui, and Gilles Zurfluh. Umltonosql : Automatic transformation of conceptual schemas to nosql databases. https://www.semanticscholar.org/paper/UMLtoNoSQL%3A-Automatic-Transformation-of-Conceptual-Abdelh%C3%A9di-Brahim/a071f19ac1a5beac1ed4a6379214b0e1fa283d2b, 2017. Accessed 20-04-2025.

[17] Mohammed ElHabib Maicha, Youcef Ouinten, and Benameur Ziani. Uml4nosql : A novel approach for modeling nosql document-oriented databases based on uml. https://www.cai.sk/ojs/index.php/cai/article/view/2022_3_813, 2022. Accessed 23-04-2025.

[18] Kwang-chul Shin, Chulhyun Hwang, and Hoekyung Jung. Design nosql data model : The proposal is how to use conceptual data model in nosql database design based on. https://www.semanticscholar.org/paper/DESIGN-NoSQL-Data-Model-%3A-The-proposal-is-how-to-in-Shin-Hwang/c832699ca6feabcb6a0262daa8d3c3b1ccebd295, 2017. Accessed 29-04-2025.

[19] Harley Vera, Wagner Boaventura, Maristela Holanda, Valeria Guimaraes, and Fernanda Hondo. Data modeling for nosql document-oriented databases. https://ceur-ws.org/Vol-1478/paper17.pdf, 2015. Accessed 21-04-2025.

[20] Robert T. Mason. Nosql databases and data modeling techniques for a document-oriented nosql database. http://Proceedings.InformingScience.org/InSITE2015/InSITE15p259-268Mason1569.pdf, 2015. Accessed 23-04-2025.

[21] Rajaram Kanchana, Sharma Pankaj, and S. Selvakumar. Dloader : Migration of data from sql to nosql databases. https://www.researchgate.net/publication/366776082_DLoader_Migration_of_Data_from_SQL_to_NoSQL_Databases, 2023. Accessed 29-04-2025.

[22] Gayan Liyanaarachchi, Lakshan Kasun, Malki Nimesha, Kanishka Lahiru, and Anuradha Karunasena. Migdb - relational to nosql mapper. https://www.semanticscholar.org/paper/MigDB-relational-to-NoSQL-mapper-Liyanaarachchi-Kasun/a76b4a0085889a579807e7c31988e31b1d42f132, 2016. Accessed 05-04-2025.

[23] Gansen Zhao, Qiaoying Lin, Libo Li, and Zijing Li. Schema conversion model of sql database to nosql. http://ieeexplore.ieee.org/document/7024609/, 2014. Accessed 21-02-2025.

[24] Alza A. Mahmood. Automated algorithm for data migration from relational to nosql databases. https://www.nahje.com/index.php/main/article/view/NJES21010060, 2018. Accessed 26-03-2025.

[25] Chongxin Li. Transforming relational database into hbase : A case study. https://www.semanticscholar.org/paper/Transforming-relational-database-into-HBase%3A-A-case-Li/1e0e27bb3f117f11b0ffd0ed3f48c5f513680505, 2010. Accessed 13-05-2025.

[26] Chao-Hsien Lee and Yu-Lin Zheng. Sql-to-nosql schema denormalization and migration : A study on content management systems. http://ieeexplore.ieee.org/document/7379485/, 2015. Accessed 21-02-2025.

[27] Max Chevalier, Mohammed El Malki, Arlind Kopliku, Olivier Teste, and Ronan Tournier. Implementing multidimensional data warehouses into nosql :. http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0005379801720183, 2015. Accessed 13-05-2025.

[28] Rania Yangui, Ahlem Nabli, and Faiez Gargouri. Automatic transformation of data warehouse schema to nosql data base : Comparative study. https://www.sciencedirect.com/science/article/pii/S1877050916319391, 2016. Accessed 14-05-2025.

[29] Obaid Alotaibi and Eric Pardede. Transformation of schema from relational database (rdb) to nosql databases. https://www.mdpi.com/2306-5729/4/4/148, 2019. Accessed 10-04-2025.

[30] Tatyana S. Ivanova and Evgeniy A. Ivanov. Research and development of the method of investigating the possibility of transformation relational databases to nosql. https://ieeexplore.ieee.org/document/9396363/, 2021. Accessed 27-02-2025.

[31] Alae El Alami and Mohamed Bahaj. Migration of a relational databases to nosql : The way forward. http://ieeexplore.ieee.org/document/7905665/, 2016. Accessed 15-04-2025.

[32] Pure Storage. Qu'est-ce que mongodb ? et comment fonctionne-t-elle ? https://www.purestorage.com/fr/knowledge/what-is-mongodb.html. Accessed 14-06-2025.

[33] MongoDB Inc. Model one-to-one relationships with embedded documents. https://www.mongodb.com/docs/manual/tutorial/model-embedded-one-to-one-relationships-between-documents/, 2024. Accessed 27-06-2025.