Université Hassan 1er
Faculté des Sciences et Techniques
Settat

N° d'ordre ……année 2021

Centre d'Etudes Doctorales
Sciences et Techniques Sciences Médicales

Formation doctorale
Mathématiques, Informatique et Applications

Conception d'une Architecture de Migration entre des Bases de Données Diverses : XML, RDB, NoSQL

Mémoire de thèse de Doctorat

Présentée et soutenue publiquement le 6 juillet 2021

Par

Zakaria BOUSALEM

Sous la direction de :

M. Ilias CHERTI

En vue de l'obtention de

Doctorat National de l'Université Hassane 1er

(Spécialité : Informatique)

Composition du jury

Pr. Ahmed ERRKIK	Faculté des Sciences et Techniques, Settat	Président
Pr. Mohamed BAHAJ	Faculté des Sciences et Techniques, Settat	Rapporteur
Pr. Abdellatif EL AFIA	École Nationale Supérieure d'Informatique et d'Analyse des Systèmes, Rabat	Rapporteur
Pr. Mohammed Chaouki ABOUNAIMA	Faculté des Sciences et Techniques, Fès	Rapporteur
Pr. Mohamed Abdou ELOMARY	Faculté des Sciences et Techniques, Settat	Examinateur
Pr. Ilias CHERTI	Faculté des Sciences et Techniques, Settat	Directeur de thèse

Laboratoire Mécanique, Informatique, Electronique et Télécommunications

Résumé

L’objectif principal de cette thèse est de contribuer à la conception et le développement des architectures de migration et de mapping entre trois types de bases de données à savoir XML, les bases de données relationnelles et les bases de données NoSQL orientées colonnes.

Cette thèse s’articule autour de deux axes : Au niveau du premier axe, nous avons proposé une approche permettant de stocker et interroger des documents XML à l'aide d’une base de données relationnelle. Cette approche supporte efficacement les modifications structurelles de l'arbre XML (i.e. insertion, modification et suppression).

Au niveau du deuxième axe, nous avons mis en place une méthode de migration d’une base de données relationnelle vers une base de données NoSQL orientées colonnes (i.e. HBase). Pour ce faire, nous avons commencé tout d’abord par une étude de faisabilité de la migration d’une base de données relationnelle vers une base de données orientée colonnes HBase. Ensuite, nous avons proposé une étude comparative des performances entre un système de gestion de bases de données relationnelles et le système de gestion de bases de données NoSQL HBase. Ceci pour objectif d'identifier clairement les points forts et les points faibles de chaque modèle pour les prendre en considération dans une éventuelle migration. Finalement, nous avons présenté une approche de conversion d’une base de données relationnelle vers une base de données HBase en proposant un ensemble des règles aidant à réussir l’opération de conversion de schéma d’une base de données relationnelle vers une base de données HBase.

Mots-clés: Migration, Mapping de schéma, Mapping de requêtes, XML, Base de données relationnelle, Big Data, NoSQL, HBase

Abstract

The main objective of this thesis is to contribute to the design and development of migration and mapping architectures from XML databases to relational databases and from relational databases to column-oriented NoSQL databases.

This thesis is articulated around two axes. In the first axis, we proposed an approach to store and query XML documents using a relational database. Our approach efficiently supports structural modifications of the XML tree (Insert, modify and delete).

As for the second axis, we presented a method of migration from a relational database to a column-oriented NoSQL database (HBase). To do so, we first of all start with a feasibility study of the relational databases migration to NoSQL databases, in particular the column-oriented database HBase. Next, we propose a performance comparative study of a relational database management system and the NoSQL HBase database management system, to clearly identify the strengths and weaknesses of each model to consider them in a possible migration. Finally, we present an approach to converting a relational database to an HBase database by proposing a set of rules that help to successfully perform the schema conversion operation from a relational database to an HBase database.

Keywords: Migration, Schema Mapping, Query Mapping, XML, Relational Database, Big Data, NoSQL, HBase

Dédicace

À la mémoire de ma mère

À mon très cher père

À ma très chère épouse et mes aimables enfants

À mes frères et sœurs

À tous les membres de la famille

À mes chers amis

À tous ceux qui ont participé de près ou de loin à la réussite de ce mémoire

Je dédie du fond de mon cœur ce modeste travail.

Zakaria

Remerciements

« La gratitude est le secret de la vie. L'essentiel est de remercier pour tout. Celui qui a appris cela sait ce que vivre signifie. Il a pénétré le profond mystère de la vie» Albert Schweitzer.

A la fin de cette aventure doctorale aussi scientifique qu'humaine, je me suis rendu compte qu'un mémoire est loin d'être un travail individuel. A cette occasion, je tiens à témoigner ma profonde reconnaissance à tous ceux qui, de près ou de loin, ont participé au bon avancement de ce travail de thèse.

En premier lieu, j'exprime mes sincères remerciements à mon directeur de thèse, le professeur Ilias CHERTI pour avoir évalué et assuré la responsabilité de mes travaux de thèse, ses encouragements, sa disponibilité, son enthousiasme. Et plus particulièrement pour ses qualités humaines d'écoute et de compréhension qui font un très bon exemple à suivre. Ce travail n'aurait pas avancé sans son aide et son soutien.

En deuxième lieu, j'adresse ma profonde gratitude à tous les membres de mon jury d'avoir accepté d'évaluer mon travail doctoral:

- Monsieur ERRKIK AHMED, professeur à la Faculté des Sciences et Techniques de Settat, pour l'intérêt qu'il a accordé à cette thèse en acceptant de faire partie du jury en tant que président. Je lui exprime toute mes reconnaissances.
- Messieurs BAHAJ Mohamed (professeur à la Faculté des Sciences et Techniques de Settat), EL AFIA Abdellatif (professeur à École Nationale Supérieure d'Informatique et d'Analyse des Systèmes), et ABOUNAIMA Mohammed Chaouki (professeur à Faculté des Sciences et Techniques de Fès) pour l'honneur qu'ils m'ont fait pour leurs participation à mon jury de thèse en tant que rapporteurs de mon travail, pour le temps consacré à la lecture de cette thèse, et pour les suggestions et les remarques pertinentes qu'ils m'ont prodiguées..
- Monsieur ELOMARY Mohamed abdou, professeur à la Faculté des Sciences et Techniques de Settat, de faire partie de ce jury en tant qu'examinateur.

Mes remerciements ne peuvent s'achever sans une pensée à mon épouse Inssaf EL GUABASSI. Je tiens aussi à lui adresser mes sincères remerciements pour son aide et soutien permanent aussi bien pour le plan personnel que professionnel.

J'adresse aussi mes remerciements à mes amis qui m'ont supportés et soutenus chacun selon sa spécialité et son savoir-faire pendant cette période de temps, en particulier Abdellatif HAJ Abdellatif, Moaad ABOUMALIK, Abdellah AMARIR, Mohamed LAMRABET, Tarik ACHAK, Issam ER-RRAHMANI, Mohammed AIT HMAD OUBRAHIM et Lahcen AKDIM.

Liste des Abréviations

Abréviation	Désignation
XML	Extensible Markup Language
W3C	World Wide Web Consortium
HTML	HyperText Markup Language
XSLT	eXtensible Stylesheet Language Transformations
DTD	Document Type Definition
RelaxNG	Regular Language for XML Next Generation
XDR	XML-Data Reduced
SOX	Simple Outline XML
DSD	Document Structure Description
TREX	Tree Regular Expression for XML
RELAX	Regular Language description for XML
SGBDR	Systèmes De Gestion De Bases De Données Relationnels
RDB	Relational Database
SQL	Structured Query Language
SEQUEL	Structured English Query Language
XPath	XML Path Language
DOM	Document Object Model
SAX	Simple API for XML
StAX	Streaming API for XML
VTD-XML	Virtual Token Descriptor for eXtensible Markup Language
LCA	Lowest Common Ancestor
DPLS	Dynamic Prefix-based Labeling Scheme
XDAS	XML Documents Addressing and Subnetting
DFPD	Dynamic Float-point Dewey
IBSL	Improved Binary String Labeling
NSM	N-ary Storage Model
LDD	Langage de définition de données
LMD	Langage de Manipulation de Données
HDFS	Hadoop Distributed File System
NoSQL	Not Only SQL
CAP	Consistency, Availability and Partition Tolerance
ACID	Atomicity, Consistency, Isolation and Durability
YCSB	Yahoo! Cloud Serving Benchmark
WAL	Write-Ahead Log
DDI	Denormalization, Duplication, and Intelligent Keys
LDD	Langage de définition des données (LDD)
LCT	Langage de contrôle des transactions
LCD	Langage de contrôle de données
LMD	Langage de manipulation de données

Liste des Figures

Figure 1: Le document XML profile.xml 13
Figure 2: La DTD du fichier XML de la Figure 1 17
Figure 3: Extrait de schéma XML Schema du fichier XML de la Figure 1 20
Figure 4: Extrait de schéma RELAX NG (XML) qui modélise les profils personnels 22
Figure 5: Extrait de schéma RELAX NG (Compact) qui modélise les profils personnels 23
Figure 6: L'architecture globale de HDFS 31
Figure 7: L'architecture détaillée de HDFS [61] 32
Figure 8: Processus de traitement des données dans MapReduce [65] 33
Figure 9: Classification des systèmes de gestion bases de données selon le théorème CAP[77] 37
Figure 10: Document XML 48
Figure 11: Arbre de données XML de notre document XML dans la Figure 10 49
Figure 12: Le schéma relationnel généré à l'aide de l'approche XMap en se basant sur le document XML de la Figure 10 50
Figure 13: Diagramme de processus de l'approche XMap 52
Figure 14: La fonction XMLParser 54
Figure 15: Organigramme de la fonction XMLParser 54
Figure 16: Fonction getOrdPathId 55
Figure 17 : Étiquetage d'un nœud précédé par des nœud frères 56
Figure 18 : Étiquetage d'un nœud non précédé par des nœud frères 57
Figure 19: Organigramme de la fonction getOrdPathId 57
Figure 20: Fonction treeBrowser 58
Figure 21: Organigramme de la fonction treeBrowser 61
Figure 22: Algorithme RDBtoXML 62
Figure 23: Logigramme de l'algorithme RDBtoXML 64
Figure 24: Fonction *getPath()* 65
Figure 25: Fonction *getVertexId()* 66
Figure 26: Fonction *getParent()* 66
Figure 27 : Requête SQL pour sélectionner tous les nœuds du document XML 67
Figure 28: Requête SQL correspondante à la requête XPath R2 68
Figure 29: Requête SQL correspondante à la requête XPath R3 68
Figure 30: Composants des systèmes d'apprentissages adaptatifs [102] 72
Figure 31: Diagramme de processus de l'approche E-XMap 77
Figure 32: Structure du Cours 78
Figure 33: Document XML du cours de Python 79
Figure 34: Arbre de données XML du cours de Python 79
Figure 35: Arbre de données XML du cours de Python étiqueté avec *ORDPATH* 81
Figure 36: Algorithme de la technique *ORDPATH* 81
Figure 37: Arbre de données XML du cours de Python étiqueté avec *Dynamic XDAS* 82
Figure 38: Algorithme de la technique *Dynamic XDAS* 83
Figure 39: Arbre de données XML du cours de Python étiqueté avec *GroupBased* 84

Figure 40: Algorithme de la technique *GroupBased*..... 85
Figure 41: Arbre de données XML du cours de Python étiqueté avec *DPLS*..... 86
Figure 42: Algorithme de la technique *DPLS*..... 87
Figure 43: Différence entre l'organisation d'une table dans une BDD relationnelle et l'organisation d'une table dans une BDD orientée colonnes..... 93
Figure 44: Architecture de HBase..... 95
Figure 45: Modèle de données de HBase..... 96
Figure 46: Algorithme de la commande Scan distribuées en MapReduce..... 102
Figure 47: Algorithme de la fonction COUNT en MapReduce..... 105
Figure 48: Algorithme de la fonction SUM en MapReduce..... 106
Figure 49: Algorithme de la fonction AVG en MapReduce..... 108
Figure 50: Algorithme de Repartition Join [172]..... 109
Figure 51: Processus d'évaluation comparative des mégadonnées [177]..... 113
Figure 52: Architecture de YCSB[181]..... 114
Figure 53: Temps de chargement de données..... 118
Figure 54: Test 1, Latence Moyenne d'insertion..... 119
Figure 55: Test 2, Temps d'exécution de charge de travail A..... 119
Figure 56: Test 2, Latence de lecture de charge de travail A..... 120
Figure 57: Test 2, Latence de mise à jour de charge de travail A..... 120
Figure 58: Test 2, Temps d'exécution de charge de travail C..... 121
Figure 59: Test 2, Latence de lecture de charge de travail C..... 121
Figure 60: Test 2, Temps d'exécution de charge de travail G..... 122
Figure 61: Test 2, Latence de mise à jour de charge de travail G..... 122
Figure 62: Test 3, Temps d'exécution de charge de travail A..... 122
Figure 63: Test 3, Latence de lecture de charge de travail A..... 123
Figure 64: Test 3, Latence de mise à jour de charge de travail A..... 123
Figure 65: Test 3, Temps d'exécution de charge de travail C..... 124
Figure 66: Test 3, Latence de lecture de charge de travail C..... 124
Figure 67: Test 3, Temps d'exécution de charge de travail G..... 125
Figure 68: Test 3, Latence de mise à jour de charge de travail G..... 125
Figure 69: Croissance exponentielle des données [192]..... 131
Figure 70: Processus de migration d'une base de données relationnelle vers une base de données orientée colonnes HBase..... 135
Figure 71: Méthode simple de transformation d'une table relationnelle à une table HBase . 136
Figure 72: Méthode améliorée de transformation d'une table relationnelle à une table HBase 137
Figure 73: Exemple d'une relation « 1-1 » dans le model relationnel..... 139
Figure 74: Transformation d'une relation de type « 1-1 »..... 139
Figure 75: Exemple de d'une relation de type « 1-n » dans le model relationnel..... 141
Figure 76: Deux tables relationnelles avec une relation « 1-n »..... 142
Figure 77: Le résultat de la conversion des tables relationnelles de la Figure 76 vers HBase en utilisant la méthode verticale..... 143
Figure 78: Le résultat de la conversion des tables relationnelles de la Figure 76 vers HBase en utilisant la méthode horizontale..... 144

Figure 79: Le résultat de la conversion des tables relationnelles de la Figure 76 vers HBase en utilisant la méthode horizontale par colonnes 145
Figure 80: Exemple d'une relation « n-m » dans le model relationnel 147
Figure 81: Exemple de tables relationnelles avec une relation « n-m » 148
Figure 82: Le résultat de la conversion des tables relationnelles de la Figure 81 vers HBase en utilisant la méthode hybride 149

Liste des Tableaux

Tableau 1: Comparaison entre XML et HTML 14
Tableau 2 : Requêtes XPath 67
Tableau 3: Récapitulatif des techniques d'étiquetage 87
Tableau 4: Relations structurelles supportées par chaque technique 88
Tableau 5: Equivalence des concepts du modèle de HBase dans le modèle relationnel..... 97
Tableau 6: Caractéristiques de l'environnement de l'expérimentation..... 116
Tableau 7: Charges de travail utilisés 117
Tableau 8: Structure du dataset de YCSB 117

Table des Matières

Résumé ii
Abstract iii
Dédicace iv
Remerciements v
Liste des Abréviations vi
Liste des Figures vii
Liste des Tableaux x
Table des Matières xi
Introduction Générale 16
1. Contexte Général 2
2. Motivation et Objectifs 4
3. Contributions de la Thèse 5
3.1. XMap : une Nouvelle Approche pour Stocker, Interroger et Extraire les Documents XML à partir de Bases de Données Relationnelles 5
3.2. L'Application de XMap dans le Domaine de la Conception des Systèmes d'Apprentissage Adaptatifs 5
3.3. Migration des Bases de Données Relationnelles vers le NoSQL : Une Etude de Faisabilité 6
3.4. Etude Comparative des Performances du SGBD Relationnel MySQL et le SGBD NoSQL HBase 6
3.5. Une approche de conversion d'une base de données relationnelle vers une base de données NoSQL HBase 7
4. Organisation de la Thèse 7
5. Productions Scientifiques 8
5.1. Articles Publiés 8
5.2. Conférences Internationales Indexées 9
5.3. Conférences Internationales avec Comités de Lecture 9
Partie I État de l'Art 10
Chapitre I 11
Préliminaires 11
1. Bases de données XML 12
1.1. Langage de balisage extensible (XML) 12
1.2. Un document XML bien formé est valide 15

2. Bases de données relationnelles 23
2.1. Modèle relationnel 23
2.2. Systèmes de gestion de bases de données relationnelles 24
2.3. Concepts de base des systèmes de gestion des bases de données relationnelles 25
2.4. Langage SQL 28
3. Big Data 29
3.1. Hadoop 29
3.2. Bases de données NoSQL 33
Partie II Contributions 38
Chapitre II 39
XMap : une Nouvelle Approche pour Stocker, Interroger et Extraire les Documents XML à partir de Bases de Données Relationnelles 39
1. Introduction 40
2. Travaux connexes 41
4.1. XRel 41
4.2. XParent 42
4.3. Edge 43
4.4. Xlight 43
4.5. Mini-XML 44
4.6. L'approche de Ying 45
4.7. XRecursive 45
4.8. XAncestor 46
3. Technique d'étiquetage ORDPATH 46
4. Modèle de données XML 48
5. Présentation du schéma relationnel proposé 49
6. Implémentation 51
8.1. Modèle de processus de l'approche XMap 52
8.2. Algorithme XMLtoRDB 53
8.3. Algorithme RDBtoXML 60
8.4. Mapping des requêtes XPath en requêtes SQL 65
7. Conclusion 68
Chapitre III 70
L'Application de XMap dans le Domaine de la Conception des Systèmes d'Apprentissage Adaptatifs 70

1. Introduction 71
2. Concepts de base 72
2.1. Systèmes d'Apprentissage Adaptatifs 72
2.2. Les techniques d'étiquetage 73
3. Notre approche 75
3.1. Principe de l'approche 75
3.2. Support de cours 77
3.3. Etiquetage des nœuds d'un arbre de données XML d'un support de cours 78
4. Conclusion 89
Chapitre IV 90
Migration des Bases de Données Relationnelles vers le NoSQL : Une Etude de Faisabilité 90
1. Introduction 91
2. Concepts de base 91
2.1. Les bases de données orientées colonnes 91
2.2. HBase 93
2.3. Justification de choix de HBase 96
3. Modélisation d'une base de données NoSQL HBase en utilisant le modèle relationnel 97
4. L'application des opérations de l'algèbre relationnelle sur HBase 98
4.1. Union 98
4.2. Différence 98
4.3. Intersection 99
4.4. Produit cartésien 99
4.5. Sélection 99
4.6. Projection 99
4.7. Renommage 100
4.8. θ-jointure 100
4.9. Jointure naturelle 100
4.10. Division 101
5. L'implémentation des opérations de l'algèbre relationnelle dans HBase 101
5.1. Scan 101
5.2. Get 103
5.3. Group by 104
5.4. Fonctions d'agrégation 104

5.5. Jointure 108
6. Conclusion 109
Chapitre V 111
Etude Comparative des Performances du SGBD Relationnel MySQL et le SGBD NoSQL HBase 111
1. Introduction 112
2. Préliminaires 113
2.1. Benchmarks pour les systèmes de gestion de bases de données 113
2.2. YCSB 113
3. Méthodologie de l'expérimentation 115
4. Environnement de l'expérimentation 116
5. Résultats expérimentaux 117
5.1. Chargement des données 118
5.2. Evaluation de la variation des performances en fonction du nombre d'enregistrement de la base de données 119
5.3. Evaluation de la variation des performances en fonction du nombre d'opérations par second 122
6. Synthèse des résultats 125
7. Perspectives 128
8. Conclusion 129
Chapitre VI 130
Une Approche de Conversion d'une Base de Données Relationnelle vers une Base de Données NoSQL HBase 130
1. Introduction 131
2. Outils de migration 132
3. Travaux connexes 132
4. Conversion d'une base de données relationnelle vers une base de données orientée colonnes HBase 133
4.1. Définitions formelles 133
4.2. Processus de migration d'une base de données relationnelle vers une base de données orientée colonnes HBase 135
4.3. Règles de transformation de schéma 135
5. Conclusions 149
Conclusion Générale et Perspectives 151
Bibliographie 155

Annexes 171
Annexe 1 172
Commandes YCSB 172
1. Test1 173

Introduction Générale

1. Contexte Général

Il est incontestable que les données sont devenues une partie importante de la vie au XXIe siècle. Avec l'accélération de la croissance des données générées, il faut trouver un moyen efficace pour gérer ces données, de les traiter et d'en extraire des connaissances. La solution alors, sont les systèmes de gestion de bases de données(SGBD). Ces derniers sont apparus dans les années soixante, et au fil du temps, ils sont développés. On distingue cinq générations des SGBD [1]:

- Bases de données réseau et hiérarchiques : ce sont les premiers systèmes introduits pour stockées les données. Le modèle hiérarchique organisait les données dans une structure arborescente. Il est développé par IBM dans les années 1960. L'inconvénient principal de ce modèle c'est que les relations complexes entre les nœuds ne sont pas supportées. Pour surmonter les lacunes du modèle hiérarchique, les chercheurs du consortium CODASYL ont proposé le modèle réseau, qui a été une innovation dans le marché des systèmes de gestion des bases de données.
- Bases de données relationnelles : Dans les années 1970, le monde s'est doté de son tout premier système de gestion de base de données relationnelle ou SGBDR qui s'est basé sur le modèle relationnel proposé par Edgar Codd[2]. Cette version avancée et plus efficace des systèmes de gestion de bases de données traditionnels peut stocker des données en lignes et en colonnes et y accéder en utilisant un langage d'interrogation de base de données nommé SQL.
- Bases de données orientées objet : Au milieu des années 1980, commence l'effervescence du paradigme de la programmation orientée objet(POO), et les années 1990 commence l'âge d'or des langages de la POO. À cette époque, les chercheurs ont proposé un nouveau modèle de base de données nommé : les bases de données orientées objet. L'objectif de ce type de bases de données est de surmonter les limites du modèle relationnel qui ne prend pas en charge certaines applications avancées surtout au niveau de la capacité de modélisation. Dans les bases de données orientées objet, les données sont stockées sous forme d'objets. Ces systèmes de gestion de bases de données ne sont pas très populaires en raison de l'absence de normes et l'énormes investissements consentis dans les systèmes relationnels[1][3].

- Le bases de données non-relationnelles : Dans les années 2000, une autre génération des bases de données a vu le jour, il s'agit des bases de données NoSQL. Le progrès technologique et l'avènement du web 2.0 et les réseaux sociaux engendre une explosion des données avec différents types (structurées, semi structurées et non structurées) et des exigences qui ne sont pas supportées par les systèmes de base de données traditionnelles. Les bases de données NoSQL se caractérisent par une conception simple[4], [5],
- Le bases de données NewSQL : les SGBD NewSQL sont des systèmes de bases de données relationnelles avec des architectures distribuées et tolérantes aux pannes. La capacité de gestion des données en mémoire est une caractéristique supplémentaire et typique des bases de données NewSQL[6]. L'avantage de ce type de bases de données est qu'il peut combiner les propriétés de ACID des SGBDR et l'évolutivité horizontale de NoSQL[7].

En 1998 le consortium W3C a publié la première version du langage XML qu'a été un outil pour stocker, organiser, et transmettre les données sur internet. Sa robustesse réside dans sa capacité de partage des données entre les différents systèmes en raison de sa nature indépendante de la plate-forme. Les données XML ne nécessitent aucune conversion lorsqu'elles sont transférées entre différents systèmes. Après des années, un nombre important d'entreprises utilisent le XML comme format d'échange de données. Il a été la norme de facto pour l'échange des informations entre les applications d'entreprise dans des architectures orientées services.

En raison de la forte utilisation du XML et l'émergence rapide de la technologie XML au cours des deux dernières décennies, un grand volume des documents XML est accumulé [8], [9]. Ce qui nécessite une gestion efficace, transparente et précise tout en conservant la sémantique du document original. À ce stade, les systèmes de gestion de bases de données (SGBD) entrent en jeu. Il existe quatre approches pour le stockage des données XML[9]. La première approche est les bases de données XML native qui prennent en charge le modèle de données XML et les langages de requêtes XML. La deuxième consiste à stocker les données dans des bases de données relationnelles, la troisième approche reposer sur l'utilisation des fichiers et la dernière elle permet de stocker les données XML dans des bases de données orienté-objets [10], [11].

De nos jours, avec l'apparition de l'Internet et l'avènement de Web 2.0, des sites des réseaux sociaux et les objets connectés, les données montrent une tendance à la croissance exponentielle. Ces données sont rapides, volumineuses et variées. Selon les statistiques de la société américaine Domo [12], pendant chaque minute en 2020, 208333 personnes participent à des réunions sur Zoom, 52083 utilisateurs se connectent à Microsoft Teams, les utilisateurs de LinkedIn postulent pour 69444 offres d'emploi, 150 milles messages Facebook sont partagés, 500 heures de vidéos YouTube sont uploadées et 41,66 millions de messages sont partagés sur WhatsApp. Le volume, la variété, et la vélocité de ces données nécessitent une évolutivité régulière avec une haute disponibilité[13]. Ce phénomène a remis en question la suprématie des bases de données relationnelles en tant que choix ultime de système de gestion de bases de données.

2. Motivation et Objectifs

Le XML est devenu la norme dominante pour le stockage et le transfert de données via le Web, c'est grâce à sa simplicité, sa flexibilité et sa possibilité d'extension. Le XML peut être adapté à de multiples domaines en raison de sa nature indépendante de plate-forme. Cela entraine une énorme quantité d'informations au format XML qui doit être gérées[14]. De là vient la nécessité de coopérer avec un système de gestion de bases de données pour combiner les avantages des deux mondes. Selon [9], [15] il existe quatre approches pour le stockage des données XML à savoir les bases de données XML native, les bases de données relationnelles, les fichiers XML et les bases de données orientées-objets [10], [11]. La première approche souffre de plusieurs limitations (sécurité, transactions et gestion d'accès concurrents, renvoie les données uniquement sous format XML) [16], [17]. Pour surmonter ces limitations, plusieurs approches ont été proposées [18]–[26], pour tirer parti des technologies matures fournies par les systèmes de gestion de bases de données relationnels et pour exploiter toute la puissance de cette technologie.

Dans la même veine, un grand volume de données structurées et non-structurées est généré par les réseaux sociaux, les objets connectés, les fichiers journaux des activités des utilisateurs sur le web ou les journaux de transactions monétaires. Or, les systèmes de gestion de base de données relationnelle n'ont pas été conçues pour traiter efficacement les données très volumineuses et les données non- structurées. C'est la raison pour laquelle les leaders de l'Internet tels que Google, Amazon, eBay, Alibaba et Facebook ont développé un nouveau modèle appelé bases de données NoSQL[27], afin de surmonter la faiblesse des systèmes de

gestion de bases de données relationnelles face à la variété, la vélocité et le grand volume de nouvelles données capturées. L'utilisation ou la migration vers ces nouvelles technologies de bases de données, donc, est un véritable défi pour les concepteurs de logiciels.

C'est dans le contexte de la migration et la conversion des bases de données que se placent les travaux de cette thèse. L'objectif principal de notre contribution est de proposer :

1. Une approche pour stocker et interroger les documents XML à l'aide des systèmes de gestion de bases de données relationnelles pour tirer parti de la maturité de ces systèmes.
2. Une approche pour la conversion de schéma d'une base de données relationnelle vers une base de données NoSQL orientée colonnes (HBase).

3. Contributions de la Thèse

L'objectif principal de cette thèse est de contribuer à la conception et le développement des architectures de migration des bases de données XML vers les bases de données relationnelles et des bases de données relationnelles vers le NoSQL. Les contributions proposées dans le cadre de ce travail sont comme suite :

3.1. XMap : une Nouvelle Approche pour Stocker, Interroger et Extraire les Documents XML à partir de Bases de Données Relationnelles

Dans ce travail, nous avons proposé une approche qui permet de stocker et d'interroger un document XML à l'aide de bases de données relationnelles. Ceci en décomposant un document XML en trois tables sans utiliser le XML Schema ou DTD. Notre approche supporte efficacement les modifications structurelles de l'arbre XML (Insertion, modification et suppression). Pour cela, nous proposons un algorithme pour mapper les données XML vers une base de données relationnelle et un autre algorithme qui permet d'assurer la reconstruction d'un document XML à partir d'une base de données relationnelle.

3.2. L'Application de XMap dans le Domaine de la Conception des Systèmes d'Apprentissage Adaptatifs

Notre deuxième contribution vise à mettre en production l'approche précédente pour montrer son utilité et son efficacité. Pour atteindre cet objectif, nous avons adopté une approche pluridisciplinaire. Ainsi, Notre choix est tombé sur le domaine de la conception des systèmes d'apprentissage adaptatifs vu que l'apprentissage est parmi les premières préoccupations de la société. Dans cette contribution nous avons discuté du potentiel

d'utilisation de notre approche E-XMap pour stocker le document XML du support de cours dans une base de données relationnelle. Ceci en prenant en compte l'adaptation du contenu pédagogique en fonction des caractéristiques spécifiques de chaque apprenant. En même temps, nous avons utilisé les techniques d'étiquetage XML afin d'identifier les relations structurelles entre les nœuds et aussi pour accélérer le traitement des requêtes.

3.3. Migration des Bases de Données Relationnelles vers le NoSQL : Une Etude de Faisabilité

Dans le cadre de cette contribution, nous proposerons une étude de faisabilité de la migration des bases de données relationnelles vers les bases de données NoSQL, en particulier la base de données orientée colonnes HBase. Cette contribution examine la faisabilité de la migration d'une base de données relationnelle vers une base de données HBase, en proposant un modèle relationnel à la base de données HBase. Cette modélisation permettra de comparer le modèle des bases de données relationnelles et le modèle de HBase, en examinant les opérations de base de l'algèbre relationnelle dans le modèle de données HBase et en citant quelques implémentations des opérations clés de l'algèbre relationnelle dans la base de données de HBase.

3.4. Etude Comparative des Performances du SGBD Relationnel MySQL et le SGBD NoSQL HBase

La comparaison de différents modèles de bases de données permet d'avoir une vision claire pour choisir le modèle le plus approprié à un contexte donné. L'objectif de cette contribution est de comparer les performances du modèle relationnel (MySQL) et le modèle NoSQL (HBase) et révéler comment le nombre d'opérations et le nombre d'enregistrements affectent la performance en termes de latence et en termes de temps d'exécution en utilisant le Framework YCSB. Afin d'élaborer une approche efficace pour la migration d'une base de données relationnelle vers une base de données HBase, nous avons commencé par une étude de faisabilité dans la contribution précédente, ensuite, dans cette contribution, nous allons établir une comparaison expérimentale des performances entre la base de données relationnelle MySQL et la base de données NoSQL HBase. Le but de cette comparaison est d'identifier clairement les points forts et les points faibles de chaque modèle pour les prendre en considération dans une éventuelle migration. De plus, il est important de comprendre quel type de base de données est le plus adapté et le plus efficace pour une telle application. Par conséquent, nous pensons qu'il est nécessaire de comparer les modèles relationnels (MySQL)

et non relationnels (HBase) en se basant sur la capacité de traitement de différents types de requêtes.

3.5. Une approche de conversion d'une base de données relationnelle vers une base de données NoSQL HBase

Dans cette contribution nous présentons le processus de migration d'une base de données relationnelle vers HBase avec l'explication de chaque étape de migration, ensuite nous proposons un ensemble de règles de transformation de schéma d'une base de données relationnelle vers HBase.

4. Organisation de la Thèse

Le présent manuscrit contient deux grandes parties à savoir « État de l'art » et « Contributions » en plus de l'introduction et de la conclusion générale.

- **Partie I : État de l'Art**

Elle a comme objectif d'introduire les principaux concepts que nous aborderons le long de notre manuscrit, elle décrit les concepts fondamentaux utilisés dans cette thèse.

- **Partie II : Contributions**

Cette partie est consacrée à la présentation des différentes contributions de cette thèse, elle comporte cinq chapitres :

1. **Chapitre 1** : Présente une approche qui permet de stocker et d'interroger un document XML à l'aide des bases de données relationnelles, en décomposant ce document en trois tables sans utiliser le XML Schema ou DTD.
2. **Chapitre 2** : Aborde notre deuxième contribution. En effet, nous présentons une approche qui permet de générer automatiquement un cours pour chaque apprenant selon ses préférences individuelles afin d'assurer une meilleure adaptation. Dans ce but, nous avons opté pour le langage XML pour représenter le support de cours. Afin d'éviter les faiblesses des bases de données XML et de bénéficier des atouts des bases de données relationnelles, le document XML du support de cours sera stocké dans des bases de données relationnelles et afin d'identifier les relations entre les nœuds et d'accélérer le traitement des requêtes, nous utilisons les techniques d'étiquetage XML.

3. **Chapitre 3** : Élabore une étude de faisabilité de la migration des bases de données relationnelles vers les bases de données NoSQL spécifiquement la base de données HBase.
4. **Chapitre 4** : Met la lumière sur une comparaison expérimentale entre une base de données relationnelle (MySQL) et une base de données NoSQL (HBase) au niveau de temps d'exécution et de latence dans différents scénarios en utilisant le Framework YCSB.
5. **Chapitre 5** : Présente une approche pour la migration d'une base de données relationnelle vers une base de données orientée colonne HBase.

En guise de conclusion, ce manuscrit se termine par une synthèse générale des différents axes abordés dans cette thèse. Ensuite, nous proposons des perspectives et pistes d'évolution que nous envisageons pour compléter et améliorer ce travail.

5. Productions Scientifiques

La liste suivante présente les publications concernant les travaux effectués dans le cadre de cette thèse.

5.1. Articles Publiés

- Bousalem, Z., & Cherti, I. (2015). XMap: A Novel Approach to Store and Retrieve XML Document in Relational Databases. J. Softw., 10(12), 1389-1401. (**DBLP, EBSCO, ProQuest, INSPEC**)
- Bousalem, Z., Cherti, I., & Zhao, G. (2017, April). Migration from Relational Databases to HBase: A Feasibility Assessment. In International Conference on Advanced Information Technology, Services and Systems (pp. 383-395). Springer, Cham. (**Scopus, Web of Science**).
- Bousalem, Z., El Guabassi, I., & Cherti, I. (2018, July). Toward adaptive and reusable learning content using XML dynamic labeling schemes and relational databases. In International Conference on Advanced Intelligent Systems for Sustainable Development (pp. 787-799). Springer, Cham. (**Scopus, Web of Science, DBLP, EI-Compendex**).
- Bousalem, Z., El Guabassi, I., & Cherti, I. (2019). Relational databases versus HBase: an experimental evaluation. Adv. Sci. Technol. Eng. Syst. J., 4(2), 395-401. +(**Scopus**).

5.2.Conférences Internationales Indexées

- BOUSALEM, Zakaria, CHERTI, Ilias, et ZHAO, Gansen (2017). Migration from Relational Databases to HBase: A Feasibility Assessment. International Conference on Advanced Information Technology, Services and Systems **(Springer) (Scopus, Thomson Reuters)**
- BOUSALEM, Zakaria, EL GUABASSI, Inssaf, et CHERTI, Ilias (2018). Toward Adaptive and Reusable Learning Content Using XML Dynamic Labeling Schemes and Relational Databases. International Conference on Advanced Intelligent Systems for Sustainable Development **(Springer) (Scopus, Thomson Reuters, DBLP, EI-Compendex)**

5.3. Conférences Internationales avec Comités de Lecture

- BOUSALEM, Zakaria et CHERTI, Ilias (2015). XMap: An approach to store XML Documents in Relational Databases. International Conference on Advanced Information Technology, Services and Systems (AIT2S'15). Settat. Maroc.

Partie I
État de l’Art

Chapitre I

Préliminaires

1. Bases de données XML

1.1. Langage de balisage extensible (XML)

L'évolution des technologies des systèmes d'information a entraîné l'informatisation de nombreuses applications dans divers domaines d'activité. Les données sont des ressources essentielles dans de nombreuses organisations ; il est donc devenu urgent d'y accéder et de les partager efficacement, d'en extraire des informations et de les utiliser. Par conséquent, de nombreux efforts visant à intégrer les diverses sources de données dispersées sur plusieurs sites et à extraire des informations de ces bases de données sous forme de modèles. Ces sources de données peuvent être des bases de données gérées par des systèmes de gestion de bases de données, ou elles peuvent être stockées dans un référentiel à partir de multiples sources de données.

L'avènement du Web au milieu des années 1990 a entraîné une demande encore plus importante de gestion efficace des données, des informations et des connaissances. Il devient presque impossible de gérer l'information avec les outils classiques grâce à la grande quantité de données disponibles sur le Web. De nouveaux outils et techniques sont nécessaires pour traiter ces données. Par conséquent, divers outils sont en cours de développement pour assurer le stockage et l'interopérabilité entre les multiples sources de données et les systèmes, et pour extraire des informations des bases de données et des entrepôts sur le Web.

En général, la gestion des données comprend la gestion des bases de données, l'interopérabilité, la migration, l'entreposage et l'extraction [28]. Par exemple, les données sur le Web doivent être gérées et exploitées pour extraire des informations et des modèles. Les données peuvent se trouver dans des fichiers, des bases de données relationnelles ou d'autres types de bases de données comme les bases de données multimédia. Les données peuvent être structurées ou non structurées.

De nos jours, le XML [29] est sans doute devenu le langage le plus important pour le balisage des documents pour le Web et pour l'industrie en général. Tout aussi important, le XML devient rapidement un standard de facto pour le balisage des données, pour l'échange d'informations entre les systèmes d’information hétérogènes et entre les machines ; Le XML a introduit toute une série de concepts qui améliorent la convivialité des systèmes informatiques.

XML [29] est un langage de balisage créé par le World Wide Web Consortium (W3C) pour définir une syntaxe permettant d'encoder des documents que les humains et les machines peuvent lire. Pour ce faire, il utilise des balises qui définissent la structure du document, ainsi que la manière dont le document doit être stocké et transféré. La Figure 1 illustre un document XML qui représente les informations de profile d'une personne.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<profile>
    <proprietaire type="etudiant" age="32">
        <Name>
            <Prenom>Zakaria</Prenom>
            <Nom>BOUSALEM</Nom>
        </Name>
        <Telephone>
            <Maison>(000)000-0000</Maison>
            <Travail>(000)000-0000</Travail>
            <Fax />
            <portable />
        </Telephone>
        <Adresse type="personnelle">
            <Rue>Hay El Farah, N°99</Rue>
            <Ville>Agadir</Ville>
            <Province>Agadir</Province>
            <codePostal>80000</codePostal>
        </Adresse>
        <Email>zakaria.bousalem@gmail.com</Email>
        <Formation>
            <Institution>
                <diplome type="Master" specialite="Info" moyenne="15.86" />
                <dateObtention>2010</dateObtention>
                <nomInstitution>Université Abdelmalek Essaadi</nomInstitution>
            </Institution>
            <Institution>
                <diplome type="Licence" specialite="Info" moyenne="14.77" />
                <dateObtention>2008</dateObtention>
                <nomInstitution>Université Cadi Ayyad</nomInstitution>
            </Institution>
        </Formation>
        <Competences>
            <LangagesProg>Java</LangagesProg>
            <LangagesProg>JavaScript</LangagesProg>
            <LangagesProg>XML</LangagesProg>
            <Systeme>Linux</Systeme>
        </Competences>
    </proprietaire>
</profile>
```

Figure 1: Le document XML profile.xml

La chose qui différencie XML, c'est qu'il est extensible. Le XML n'a pas de langage de balisage prédéfini, comme le HTML. Il permet aux utilisateurs de créer leurs propres balises pour décrire le contenu, ce qui donne un ensemble de tags illimité et auto-défini. Le HTML est essentiellement un langage qui se concentre sur la présentation du contenu, tandis que le

XML est un langage dédié à la description des données utilisé pour le stockage des données. Tableau 1 représente un tableau comparatif de XML et HTML.

Un avantage évident du XML est qu'il offre un moyen de représentation des données structurées sans aucune information supplémentaire. Alors il devient très facile d'envoyer des informations structurées entre les systèmes tant que la structure est "inhérente" au document XML contrairement aux fichiers plats qui dévoient être attachés à un document supplémentaire qui décrit comment la structure apparaît. Les documents XML sont des fichiers texte, donc ils peuvent être facilement produits et utilisés par les systèmes existants, ce qui permet à ces systèmes d'exposer leurs données existantes de telle manière que différents utilisateurs puissent y accéder facilement.

Un autre avantage de l'utilisation du XML est la possibilité de tirer parti d'outils déjà disponibles ou commençant à apparaître, qui utilisent le XML pour susciter des comportements plus sophistiqués. Par exemple, XSLT peut être utilisé pour styliser des documents XML, produire des documents HTML ou tout autre type de document texte.

Tableau 1: Comparaison entre XML et HTML

Critère de comparaison	XML	HTML
Type de langage	XML fournit un Framework pour définir les langages de balisage.	HTML est un langage de balisage prédéfini.
Erreurs	Aucune erreur permise.	Erreurs minimes possibles.
Utilisation	XML est un outil indépendant du logiciel et du matériel, il est utilisé principalement pour l'échange de données entre deux partenaires. Il se concentre sur les données elles même.	HTML est utilisé pour afficher les données et se concentre sur l'apparence des données.
Espaces blancs	Les espaces blancs peuvent être préservés en XML.	HTML ne préserve pas les espaces blancs.
Balises fermantes	Fermeture des balises obligatoire.	Fermeture non indispensable.

Sensibilité à la casse	Sensible à la casse	Insensibles à la casse
Imbrication	L'imbrication des balises doit, être correcte	N’est pas important

1.2. Un document XML bien formé est valide

1.2.1. Document XML bien formé

Un document XML est dit "bien formé" s'il satisfait à certaines règles, spécifiées par le W3C.

Ci-dessous une liste non exhaustive de ces règles :

- Un document XML doit avoir un élément racine.
- Un document XML ne peut contenir qu'un seul élément racine.
- L'élément racine d'un document XML est un élément qui n'est présent qu'une seule fois dans un document XML et il n'apparaît pas comme un élément enfant au sein d'un autre élément.
- Un document XML bien formé doit avoir une balise de fermante correspondante à toute balise ouvrante.
- L'imbrication des éléments dans un document XML doit être correcte.
- Dans chaque élément, deux attributs ne doivent pas avoir le même nom.

1.2.2. Document XML valide

Un document XML est valide s’il est associé à un fichier de règles DTD [30], XML Schema [31], RelaxNG [32], Schematron [33], XDR [34] ou SOX [35] et si le document est conforme à ce fichier de règles. Nous allons voir les langages de définition de schémas les plus utilisés DTD, XML Schema et RelaxNG [36].

1.2.2.1. Document Type Definitions (DTD)

Désigne la "Définition de type de document". Une DTD définit les balises et les attributs utilisés dans un document XML. Les DTD vérifient la validité de la structure et du vocabulaire d'un document XML par rapport aux règles grammaticales du langage XML appropriées.

Les DTD font partie de la norme XML du W3C, mais sont généralement considérés comme une technologie de schéma distincte et ne sont pas généralement utilisés en conjonction avec d'autres formats de schéma comme XSD, RelaxNG, etc.

Les DTD font deux types de déclarations :

- Déclaration interne : Fait partie du document lui-même, elle est insérée dans la définition de DOCTYPE près du début du document XML.
- Déclaration externe : Indique les déclarations DTD contenues dans un fichier externe.

Les DTD décrivent la structure d'un document XML par le biais de déclarations d'éléments et de listes d'attributs. Les déclarations d'éléments désignent l'ensemble des éléments autorisés dans le document, et précisent comment les éléments déclarés et les séries de données qui peuvent être contenus dans chaque élément. Les déclarations de listes d'attributs désignent l'ensemble des attributs autorisés pour chaque élément déclaré, y compris le type de chaque valeur d'attribut, ou un ensemble explicite de valeur(s) valide(s). Les déclarations de balisage DTD déclarent quels types d'éléments, listes d'attributs, entités et notations qui sont autorisés dans la structure d’un document XML. La Figure 2 représente la DTD du fichier profile.xml (cf. Figure 1)

```
<?xml encoding="UTF-8"?>
        <!ELEMENT profile (proprietaire)>
        <!ELEMENT proprietaire (Name,Telephone,Adresse,Email,Formation,
                Competences)>
        <!ATTLIST proprietaire
                age CDATA #REQUIRED
                type NMTOKEN #REQUIRED>
        <!ELEMENT Name (Prenom,Nom)>
        <!ELEMENT Telephone (Maison,Travail,Fax,portable)>
        <!ELEMENT Adresse (Rue,Ville,Province,codePostal)>
        <!ATTLIST Adresse
                type NMTOKEN #REQUIRED>
        <!ELEMENT Email (#PCDATA)>
        <!ELEMENT Formation (Institution)+>''>
        <!ELEMENT Competences (LangagesProg|Systeme)+>
        <!ELEMENT Prenom (#PCDATA)>
        <!ELEMENT Nom (#PCDATA)>
        <!ELEMENT Maison (#PCDATA)>
        <!ELEMENT Travail (#PCDATA)>
        <!ELEMENT Fax EMPTY>
        <!ELEMENT portable EMPTY>
        <!ELEMENT Rue (#PCDATA)>
        <!ELEMENT Ville (#PCDATA)>
        <!ELEMENT Province (#PCDATA)>
        <!ELEMENT codePostal (#PCDATA)>
        <!ELEMENT Institution (diplome,dateObtention,nomInstitution)>
        <!ELEMENT LangagesProg (#PCDATA)>
        <!ELEMENT Systeme (#PCDATA)>
        <!ELEMENT diplome EMPTY>
        <!ATTLIST diplome
                type (Licence|Master|Doctort) "Licence" #REQUIRED
                moyenne CDATA #REQUIRED
                specialite NMTOKEN #REQUIRED>
        <!ELEMENT dateObtention (#PCDATA)>
        <!ELEMENT nomInstitution (#PCDATA)>
```

Figure 2: La DTD du fichier XML de la Figure 1

Les DTD ont certaines limites qui sont liées à leur flexibilité comme :

- Les différences entre la syntaxe de la DTD et la syntaxe XML.
- Elles ne supportent pas l'utilisation des espaces des nomes.
- Manque de typage des données.
- Descriptions limitées des modèles de contenu.

1.2.2.2. XML Schema

Un XML Schema fournit un mécanisme permettant de spécifier la sémantique structurelle des documents XML. Il vise à définir une classe de documents XML de sorte que ; chaque document XML instancié puisse être décrit pour se conformer à un schéma particulier et bien défini auparavant. Le langage XML Schema représente la grammaire d'un document, il est basé sur la syntaxe XML 1.0, et utilise des espaces de noms. Les

caractéristiques du langage XML Schema offrent des fonctionnalités qui vont au-delà de celles des DTD.

Les mécanismes de base de XML Schema comprennent la déclaration d'éléments et d'attributs, ainsi que la définition de types simples et complexes. Les éléments sont dits complexes s'ils contiennent des sous-éléments ou attributs. Les éléments sont dits simple lorsqu'ils contiennent des valeurs de type nombres, chaînes de caractères ou des dates mais ils ne contiennent aucun sous-élément. Les attributs sont toujours des types simples. Les types simples peuvent être soit des chiffres, des chaînes de caractères ou des dates, soit des dérivés de types prédéfinis. En utilisant le mot clé "restriction", de nouveaux types simples peuvent être définis à la base de types simples existants (prédéfinis et dérivés).

Les types simples peuvent également être des types liste ou des types union. Les types de liste sont constitués de valeur de type atomiques séparés par des espaces. Les types d'union permettent de faire l'union entre des types simples. En utilisant l'élément complexType, de nouveaux types complexes peuvent être définis. Un élément de type complexe est un élément XML qui contient d'autres éléments et/ou attributs.

Un élément ou attribut global est déclaré par xsd:element comme étant un enfant direct de l'élément xsd:schema. Ils peuvent être référencés ailleurs dans le schéma pour créer de nouveaux éléments ou attributs, ce qui est permet d’éviter les répétitions et donne une certaine modularité au code.

Voici quelques notions particulières au XML Schéma :

- Les espaces de noms XML (*namespaces*) sont utilisés pour fournir des éléments et des attributs à nom unique dans un document XML. Une instance XML peut contenir des noms d'éléments ou d'attributs provenant de plus d'un vocabulaire XML. Si chaque nom d'élément ou d'attribut est doté d'un espace de noms, l'ambiguïté entre les éléments ou les attributs avec le même nom peut être résolue.
- L'une des innovations les plus puissantes de XML Schema est la possibilité de la définition de nouveaux types de données par l'utilisateur par la dérivation de types fournis par la recommandation XML Schema ou par des types déjà créés par un auteur. Grâce à cette nouvelle fonctionnalité, les auteurs de documents sont en mesure de dériver de nouveaux types sur la base de types déjà créés. La

dérivation de nouveaux types, qu'ils soient simples ou complexes, constitue une amélioration par rapport à l'utilisation des modèles de contenu statiques qui peut être créer à l'aide de DTD. La possibilité de dériver de nouveaux types de données dans le schéma XML permet de réutiliser les types existants d'une manière similaire à celle de l'utilisation de l'héritage dans la programmation orientée objet. En utilisant la dérivation des types il n'est pas nécessaire de coder ou de créer de nouveaux types de données à partir de zéro. Cette fonctionnalité présente un grand avantage en permettant la réutilisation du code, plutôt que la création répétitive de nouveau code qui doit ensuite passer par un processus de débogage complet. La dérivation permet de la définition de nouveaux types en utilisant la restriction ou l'extension de la définition des types.

- La définition des nouveaux types par dérivation par extension permet simplement d'étendre la gamme de valeurs fournies par le type de base. L'extension n'est pas autorisée que pour les types complexes.
- La dérivation de nouveaux types par restriction est un outil puissant dans l'utilisation de XML Schema. Elle permet de définir précisément les valeurs autorisées pour le contenu d'un élément ou la valeur d'un attribut.

La Figure 3 présente le schéma XML de document XML de la Figure 1.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:element name="profile" type="proprietaire"/>
    <xs:complexType name="proprietaire">
        <xs:sequence>
            <xs:element ref="proprietaire"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="proprietaire">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="NomComplet"/>
                <xs:element ref="Telephone"/>
                <xs:element ref="Adresse"/>
                <xs:element ref="Email"/>
                <xs:element ref="Formation"/>
                <xs:element ref="Competences"/>
            </xs:sequence>
            <xs:attributeGroup ref="attlist.proprietaire"/>
        </xs:complexType>
    </xs:element>
    <xs:attributeGroup name="attlist.proprietaire">
        <xs:attribute name="age" use="required"/>
        <xs:attribute name="type" use="required" type="xs:NMTOKEN"/>
    </xs:attributeGroup>
    <xs:element name="NomComplet">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Prenom"/>
                <xs:element ref="Nom"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Telephone">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Maison"/>
                <xs:element ref="Travail"/>
                <xs:element ref="Fax"/>
                <xs:element ref="portable"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Adresse">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Rue"/>
                <xs:element ref="Ville"/>
                <xs:element ref="Province"/>
                <xs:element ref="codePostal"/>
            </xs:sequence>
            <xs:attributeGroup ref="attlist.Adresse"/>
        </xs:complexType>
    </xs:element>
    <xs:attributeGroup name="attlist.Adresse">
```

Figure 3: Extrait de schéma XML Schema du fichier XML de la Figure 1

1.2.2.3. RelaxNG

RELAX NG (Regular Language for XML Next Generation) est un langage de schéma pour XML. RELAX NG est utilisé pour valider les documents d'instance XML. Il a été écrit par Makoto Murata et James Clark avec une conception basée sur TREX (Tree Regular Expression for XML) de Clark et RELAX (Regular Language description for XML) de Murata.

Lorsque la recommandation XML 1.0 a été créée, elle comprenait la Définition de type de document DTD, le format de validation du prédécesseur du XML, le SGML. Les DTD étaient utiles dans la mesure où elles permettaient aux praticiens de créer une hiérarchie qu'un document XML devait suivre. Malheureusement, les DTD ont deux grands inconvénients. Premièrement, ce n'étaient pas des documents XML, deuxièmement, elles avaient une capacité limitée à créer des types de données et d'autres restrictions. Face au problème, des groupes de développeurs ont commencé à créer de nouveaux moyens basés sur XML pour créer la hiérarchie (ou la grammaire) que le document XML doit suivre. La norme officielle est le langage XML Schema du W3C, mais celui-ci souffre également de plusieurs handicaps, principalement le fait que la création de règles complexes peut être difficile. Pour contourner ce problème, la communauté du logiciel libre a combiné deux propositions différentes, TREX et RELAX, pour former RELAX NG, un langage de schéma basé sur XML qui offre beaucoup des fonctionnalités similaires de celles de XML Schéma mais beaucoup plus simple.

L'une des caractéristiques de RELAX NG c'est que plusieurs documents d'instance XML peuvent être validés par un seul document de schéma RELAX NG (Ou un seul document d'instance XML peut être validé par plusieurs documents de schéma RELAX NG).

On peut citer quelques autres caractéristiques de RELAX NG :

- Il s'agit d'une grammaire à base des modèles reposant sur une base mathématique solide,
- Il prend en charge les types de données de XML Schema.
- Il prend en charge les types de données définis par l'utilisateur.
- Il prend en charge les espaces de noms XML.
- C’est un langage hautement modulaire.
- Il traite les éléments et les attributs de la même manière.

RELEAX NG a deux syntaxes différentes, la syntaxe XML et la syntaxe compacte. Les extensions de fichier de RELAX NG sont ".rng" pour une syntaxe régulière et ".rnc" pour une syntaxe compacte.

La Figure 4 et la Figure 5 modélisent les documents XML qui stockent les informations des profils personnels (Figure 1) en utilisant le langage de schéma RELAX NG. La Figure 4 utilise la syntaxe XML et la Figure 5 utilise syntaxe compacte.

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar
    xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
    xmlns="http://relaxng.org/ns/structure/1.0" datatypeLibrary="http://www.w3.org
        /2001/XMLSchema-datatypes">
    <define name="profile">
        <element name="profile">
            <ref name="attlist.profile"/>
            <ref name="proprietaire"/>
        </element>
    </define>
    <define name="attlist.profile" combine="interleave">
        <empty/>
    </define>
    <define name="proprietaire">
        <element name="proprietaire">
            <ref name="attlist.proprietaire"/>
            <ref name="NomComplet"/>
            <ref name="Telephone"/>
            <ref name="Adresse"/>
            <ref name="Email"/>
            <ref name="Formation"/>
            <ref name="Competences"/>
        </element>
    </define>
    <define name="attlist.proprietaire" combine="interleave">
        <attribute name="age"/>
        <attribute name="type">
            <data type="NMTOKEN"/>
        </attribute>
    </define>
    <define name="NomComplet">
        <element name="NomComplet">
            <ref name="attlist.NomComplet"/>
            <ref name="Prenom"/>
            <ref name="Nom"/>
        </element>
    </define>
    <define name="attlist.NomComplet" combine="interleave">
        <empty/>
    </define>
```

Figure 4: Extrait de schéma RELAX NG (XML) qui modélise les profils personnels

```
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

profile = element profile { attlist.profile, proprietaire }
attlist.profile &= empty
proprietaire =
  element proprietaire {
    attlist.proprietaire,
    NomComplet,
    Telephone,
    Adresse,
    Email,
    Formation,
    Competences
  }
attlist.proprietaire &=
  attribute age { text },
  attribute type { xsd:NMTOKEN }
NomComplet = element NomComplet { attlist.NomComplet, Prenom, Nom }
attlist.NomComplet &= empty
Telephone =
  element Telephone {
    attlist.Telephone, Maison, Travail, Fax, portable
  }
attlist.Telephone &= empty
Adresse =
  element Adresse { attlist.Adresse, Rue, Ville, Province, codePostal }
attlist.Adresse &= attribute type { xsd:NMTOKEN }
Email = element Email { attlist.Email, text }
attlist.Email &= empty
Formation = element Formation { attlist.Formation, Institution+ }
attlist.Formation &= empty
Competences =
  element Competences { attlist.Competences, (LangagesProg | Systeme)+ }
attlist.Competences &= empty
Prenom = element Prenom { attlist.Prenom, text }
```

Figure 5: Extrait de schéma RELAX NG (Compact) qui modélise les profils personnels

2. Bases de données relationnelles

2.1. Modèle relationnel

Le modèle relationnel a été mis au point par Edgar Frank Codd dans les années 1970 par son article "A Relational Model of Data for Large Shared Data Banks" [2]. Mais, ses mises en œuvre commerciales ont été observées dans les années 1980 [37]. Le modèle relationnel est la base théorique des bases de données relationnelles.. Il utilise un ensemble de tables pour représenter à la fois les données et les relations entre ces données. Chaque table comporte

plusieurs colonnes, et chaque colonne a un nom unique. Le modèle relationnel se situe à un niveau d'abstraction plus bas que le modèle entité-association. La conception des bases de données est souvent effectuée dans le modèle entité-association, puis traduite dans le modèle relationnel [38].

Les éléments de base du modèle relationnel ont été développés sur le concept de relation mathématique, ses concepts théoriques sont basés sur la théorie des ensembles et la logique du premier ordre [39]. Le modèle relationnel est populaire grâce à sa simplicité et la possibilité de cacher les détails de la mise en œuvre de bas niveau au développeur et aux utilisateurs de la base de données.

2.2. Systèmes de gestion de bases de données relationnelles

En général, les bases de données stockent des ensembles de données qui peuvent être interrogées pour être utilisées dans d'autres applications. Un système de gestion de base de données permet de développer, d'administrer et d'utiliser des plates-formes de bases de données.

Un système de gestion de base de données relationnelle (SGBDR) est un type de systèmes de gestion de bases de données (SGBD) qui stocke les données dans une structure de table à base de lignes et utilisent le langage SQL (Structured Query Language) pour accéder à la base de données. Un SGBDR comprend des fonctions qui maintiennent la sécurité, l'exactitude, l'intégrité et la cohérence des données. Ce système est différent du stockage de fichiers utilisé dans un SGBD [40].

Le SGBDR est le système de base de données le plus populaire [41]. Il fournit une méthode fiable de stockage et de récupération de grandes quantités de données tout en offrant une combinaison de performance du système et de facilité de mise en œuvre.

Les fonctions les plus élémentaires du SGBDR sont liées aux opérations de d’insertion, de lecture, de mise à jour et de suppression des données - collectivement appelées CRUD. Le SGBDR fournit généralement des dictionnaires de données et des collections de métadonnées qui sont utiles pour le traitement des données. Ces derniers maintiennent la structure de données et les relations entre les tables [42].

Au fur et à mesure que les SGBDR ont mûri, ils ont atteint des niveaux d'optimisation des requêtes de plus en plus élevés et sont devenus des éléments clés des applications de

reporting, d'analyse et d'entreposage de données pour les entreprises. Ils sont intrinsèques aux opérations de diverses applications des entreprises.

2.2.1. Principe de fonctionnement

Un SGBDR stocke les données sous forme de table. Chaque base de données a un nombre variable de tables, chaque table possédant sa propre clé primaire unique. La clé primaire est ensuite utilisée pour identifier chaque ligne d'une table.

Une table est une structure bidimensionnelle composée de lignes (tuples, enregistrements) et de colonnes (attributs, champs). Chaque ligne est unique et stocke des données sur une entité. Chaque colonne a un nom d'attribut unique. toutes les entrées d'une colonne ont le même type de données.

2.2.2. Avantages des systèmes de gestion de bases de données relationnelles

Le SGBDR présente plusieurs avantages tels que [42], [43] :

- Les administrateurs de la base de données peuvent facilement maintenir, contrôler et mettre à jour les données dans la base de données. Les sauvegardes deviennent également plus faciles puisque les outils d'automatisation inclus dans le SGBDR automatisent ces tâches.
- Puisque le SGBDR stocke les données sous forme de table, elles sont facilement compréhensibles par les utilisateurs.
- Les données ne sont stockées qu'une seule fois et il n'est donc pas nécessaire de modifier plusieurs enregistrements. La suppression et la modification des données deviennent également plus simples et l'efficacité du stockage est très élevée.
- L'accès aux données est privilégié, ce qui signifie que l'administrateur de la base de données a le pouvoir de donner accès aux données à certains utilisateurs particuliers, ce qui sécurise les données.

2.3. Concepts de base des systèmes de gestion des bases de données relationnelles

Le modèle de données relationnel a été développé pour la première fois en 1970 par E.F. Codd[2]. En 1985, Codd a publié une liste de 12 règles qui définissaient de manière concise un système de gestion des bases de données relationnelles idéal [44], [45]. Ces règles ont été utilisées comme des lignes directrices pour la conception de tous les systèmes de bases de données relationnelles depuis lors. Ci-dessous quelques concepts proposés par E.F. Codd :

2.3.1. Unicité

- Clé candidate : Une clé candidate est une clé qui peut être utilisée pour identifier de manière unique tout enregistrement dans une table sans faire référence à d'autres données. Une clé candidate peut représenter une seule colonne ou une combinaison de plusieurs colonnes. Une table peut avoir plus d'une clé candidate.
- Clé primaire : Une clé primaire est une clé par laquelle tout enregistrement peut être identifié de manière unique dans une table. Une table peut avoir plusieurs clés candidates. Mais une seule de ces clés peut être considérée ou choisie comme clé primaire. Les clés primaires garantissent l'unicité des enregistrements dans une table et réduisent la redondance des données. Elle peut être définie comme une colonne unique ou une combinaison de plusieurs colonnes.
- Clé étrangère : Une clé étrangère représente une colonne ou un ensemble de colonnes dans une table qui fait référence à la clé primaire d'une autre table afin d'identifier de manière unique un enregistrement de cette table.

2.3.2. Normalisation

Codd a initialement décrit le problème des domaines complexes[2] qui peuvent être décomposés en sous-domaines indépendants mais reliés entre eux par des clés primaires et des clés étrangères. Ce processus était appelé la normalisation. Ce processus garantit que les données sont isolées pour chaque sous-domaine et que chaque sous-domaine contient les données qui ne concernent que ce domaine, mais qui sont toutes liées les unes aux autres par des clés étrangères. Le processus de décomposition des domaines en sous-domaines implique qu'une table est divisée en plusieurs tables, ce qui rend les opérations de la base de données relativement simples et élimine la redondance des données. La normalisation est généralement décrite par diverses formes normales. Lorsque la structure d'une relation répond à certains ensembles de conditions, elle peut être considérée comme elle est en une forme normale particulière. Les formes normales courantes comprennent la première forme normale, la deuxième forme normale, la troisième forme normale et la forme normale de Boyce/Codd (BCNF)[46].

Au fur et à mesure qu'un schéma de base de données se normalise, les redondances dans les données sont décomposées en d'autres tables connexes. La normalisation est généralement utilisée pour répartir les données afin de réduire le nombre des opérations d’entrée/sortie dans

la base de données lorsque seule une partie des données doit être retournée. Lorsque les tables sont normalisées, elles peuvent être dénormalisées à nouveau en réunissant les tables décomposées. Le schéma dénormalisé est plus rapide pour récupérer de grands ensembles de données, mais peut prendre plus d'espace de stockage en raison des redondances dans les données.

2.3.3. Intégrité et fiabilité des données

Tous les SGBDR garantissent la cohérence des données. La cohérence des données est très importante car des données erronées ou corrompues peuvent être préjudiciables à toute entreprise. Le SGBDR assure la cohérence des données grâce aux propriétés ACID (Atomicité, Cohérence, Isolation et Durabilité)[47]. L'acronyme ACID fait référence aux quatre propriétés dans le contexte du traitement des transactions du SGBD. Elles sont définies comme suit [47]:

- Atomicité : Dans une transaction comportant deux ou plusieurs éléments d'information, soit toutes les informations sont engagées pour être sauvegardées, soit aucune n'est sauvegardée. Essentiellement, la règle « all or nothing » est appliquée.
- Cohérence : Cette propriété signifie que les contraintes d'intégrité doivent être maintenues afin que la base de données soit cohérente avant et après la transaction. Il s'agit de l'exactitude d'une base de données.
- Isolation : Cette propriété garantit que plusieurs transactions peuvent s’exécuter simultanément sans entraîner l'incohérence de l'état de la base de données. Les transactions se produisent indépendamment sans interférence. Les changements survenant dans une transaction particulière ne seront visibles pour aucune autre transaction jusqu'à ce que ce changement particulier dans cette transaction soit écrit en mémoire ou ait été validé. Cette propriété garantit que l'exécution simultanée de transactions aboutira à un état équivalent à celui obtenu lors de leur exécution en série dans un certain ordre.
- Durabilité : Cette propriété garantit qu'une fois l'exécution de la transaction terminée, les mises à jour et les modifications de la base de données sont stockées et écrites sur le disque et qu'elles persistent même si une panne du système se produit. Ces mises à jour deviennent alors permanentes et sont

stockées dans une mémoire non volatile. Les effets de la transaction ne sont donc jamais perdus.

2.4. Langage SQL

SQL (Structured Query Language) est un langage de base de données conçu pour gérer les données contenues dans un système de gestion de base de données relationnelle. SQL a été initialement développé par IBM au début des années 1970 [48]. La version initiale, appelée SEQUEL, elle était conçue pour manipuler et récupérer les données stockées dans le système de gestion de base de données quasi-relationnelle d'IBM, System R. Puis, à la fin des années 1970, la société Relational Software Inc qui est aujourd'hui Oracle Corporation, a introduit la première implémentation de SQL disponible dans le marché, Oracle V2 pour les ordinateurs VAX [49], [50].

Au fil des années, SQL est resté un choix populaire pour les utilisateurs de bases de données, principalement en raison de sa facilité d'utilisation et de la manière très efficace dont il interroge, manipule, agrège les données et exécute un large éventail d'autres fonctions pour transformer des collections massives de données structurées en informations utilisables.

Le langage SQL comporte 4 grandes parties [48]–[50] :

- Langage de définition des données (LDD) : un schéma relationnel est un ensemble de tables relationnelles et d'éléments associés qui sont liés les uns aux autres (vues, index, rôles d'utilisateur, procédures stockées, etc…). SQL fournit un ensemble des instructions pour gérer ces éléments.

  Exemple : CREATE, ALTER, TRUNCATE et DROP
- Langage de manipulation de données (LMD) : Le langage de manipulation de données spécifie comment les données sont insérées, mises à jour, récupérées et supprimées dans les objets définis à l'aide du langage de définition des données (LDD).

  Exemple : SELECT, UPDATE, DELETE et INSERT
- Langage de contrôle de données (LCD) : SQL comprend des instructions pour gérer les privilèges des utilisateurs. Elles permettent d'attribuer ou de révoquer ces privilèges. Les privilèges sont les actions qu'un utilisateur est autorisé à effectuer sur les objets de la base de données tels que les tables et les vues.

  Exemple : GRANT et REVOKE

- Langage de contrôle des transactions (LCT) : Ces instructions sont utilisées pour gérer les transactions dans la base de données. Elles sont utilisées pour gérer les modifications apportées par les instructions du langage de manipulation de données.

Exemple : COMMIT, ROLLBACK et SAVEPOINT.

3. Big Data

Selon Gartner[51], la société de recherche et de conseils en technologies de l'information, le Big Data est un ensemble des ressources d'informations avec trois caractéristiques principales à savoir Volume, Vélocité et Variété qui exigent une plateforme innovante pour une meilleure compréhension et prise de décision. En plus des trois "V" qui sont largement présents dans la littérature, de nombreux auteurs[52]–[54] indiquent que d'autres "V" ont été ajoutés pour mieux définir le Big Data tels que Visibilité, Véracité et Valeur.

Nous présentons dans cette section deux outils clés de la technologie Big Data à savoir Hadoop et les bases de données NoSQL.

3.1. Hadoop

Apache Hadoop est une plateforme logicielle open-source, écrite en Java. Elle gère le traitement et le stockage des données pour les applications big data [55]. Hadoop fonctionne en distribuant de grands ensembles de données et des tâches d'analyse sur les nœuds d'un cluster de calcul et les décomposant en charges de travail plus petites qui peuvent être exécutées en parallèle[56]. Hadoop peut traiter des données structurées et non structurées, et évoluer de manière fiable d'un seul serveur à des milliers de machines.

Apache Hadoop est né de la nécessité de traiter des volumes croissants de données volumineuses. Inspiré par MapReduce de Google, un modèle de programmation qui divise une application en petites fractions à exécuter sur différents nœuds, Doug Cutting et Mike Cafarella ont lancé Hadoop en 2005 lorsqu'ils travaillaient sur le projet Apache Nutch[57]. Selon Vasuja *et al.* [58], Doug a donné à Hadoop le nom de l'éléphant en peluche de son fils. Quelques années plus tard, Hadoop s'est détaché de Nutch et, en conséquence, Yahoo a publié Hadoop en tant que projet open-source en 2008. La fondation logicielle Apache a mis Hadoop à la disposition du public en novembre 2012.

Le noyau de Hadoop comporte cinq composants principaux, à savoir[59] :

- HDFS : c’est un système de fichiers distribué qui stocke les données sur des machines, offrant une bande passante très élevée dans le cluster.
- Hadoop MapReduce : c’est une implémentation du modèle de programmation MapReduce introduit par Google pour le traitement de données à grande échelle.
- Hadoop YARN : c’est un Framework de gestion des ressources des clusters et de planification des tâches.
- Hadoop Common : Il s'agit d’une collection des bibliothèques Java et d'utilitaires requis par d'autres modules Hadoop comme Hive, Impala, HBase, Pig, Spark Mahout, Oozie, etc...
- Hadoop Ozone : c’est un système de stockage d'objets évolutif, redondant et distribué pour Hadoop.

Malgré l'émergence d'options alternatives, notamment dans le cloud, Hadoop reste une technologie importante et précieuse pour les utilisateurs de big data pour les raisons suivantes[60]:

- Il peut stocker et traiter rapidement de grandes quantités de données structurées, semi-structurées et non structurées.
- Il protège les applications et le traitement des données contre les pannes matérielles. Si un nœud d'un cluster tombe en panne, les travaux de traitement sont automatiquement redirigés vers d'autres nœuds pour garantir que les applications continuent de s'exécuter.
- Il n'est pas nécessaire de prétraiter les données avant de les stocker. Les entreprises peuvent stocker des données brutes dans HDFS et décider ultérieurement comment les traiter et les filtrer pour des utilisations analytiques spécifiques.
- Il est évolutif, de sorte que les entreprises peuvent facilement ajouter des nœuds supplémentaires pour permettre à leurs systèmes de traiter davantage de données.
- Il peut prendre en charge les analyses en temps réel afin d'améliorer la prise de décision opérationnelle, ainsi que les charges de travail par lots pour les analyses historiques.

Dans ce qui suit, nous allons expliquer davantage les deux piliers centraux de Hadoop à savoir le système de fichier HDFS et le Framework Hadoop MapReduce.

3.1.1. Système de fichier HDFS

Lorsqu'un ensemble de données dépasse la capacité de stockage d'une seule machine physique, il devient nécessaire de le répartir sur plusieurs machines distinctes. Les systèmes de fichiers qui gèrent le stockage à travers un réseau informatique sont appelés systèmes de fichiers distribués. Comme ils sont basés sur le réseau, toutes les complications de la programmation réseau entrent en jeu, ce qui rend les systèmes de fichiers distribués plus complexes que les systèmes de fichiers sur disque ordinaires. Par exemple, l'un des plus grands défis consiste à faire en sorte que le système de fichiers puisse tolérer une panne de nœud sans subir de perte de données[55].

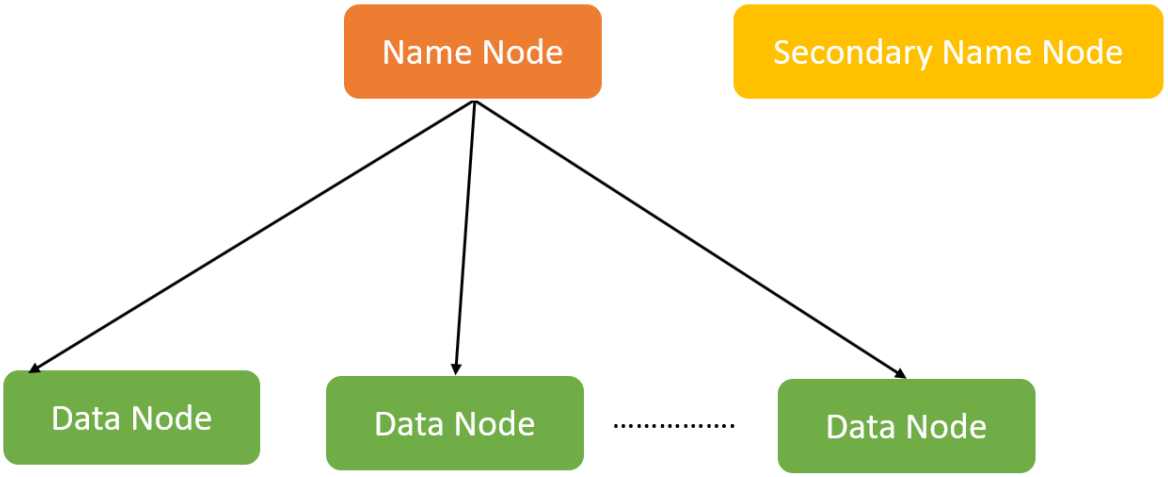


Figure 6: L'architecture globale de HDFS

Hadoop est livré avec un système de fichiers distribué appelé HDFS (Hadoop Distributed Filesystem), c'est la couche de stockage par défaut pour Hadoop. HDFS est basé sur l'architecture maître-esclave et il comporte les éléments suivants :

- Namenode: c'est l'élément central du HDFS. Il est généralement appelé maître et il est conçu pour stocker les métadonnées. Le Namenode est chargé de surveiller l'état de santé des nœuds esclaves et d'attribuer les tâches aux Datanode.
- Datanode: c'est l'élément qui stocke réellement les données. Il joue le rôle du nœud esclave. Les Datanodes servent les demandes de lecture et d'écriture des clients du système de fichiers. En outre, ils effectuent la création, la suppression et la réplication de blocs lorsque le Namenode leur demande de le faire.

- Secondary Namenode : la fonction principale de ce nœud est de prendre des copies des métadonnées du système de fichiers présentes sur le Namenode pour renforcer la tolérance en panne.

Les figures 6 et 7 présentent respectivement l'architecture globale et l'architecture détaillée de HDFS.

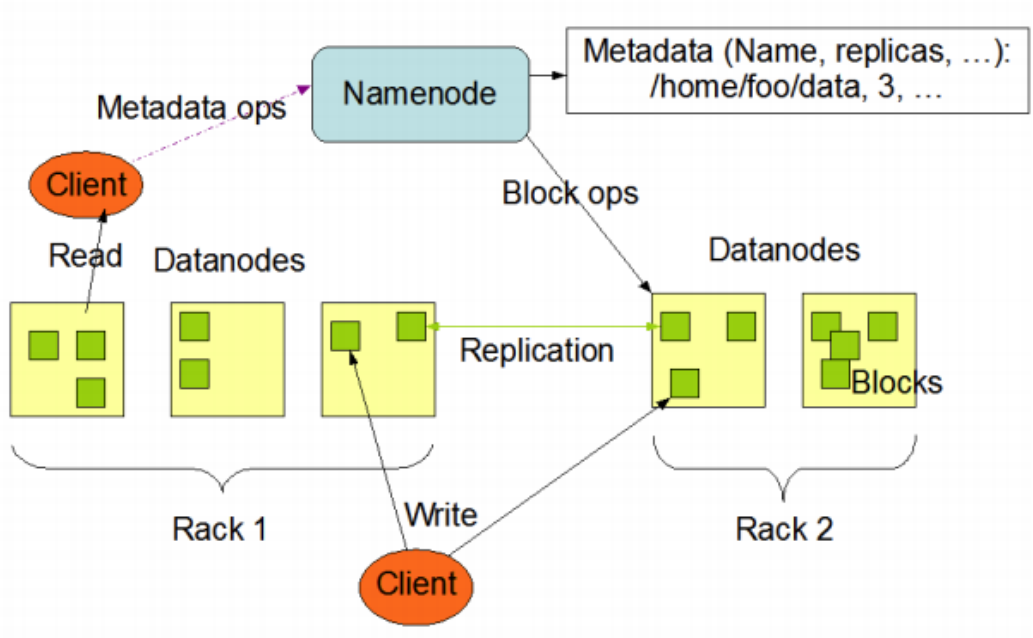


Figure 7: L'architecture détaillée de HDFS [61]

3.1.2. Hadoop MapReduce

Hadoop MapReduce est un paradigme de traitement et un modèle de programmation pour le calcul parallèle et souvent distribué basé sur Java. C'est une implémentation de l'algorithme proposé par Google en 2004 dans l'article intitulé "*MapReduce : Simplified Data Processing on Large Clusters*"[62].

MapReduce s'appuie sur le fameux concept de « Diviser pour mieux régner », où un problème unique est divisé en plusieurs sous-tâches individuelles. Cette approche devient encore plus puissante lorsque les sous-tâches sont exécutées en parallèle ; dans un cas parfait, une tâche qui prend 100 minutes pourrait être traitée en 1 minute par 100 sous-tâches parallèles.

L'algorithme MapReduce fonctionne en décomposant le traitement en deux phases, à savoir Map et Reduce. Hadoop fournit une interface standard pour les fonctions Map et Reduce, et les implémentations de celles-ci sont souvent appelées *Mappers* et *Reducers*. La

phase Map, est un ensemble des tâches dont les données d'entrée sont des tuples sous la forme d'une paire clé-valeur. La sortie de *Mapper* est transmise au *Reducer* comme entrée[63]. Le *Reducer* ne s'exécute que lorsque le Mapper est terminé. Le *Reducer* prend également en entrée des données sous le format de clé-valeur, et la sortie du *Reducer* est la sortie finale[64]. MapReduce fournit à travers les fonctions Map et Reduce une série de transformations d'un ensemble de données source à un ensemble de données résultat. Le développeur se contente de définir les transformations de données ; et MapReduce gère le processus d'application de ces transformations aux données à travers les machines du cluster.

La Figure 8 présente le processus de traitement des données dans MapReduce.

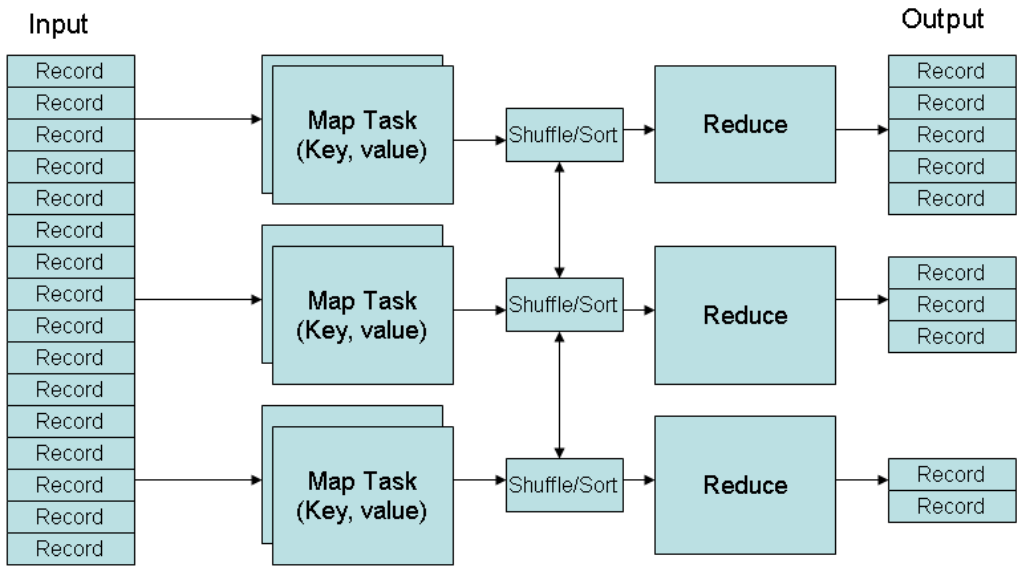


Figure 8: Processus de traitement des données dans MapReduce [65]

3.2. Bases de données NoSQL

Après l'apparition du modèle relationnel en 1970 par Edgar Frank Codd [2], les bases de données relationnelles se sont avérées être la technologie prédominante pour le stockage de données structurées dans les applications informatiques [66]. Ces dernières évoluent également avec le temps et posent des exigences difficiles pour la gestion des données. Avec l'avènement des applications Web 2.0, une quantité énorme de données a été collectée et aussi un nombre important d'utilisateurs accède simultanément à ces données plus que jamais auparavant [67]. Cela signifie que l'évolutivité et les performances sont plus que jamais un défi pour les bases de données relationnelles. Alors, les bases de données ont dû s'adapter à des charges d'applications de type OLTP/OLAP où des dizaines de millions d'utilisateurs lancent des requêtes de lecture et d'écriture de données. Ces bases de données doivent offrir

une bonne évolutivité horizontale pour les opérations de lecture/écriture réparties sur de nombreux serveurs. Les systèmes de bases de données relationnelles ont peu de possibilités d'évoluer horizontalement à ces niveaux. Cela a donc ouvert la voie à la recherche des solutions alternatives pour les scénarios dont les systèmes de bases de données relationnelles se sont avérés inconvenables[68].

Le problème de l'évolutivité des bases de données relationnelles a été reconnu par les géants du Web ayant des besoins énormes et croissants en matière de données et d'infrastructure, comme Google, Amazon et Facebook. Ils ont proposé leurs propres solutions au problème comme BigTable, DynamoDB et Cassandra.

Cet intérêt croissant a donné naissance à un certain nombre de systèmes de gestion de bases de données (SGBD) NoSQL, axés sur la performance, la fiabilité et la cohérence. Au début, Il y a eu des bases de données NoSQL propriétaires (à source fermée) développées par les grandes entreprises pour répondre à leurs besoins spécifiques, comme DynamoDB d'Amazon et BigTable de Google qui est considéré comme le premier système NoSQL. Le succès de ces systèmes propriétaires a entraîné le développement d'un certain nombre de systèmes de bases de données NoSQL propriétaires et Open Source similaires, les plus populaires étant MongoDB, Cassandra, MongoDB, Neo4j, RavenDB, HBase, Couchbase et Redis.

Selon Davoudian *et al.* [1], il existe quatre catégories de bases de données NoSQL :

- Bases de données Clé/Valeur : Cette catégorie de systèmes de gestion des données utilise une structure de stockage similaire topologiquement à un simple tableau associatif [69]. Un enregistrement dans la base de données est une paire clé-valeur. La clé est un identifiant indexé de manière unique qui est attribué à une valeur. La valeur peut être de type de données scalaires tels que des entiers ou des structures complexes telles que JSON, des listes, BLOB, etc. Cela facilite le stockage de données non structurées et semi-structurées [1]. Ces bases de données n'ont pas de langage d'interrogation, mais elles permettent d'effectuer les opérations CRUD sur les clés. Les valeurs ne peuvent pas faire l'objet d'une interrogation ou d'une recherche. Seule la clé peut être interrogée [1], [70]. Ce type de bases de données est optimisé pour la gestion des sessions dans les applications web, la gestion des sessions pour les jeux multi-joueurs en ligne et les recommandations en temps réel.

Exemples : Amazon DynamoDB, Voldemorte, Oracle Berkeley DB et Aerospike.

- Bases de données orientées documents : Elles stockent et récupèrent les données sous forme de paires clé-valeur, mais la partie valeur est stockée sous forme de document aux formats JSON, BSON, XML ou YAML [70]. Elles sont des systèmes de base de données sans schéma « schema-less », il n'y a pas une telle structure dans une base de données, mais plutôt de différentes structures dans les documents. Pour le stockage, cela ne semble pas être un problème, car les documents ne dépendent pas les uns des autres. Ces bases de données supportent efficacement l'indexation primaire sur l'identifiant du document et l'indexation secondaire est également prise en charge [1]. Les données sont stockées sur le disque pour assurer la persistance. Les index et les données fréquemment consultées sont stockés dans la mémoire cache pour permettre un accès plus rapide. Ce type de bases de données est principalement utilisé pour les CMS, les applications de commerce électronique, l'analyse des données en temps réel, etc.

  Exemples : OrientDB, IBM Cloudant, CouchDB, MongoDB, Cosmos DB et RavenDB.

- Bases de données orientées colonnes : Elles sont des systèmes de base de données qui stockent les données en colonnes et sont basées sur le papier de BigTable de Google [71]. Chaque colonne est traitée séparément. Les valeurs de la même colonne sont stockées de manière contiguë. Ces bases de données offrent des performances élevées pour les requêtes d'agrégation telles que SUM, COUNT, AVG, MIN, etc. vu que les données de la même colonne sont stockées de manière adjacente [69], [70]. Les bases de données orientées colonnes sont largement utilisées dans l'informatique décisionnelle et sont également utilisées pour gérer les entrepôts de données ainsi que les solutions de CRM.

  Exemples : BigTable, Cassandra, HBase, MonetDB et Hypertable.

  Nous allons expliquer davantage ce type de bases de données dans le Chapitre IV.

- Bases de données orientées graphe : Une base de données orientée graphe stocke les entités ainsi que les relations entre ces entités. L'entité est stockée sous forme de nœud et les relations sous forme d'arêtes (orientées ou non). Un nœud ou une arête peut avoir des propriétés. Ce type de SGBD est largement

utilisé pour la détection des fraudes, les moteurs de recommandation en temps réel et dans les réseaux sociaux [72]

Exemples : Neo4j, FlockDB, InfoGrid, AllegroGraph et JanusGraph.

3.2.1. Propriétés BASE

Les bases de données relationnelles sont conçues dans un souci de fiabilité et de cohérence. Ce type de base de données repose sur le modèle transactionnel qui garantit les quatre propriétés du principe ACID [73]. Cependant, l'avènement d'un nouveau modèle de base de données non structuré bouleverse le modèle ACID. Le modèle de base de données NoSQL délaisse le modèle relationnel hautement structuré au profit d'une approche flexible et aisée pour stocker les données. Dans le monde des bases de données NoSQL, l'utilisation des transactions est moins à la mode car certaines bases de données ont relâché les exigences de cohérence immédiate et de précision des données afin de bénéficier d'autres avantages, comme l'évolutivité et la résilience[70]. Ce nouveau type de base de données nécessite alors un alternatif au modèle ACID; c'est le modèle BASE[74].

L'acronyme BASE signifie :

- *Basically Available* (Disponibilité basique) : cela indique que le système reste disponible même en cas de dysfonctionnement d'un nœud du cluster.
- *Soft state* (Cohérence légère) : l'état des données pourrait changer sans interaction avec l'application en raison d'une éventuelle cohérence.
- *Eventually consistent* (Cohérence à terme) : cela signifie que le système deviendra cohérent au fil du temps et pas forcément après chaque opération.

3.2.2. Théorème de CAP

En 2000, le professeur Eric Brewer a proposé le fameux théorème CAP [75]. CAP est l'abréviation de *Consistency*, *Availability* and *Partition Tolerance* (Cohérence, Disponibilité, Tolérance au partitionnement). Le théorème CAP est un théorème fondamental des systèmes distribués qui stipule que tout système distribué peut avoir au plus deux des trois propriétés suivantes [76]:

- Cohérence : un système est dit cohérent si tous les nœuds voient les mêmes informations en même temps, ce qui implique que lorsqu'une information est écrite sur un nœud, les données sont répliquées sur le reste des nœuds du système.

- Disponibilité : signifie que chaque demande faite au système aura une réponse. Ainsi, tous les clients seront en mesure de lire et d'écrire dans le système.
- Tolérance au partitionnement: signifie que le système continuera à fonctionner même si une partie des nœuds est déconnectée ou devient indisponible.

Les systèmes de gestion de bases de données peuvent être alors classer en systèmes cohérents et disponibles (*CA*), disponibles et tolérants au partitionnement (AP) ou cohérents et tolérants au partitionnement (CP). Cette classification est illustrée dans la Figure 9.

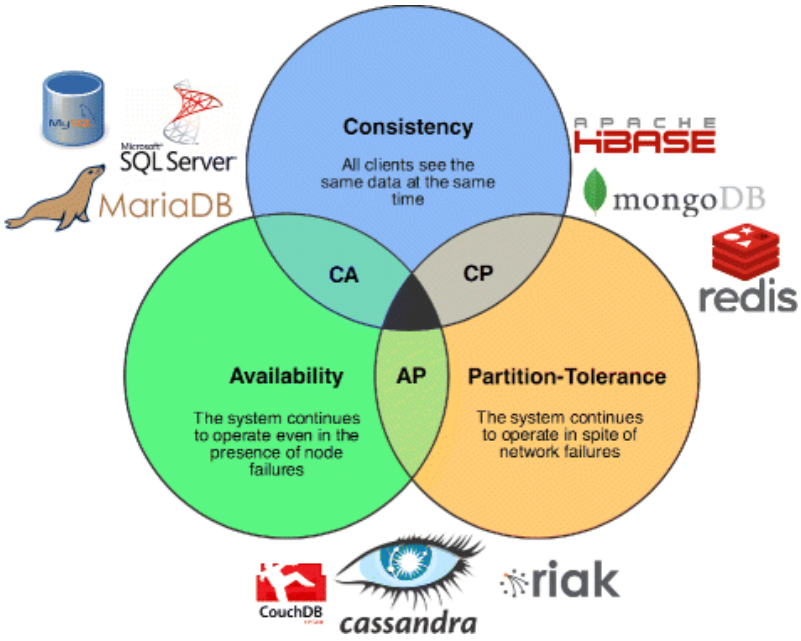


Figure 9: Classification des systèmes de gestion bases de données selon le théorème CAP[77]

Partie II
Contributions

Chapitre II

XMap : une Nouvelle Approche pour Stocker, Interroger et Extraire les Documents XML à partir de Bases de Données Relationnelles

1. Introduction

XML (eXtensible Mark-up Language) est devenu une norme de facto pour la représentation et l'échange de données sur le web et dans une grande variété de domaines, tels que les données scientifiques, financières, commerciales, sanitaires, industrielles et astronomiques. Le XML est une technologie multi-plateforme, indépendante des logiciels et des matériels [78], elle est largement reconnue pour assurer la portabilité des données à l'aide de balisage structurels et sémantiques. L'imbrication des documents XML et le fait qu'ils contiennent à la fois les données elles-mêmes et les informations sur leur structure (auto-description de structure) fournissent un moyen simple et flexible pour les applications pour modéliser et échanger les données d'une manière lisible par l'homme et facilement vérifiables, transformées et publiées.

Il existe quatre approches pour le stockage des données XML[9]. La première approche est les bases de données XML native qui prennent en charge le modèle de données XML et les langages de requêtes XML. La deuxième consiste à stocker les données dans des bases de données relationnelles, la troisième approche reposer sur l'utilisation des fichiers et la dernière elle permet de stocker les données XML dans des bases de données orientées objets [10], [11]. La première approche souffre de plusieurs limitations (sécurité, transactions et gestion d'accès concurrents, renvoie les données uniquement sous format XML) [16], [17]. Afin de surmonter ces limitations, plusieurs approches ont été proposées [18]–[26], pour tirer parti des technologies matures fournies par les SGBDR actuels (*i.e*. les index, les déclencheurs, l'intégrité des données, la sécurité et l'optimisation des requêtes SQL), et aussi pour exploiter toute la puissance de cette technologie. Ces approches peuvent être classées en deux grandes catégories:

- **Mapping à base de schémas :** Cette approche utilise la structure d'un schéma (DTD ou XML Schema) d'un document XML afin de générer le schéma relationnel pour le stockage d'un document XML [20], [23], [25], [79], [80]. La structure de la base de données relationnelle est générée par utilisation des clés primaires et des clés étrangères qui reposent sur une relation parent-enfant dans l'arbre de données XML (extraite de la DTD ou du XML Schema).
- **Mapping générique :** Le concept fondamental de cette approche est d'utiliser un schéma générique pour stocker les documents XML dans une base de données relationnelle sans l'existence de DTD ou de XML Schema. Le principe

de cette approche est de capturer l'arborescence du document XML [18], [19], [21], [22], [24], [26], [81]–[83].

Selon Atay *et al.* [84], le processus de stockage des documents XML dans une base de données relationnelle comporte quatre étapes essentielles :

1. **Mapping du schéma :** cette première étape consiste à générer le schéma relationnel correspondant au fichier XML à partir du schéma XML (DTD ou XML Schema) ou à partir du fichier XML lui-même (en cas d'absence du schéma XML).
2. **Mapping des données :** le but de cette étape est de sauvegarder les données XML dans la base de données relationnelle générée lors de la première étape, sous forme de tuples relationnels.
3. **Mapping des requêtes :** l'objectif est de convertir les requêtes XML (XPath ou XQuery) en requêtes SQL et de les exécuter ensuite dans la base de données relationnelle.
4. **Mapping inversé des données :** Le résultat de l'étape précédente (mapping des requêtes) est généré sous forme des données relationnelles. Ces données doivent être traduites en suite en données XML selon la structure de la requête XML d'entrée.

L'idée principale de cette contribution est de présenter une nouvelle approche qui effectue le mapping du schéma et le mapping des données. Notre contribution est une approche hybride qui bénéficie des avantages des approches basées sur le chemin [21], [22], [83], [85], [86] et des avantages de l'approche *ORDPATH* (approche d'étiquetage XML) [87]. Notre approche supporte efficacement les modifications structurelles de l'arbre XML (Insertion, modification et suppression).

2. Travaux connexes

De nombreuses études ont été établis sur le domaine du stockage et de la récupération de documents XML à partir des bases de données relationnelles à savoir *Edge* [19], *XRel* [21], *XParent* [85], *Xlight* [28] et *Mini-XML* [26].

4.1. XRel

L'approche *XRel* [21] déchiquette un document XML en quatre tables comme suit :

Path (PathID, Pathexp)
Element (DocId, PathID, Start, End, Ordinal)

Text (DocId, PathID, Start, End, Value)
Attribute (DocId, PathID, Start, End, Value)

L'approche *XRel* utilise le modèle de données XPath, dans lequel un document XML est représenté sous forme d'une structure de nœuds en arbre. Cette approche introduit un nouveau concept nommé : région ou inclusion (*containment*) pour maintenir la relation parent-enfant en utilisant deux attributs : *Start* et *End*. La table *Path* enregistre les différentes expressions de chemin d'accès existantes dans le document XML. La table *Element* stocke les informations de tous les nœuds de type élément dans l'arbre de données XML. La table *Text* enregistre les informations sur les nœuds texte. La table *Attribute* stocke les nœuds de type attributs dans l'arbre de données XML.

En outre, cette approche utilise le concept de région pour maintenir les relations entre ancêtres et descendants. Pour cela elle utilise les thêta-jointures (< ou >) dans la phase du mapping des requêtes XPath à des requêtes SQL, ce qui affecte la performance d'exécution des requêtes en raison de la manière dont un SGBDR traite les jointures [9], [22], [88].

4.2. XParent

L'approche *XParent* [85] consiste à stocker un document XML dans quatre tables :

LabelPath (ID, Len, Path)
DataPath (Pid, Cid)
Data (PathID, Did, Ordinal, Value)
Element (PathID, Did, Ordinal)
Ancestor (Did, Ancestor, Level)

La table *LabelPath* stocke les expressions de chemin d'accès dans l'attribut *Path*. Chaque expression de chemin d'accès est identifiée par *ID*, l'attribut *Len* enregistre le nombre des arêtes constituant chaque expression de chemin d'accès. La table *DataPath* stocke les informations relatives à la relation parent/enfant entre les nœuds de l'arbre de données XML. Le *Pid* est l'ID du nœud parent et le *Cid* est l'ID du nœud enfant. La table Data stocke les informations détaillées sur les nœuds de l'arbre de données XML. La table *Element* joue le même rôle que la table *Data*, mais elle n'enregistre pas les valeurs des *Element*. La table *Ancestor* est utilisée pour stocker tous les ancêtres d'un nœud.

L'encodage de région utilisé dans *XRel* est remplacé par un attribut unique, (*Did*), ce qui améliore le temps de traitement des requêtes dans la phase du mapping des requêtes XPath à des requêtes SQL par l'utilisation des équi-jointure au lieu de théta-jointure. Cependant, cette approche utilise la table *Ancestor* pour stocker tous les ancêtres d'un nœud ce qui entraîne une utilisation élevée d'espace de stockage en raison du stockage d'informations redondantes dans la table *Ancestor* [88].

4.3. Edge

L'approche Edge [17] enregistre toutes les informations des nœuds de l'arbre de données XML dans une seule table :

Edge (Source, Ordinal, Target, Label, Flag, Value**)**

La table *Edge* stocke l'identifiant du nœud source (*Source*) et le nœud cible (*Target*) de chaque arête de l'arbre de données XML. L'attribut *Ordinal* stocke l'ordre d'une arête par à ces arêtes adjacentes. L'attribut *Label* enregistre le nom du nœud. L'attribut *Flag* indique si cette arête pointe vers un objet (élément ou attribut) ou vers une valeur(texte). Si la valeur de l'attribut *Flag* est : val, l'attribut *Value* stocke la valeur de ce nœud ; sinon, cet attribut sera vide.

L'approche *Edge* possède des réelles limites au niveau de temps d'exécution des requêtes, parce qu'elle conserve l'étiquette de l'arête au lieu de l'étiquette des expressions de chemin d'accès. Alors pour traduire une requête XPath à une requête SQL qui sera exécutée ensuite dans la table *Edge* ; des chemins doivent être crées, pour cela, cette approche nécessite une concaténation des arêtes[89]. Par conséquent, plusieurs opérations de jointure sont requises pour valider un chemin.

4.4. Xlight

Zafari et ces collègues [82] ont proposé une approche nommée *Xlight* qui consiste à stocker les documents XML déchiquetés dans une base de données relationnelle. C'est une approche qui appartient au catégorie "Mapping générique" ; elle utilise un schéma générique pour stocker les documents XML dans la même base de données relationnelle sans l'existence de DTD ou de XML Schema. Ce schéma relationnel contient cinq tables :

Document (DocId, Name)
Path (PathId, Path)
Data (DocId, PathId, LeafNo, LeafGroup, LinkLevel, LeafValue, hasAttrib)

Ancestor (DocId, LeafGroup, AncestorPre, AncestorLevel)
Attribute (Name, Val, id, pre)

La table *Document* est utilisée pour stocker les noms des documents XML déchiquetés et stockés dans la base de données relationnelle. La table *Path* stocke toutes les expressions de chemin d'accès existantes dans le document XML. Un identifiant unique est stocké dans la colonne *PathId* pour faire référence à une expression de chemin d'accès. La table *Data* stocke les informations relatives à tous les nœuds feuilles dans le document XML. La table *Ancestor* conserve les informations relatives aux ancêtres des nœuds feuilles. Les informations des attributs existants dans les documents XML sont enregistrées dans la table *Attribute*.

La table *Ancestor* occupe moins d'espace par rapport à la table *Ancestor* de l'approche XParent, car il stocke seulement les informations des ancêtres d'un seul nœud feuille pour chaque groupe (Un group c'est l'ensemble des nœuds feuilles ayant le même parent) puisque tous les nœuds feuilles de même groupe ont les mêmes ancêtres. Mais en général cette approche a deux grandes limites ; la première concerne l'espace de stockage car elle stocke les données d'un document XML dans cinq tables et la deuxième est le temps de réponse qui s'augmente à cause de nombre élevé des jointures requise pour effectuer le mapping des requêtes XPath à des requêtes SQL [9], [90]

4.5. Mini-XML

L'approche *mini-XML* [26] contribue au mapping des documents XML vers les bases de données relationnels en proposant un schéma relationnel composé par deux tables :

PathTable (PathID, path, pos)
LeafTable (LeafID, name, value, pos)

mini-XML est une approche de mapping basée sur les expressions de chemin utilisés dans le langage XPath. XPath facilite le processus de recherche de données SQL en identifiant la relation entre les nœuds du document. L'objectif de cette approche est d'identifier les chemins des nœuds non-feuilles. Zhu *et al.* ont proposé deux tables, pour objectif de stocker les informations des nœuds feuilles et les informations sur les expressions de chemin d'accès des nœuds non-feuilles respectivement dans la table *LeafTable* et la table *PathTable*. Une phase de prétraitement se fait avant la phase de mapping afin d'étiqueter tous les nœuds de l'arbre de données XML en utilisant la technique d'étiquetage *Persistent* (persistent labelling scheme) sous la forme de (*Level*, [*P-pathID*, *S-order*]), où *Level* est la profondeur du nœud

feuille, *P-pathId* est l' expression de chemin d'accès du nœud parent, et *S-order* est la position du nœud parmi ses nœuds frères ; l'étiquetage des nœuds permet de l'identifier les relations structurelles entre les nœuds.

Zhu *et al*. [26] ont démontré expérimentalement que leur approche est plus performante que l'approche s-XML[91] au niveau d'espace de stockage et au niveau de temps de mapping. Néanmoins, Le point faible de cette approche est son manque d'une phase de mapping de requêtes XPath à des requêtes SQL.

4.6. L'approche de Ying

Ying *et al*.[88] ont proposé une approche de mapping des documents XML dans les bases de données relationnelles. C'est une technique hybride combinant l'approche basée sur les chemins et l'approche basée sur les nœuds. Cette approche permet de stocker les données XML dans une base de données relationnelles composées de quatre tables :

File(DocId, Name)
Path(PathId, PathExp)
LeafNode(DocId, LNodeId, PathId, ParentId, leafValue)
InnerNodes (DocId, INodeId, NodeName, ParentId, Level, SibilingNum)

La table *File* est utilisée pour gérer les informations des différents documents XML dans une base de données. La table *Path* permet de stocker les informations de chaque expression de chemin distincte. La table *LeafNode* sert à gérer les informations les nœuds feuilles. Les nœuds internes de l'arborescence sont stockés dans la table *InnerNodes*.

Le résultat des tests de comparaison avec les approches *XRel*[21] et *XParent*[85] montre que cette approche surpasse les autres approches au niveau de teste de temps d'exécution des requêtes et le teste de la taille de la base de données générée. L'inconvénient de technique réside dans le nombre élevé des tables nécessaires pour faire le mapping. Cela engendre un ralentissement d'exécution des requêtes causé par le nombre élevé de jointure ainsi que l'utilisation des auto-jointures.

4.7. XRecursive

XRecursive[92] est une approche de mapping des documents XML dans les bases de données relationnelles où l'expression de chemin de chaque nœud est identifiée par son parent à partir du nœud racine d'une manière récursive. Elle permet de stocker le document XML dans deux tables :

tag_structure(tagName, Id, pId)
tag_value(tagId, Value, Type)

La table *tag_structure* permet de stocker l'ID d'un élément, son nom et l'ID de son parent. La table *tag_value* permet d'enregistrer la valeur d'un élément et son type. L'expression de chemin de chaque nœud est identifiée de manière récursive en utilisant l'identifiant du parent sans stocker l'expression de chemin dans la base de données. Ce qui augmente le temps de traitement des requêtes.

4.8. XAncestor

Qtaish *et al*. ont proposé l'approche *XAncestor*[22] permettant de stocker les documents XML dans une base de données XML. C'est une approche basée sur le chemin. Elle consiste à stocker uniquement les informations des nœuds feuilles de l'arbre de données d'un document XML(en ignorant les nœuds non-feuille pour réduire l'espace de stockage). Cette approche stocke les documents XML dans deux tables :

Ancestor_Path(Ances_PathID, Ances_PathExp)
Leaf_Node (Node_Name, Ances_PathID, Ances_Pos, Node_Value)

La table *Ancestor_Path* permet de stocker les expressions de chemin distincts des nœuds feuille dans l'arbre de données XML. La table *Leaf_Node* enregistre les informations chaque nœud feuille dans l'arbre de données XML.

Qtaish et al. ont effectué une évaluation expérimentale pour comparer *XAncestor* avec les autres approches. Les résultats ont indiqué qu'elle est plus performante au niveau l'espace de stockage de la base de données et l'évolutivité.

Cette approche stocker uniquement les informations des nœuds feuilles et maintient la relation parent-enfant en utilisant la colonne *Ances_Pos*. Cela ralentit l'exécution des requêtes à cause de l'utilisation des auto-jointures.

3. Technique d'étiquetage ORDPATH

Plusieurs techniques d'étiquetage des nœuds d'un arbre de données XML ont été proposées. En plus de son efficacité au niveau d'insertion et de compression, *ORDPATH* [87] est conceptuellement similaire à la technique d'étiquetage *Dewey* [25], et tant que la plupart des techniques d'étiquetage sont conçus pour les structures de données XML statiques, l'insertion dans un arbre de données document XML reste un grand défi.

ORDPATH offre efficacement la possibilité de l'insertion à n'importe quelle position dans un arbre de données XML, et support également l'exécution de requêtes XPath avec efficience [87].

Les modifications de la structure d'une arborescence XML peuvent être effectuées de plusieurs manières :

a) De nouveaux sous-arbres peuvent être insérés.
b) Des sous-arbres peuvent être supprimés.
c) Les sous-arbres peuvent être déplacés dans l'arbre.

Au contraire, pour les systèmes d'étiquetage existants, la simple insertion d'un nœud peut être très coûteuse au niveau de temps d'exécution car elle nécessite le réétiquetage des nœuds existants.

ORDPATH matérialise la relation parent-enfant par l'utilisation d'un attribut ID. Le principe de cette approche consiste à affecter à l'étiquette du nœud enfant l'étiquette du nœud parent ORDPATH concaténée avec l'ID local de l'enfant (par exemple, 1.3 est l'ID du nœud parent de 1.3.5).

Dans un premier temps, seuls les nombres impairs et positifs sont attribués. Les autres entiers (pairs et négatifs) sont réservés pour une insertion ultérieure.

Pour insérer un nouveau nœud X entre deux nœuds existants, O'Neil *et al.* [87] ont proposé l'utilisation des nombres pairs ; par exemple, pour insérer un nouveau nœud entre deux nœuds : 1.3.5 et 1.3.7, le nouvel ID est 1.3.6.

4. Modèle de données XML

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<menu>
  <food>
    <name>Belgian Waffles</name>
    <price currency="$">5.95</price>
    <description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
    <calories fat="7.7" total="310"/>
    <vitamins>
      <a>7</a>
      <c>0</c>
    </vitamins>
  </food>
  <food>
    <name>Strawberry Belgian Waffles</name>
    <price currency="$">7.95</price>
    <description>Light Belgian waffles covered with strawberries and whipped cream</description>
    <calories fat="18" total="700"/>
  </food>
</menu>
```

Figure 10: Document XML

Dans ce chapitre, nous allons utiliser le modèle de données XPath [93] pour représenter notre document XML. Le modèle de données XPath modélise les documents XML comme un arbre ordonné en utilisant 7 types de nœuds, à savoir : racine, élément, texte, attribut, espace de noms(*namespace*), instruction de traitement et commentaire. Cependant, par souci de simplicité, nous considérons dans cette contribution uniquement les quatre types de nœuds suivants. : racine, élément, attribut et texte. La Figure 10 représente le document XML sur lequel nous allons travailler dans ce chapitre.

La Figure 11 illustre l'arbre de données XML correspondant à notre document XML de la Figure 10 basé sur le système d'étiquetage *ORDPATH*. Dans cette figure, nous représentons le nœud racine par un cercle noir, les nœuds de type élément par un cercle jaune, les nœuds de type attribut par un triangle jaune et finalement, nous représentons les nœuds de type texte par un rectangle blanc.

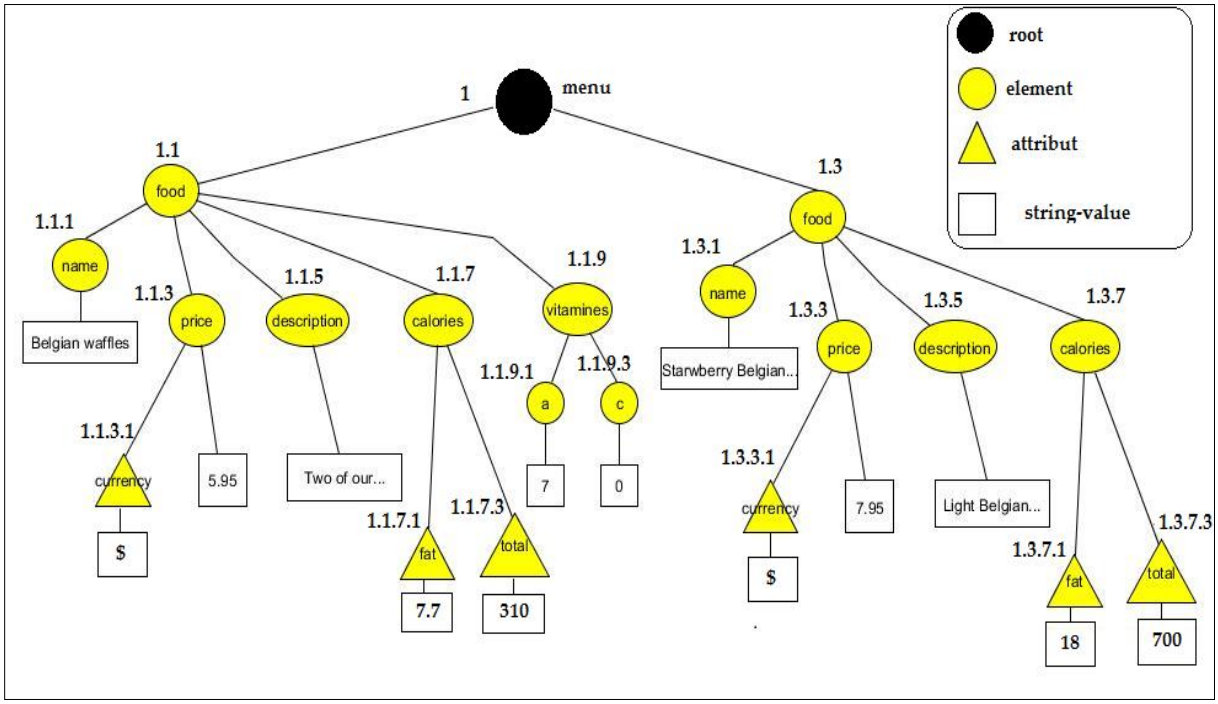


Figure 11: Arbre de données XML de notre document XML dans la Figure 10

5. Présentation du schéma relationnel proposé

Dans cette section, nous allons décrire le schéma de base de données relationnelle proposé pour stocker les documents XML. Ce schéma relationnel est composé de tables suivantes :

- **Vertex :** cette table est utilisée pour stocker le nom de tous les nœuds dans le document XML. Elle a la structure suivante :

  Vertex (id, nom) sachant que :

  - **id :** est un identifiant unique qui permet de référencer le nom d'un nœud.
  - **name :** conserve le nom d'un nœud.

- **Path :** le rôle cette table est de stocker tous les expressions de chemin distinctes existantes dans le document XML. Sa structure est comme suit :

  Path (id, path) sachant que :

  - **id :** enregistre l'ID du chemin.
  - **name :** stocke l'expression de chemin d'accès construite par la concaténation des ID de tous les nœuds constituants ce chemin (de la racine jusqu'à le dernier nœud). Nous avons choisi d'utiliser l'ID (de type numérique) au lieu d'utiliser les noms (de type chaîne de caractères) pour réduire l'espace de stockage. Ce chemin est utilisé

lors de la phase *Query Mapping*. C'est l'équivalent des chemins utilisés dans les requêtes XPath.

Data

ordpath	value	order	subElementsNbr	attributesNbr	pathId
1		1	2	0	1
1.1		2	5	0	2
1.1.1	Belgian Waffles	3	0	0	3
1.1.3	5.95	4	0	1	4
1.1.3.1	$	5	0	0	5
1.1.5	Two of our famous...	6	0	0	6
1.1.7		7	0	2	7
1.1.7.1	7.7	8	0	0	8
1.1.7.3	310	9	0	0	9
1.1.9		10	2	0	10
1.1.9.1	7	11	0	0	11
1.1.9.3	0	12	0	0	12
1.3		13	4	0	2
1.3.1	Strawberry Belgian	14	0	0	3
1.3.3	7.95	15	0	1	4
1.3.3.1	$	16	0	0	5
1.3.5	Light Belgian waffle...	17	0	0	6
1.3.7		18	0	2	7
1.3.7.1	18	19	0	0	8
1.3.7.3	700	20	0	0	9

Path

id	path
1	1
2	1.2
3	1.2.3
4	1.2.4
5	1.2.4.5
6	1.2.6
7	1.2.7
8	1.2.7.8
9	1.2.7.9
10	1.2.10
11	1.2.10.11
12	1.2.10.12

Vertex

id	name
1	menu
2	food
3	name
4	price
5	currency
6	description
7	calories
8	fat
9	total
10	vitamins
11	a
12	c

Figure 12: Le schéma relationnel généré à l'aide de l'approche XMap en se basant sur le document XML de la Figure 10

- **Data :** la table *Data* permet de stocker les informations détaillées sur tous les nœuds du document XML. Elle est composée des attributs suivants :

  Data (ordpath, value, order, subElementsNbr, attributesNbr, pathId**)** sachant que**:**

  - **ordpath :** Stocke l'ID du nœud. C'est une étiquette donnée pour chaque nœud en utilisant la technique *Ordpath*. Cet attribut stocke à la fois l'ID du nœud et de son parent. Par exemple, 1.1.9.1 est l'ID d'un nœud, et 1.1.9 est l'ID de son parent. L'objectif est de préserver la relation parent-enfant et permettre une éventuelle insertion ou suppression des nœuds sans un réétiquetage des nœuds de l'arbre de données XML.
  - **value :** enregistre la valeur textuelle du nœud (si elle existe).
  - **order :** préserve l'ordre du nœud parmi tous les nœuds du document.

- **subElementsNbr:** stocke le nombre d'éléments enfant d'un nœud. Il est utilisé dans l'étape de reconstruction d'un document XML.
- **attributesNbr :** stocke le nombre d'attributs d'un nœud. Cet attribut est utilisé dans l'étape de reconstruction d'un document XML.
- **pathId :** c'est une clé étrangère de la table *Path*. Cet attribut est utilisé lors de la phase de "*Query mapping*" pour faciliter la recherche des nœuds ayant une expression de chemin d'accès XPath donnée.

La Figure 12 représente le schéma relationnel généré à l'aide de notre approche XMap en se basant sur le document XML de la Figure 10.

6. Implémentation

Dans cette section, nous allons aborder l'architecture fonctionnelle de notre approche XMap en présentant les algorithmes et les fonctions utilisées dans les différentes étapes de migration.

Pour développer notre approche XMap, nous avons choisi d'utiliser Java comme langage de programmation grâce à sa puissance, sa stabilité, et son excellente portabilité. Nous avons opté pour MySQL comme système de gestion de base de données relationnelle pour sa rapidité, et aussi pour sa portabilité.

Il existe plusieurs parseurs pour analyser syntaxiquement un document XML comme DOM [94], SAX [95], StAX [96], et VTD [97]. Les plus répandus sont SAX et DOM. L'analyseur SAX est une interface standard orientées événement pour l'analyse des documents XML. Il analyse un document XML comme une séquence d'événements. L'analyseur DOM est un parseur orienté hiérarchie, il opère sur un document XML et construit ensuite une structure arborescente (arbre DOM) dans la mémoire pour analyser le document XML.

Nous avons choisi d'utiliser le DOM comme parseur parce qu'il représente plusieurs avantages par rapport au SAX, à titre d'exemples, nous citons :

- Il fournit un modèle d'objet pouvant modéliser tout document XML quelle que soit sa structure, ce qui facilite l'accès au contenu des documents XML.
- Il offre un accès total et aléatoire aux données XML [98] contrairement à un parseur SAX qui traite les données d'une manière séquentielle et qui offre des fonctions limitées [98].

Dans ce qui suit, nous présentons le modèle proposé pour les processus de mapping de XMap, l'algorithme de mapping de XML vers le modèle relationnel et finalement l'algorithme de reconstruction du document XML à partir d'une base de données relationnelle.

8.1. Modèle de processus de l'approche XMap

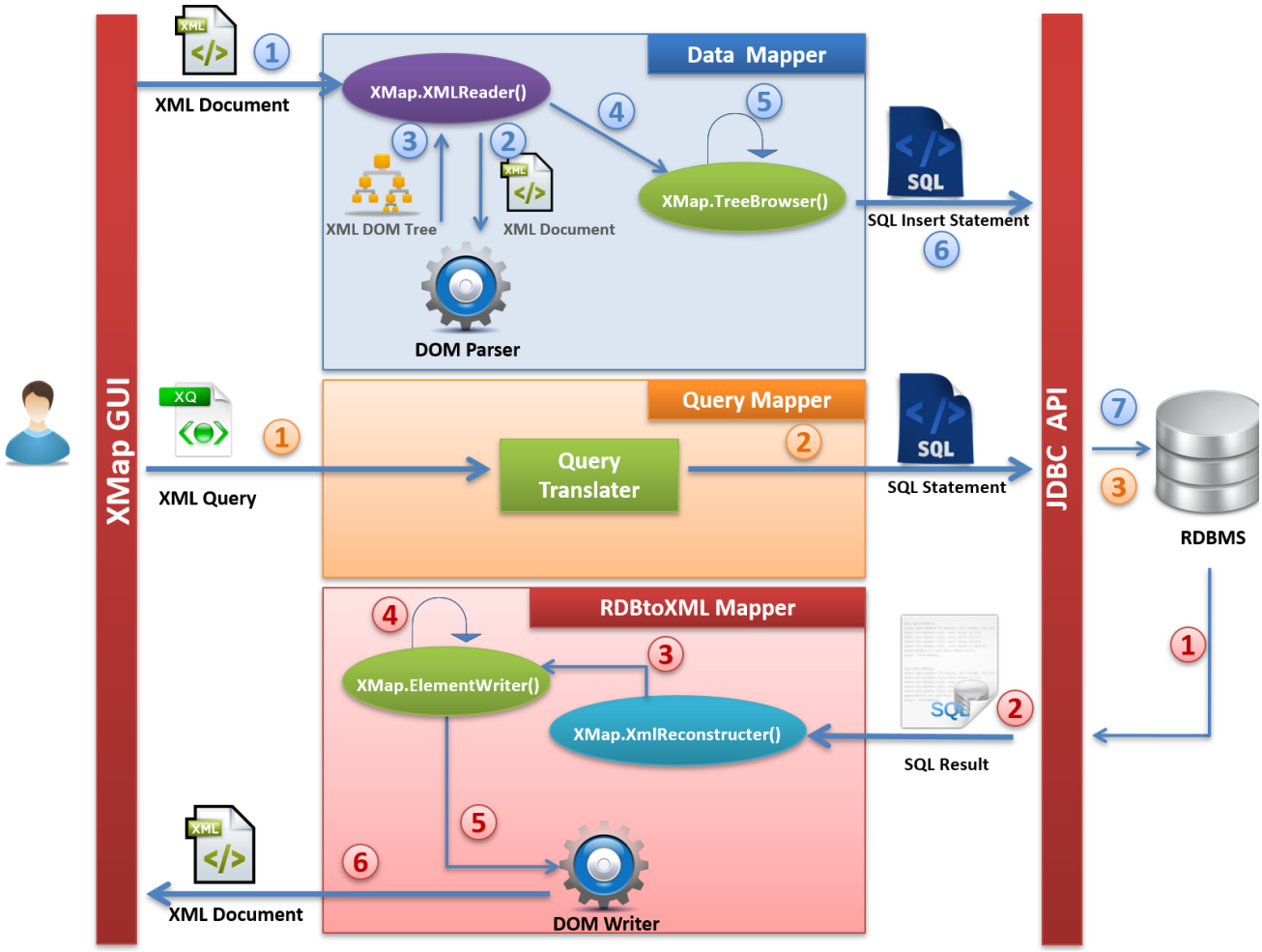


Figure 13: Diagramme de processus de l'approche XMap

L'approche XMap comprend trois composants principaux : *Data mapper*, *Query mapper* et *SQLtoXML mapper* comme il est illustré dans la Figure 13;

- *Data mapper* : Ce composant permet de charger le document XML et de l'analyser à l'aide de parseur DOM. Ensuite, il annote chaque nœud de l'arbre de données XML par une étiquette en utilisant la technique *Ordapath*. Finalement il déchiquette et décompose le contenu et aussi la structure hiérarchique du document XML dans un schéma relationnel prédéfini contenant les tables *Data*, *Vertex* et *Path*.

- *Query mapper* : Le rôle de ce composant est de traduire les requêtes XPath en requêtes SQL. D'ailleurs, les requêtes XPath sont saisies par l'utilisateur et traduites ensuite en requêtes SQL équivalentes en utilisant un algorithme de mapping de requêtes.
- *SQLtoXML mapper* : Les instructions SQL générées par le composant *Query mapper* vont être envoyées au moteur de base de données relationnelle afin de récupérer les résultats des requêtes. Ce composant permet de reconstruire ces résultats sous forme de données XML et de les renvoyer à l'utilisateur.

8.2. Algorithme XMLtoRDB

L'algorithme *XMLtoRDB* se focalise sur le mapping des documents XML vers une base de données relationnelle en utilisant une technique basée sur les chemins. Il prend en entrée un document XML. En effet, il le déchiquette et décompose la structure hiérarchique qui est représentée dans ce document XML en une structure plate qui est représentée par notre schéma relationnel composé par trois tables ; *Path*, *Vertex* et *Data*. L'idée de base de cet algorithme est de stocker les chemins des nœuds une seule fois dans la table *Path* et utiliser des chemins composés par des nombres au lieu des chaines des caractères (ils seront convertis par la suite par une fonction SQL dans la phase de *Query Mapping*). Cela entraîne une réduction de l'espace de stockage. Cet algorithme est subdivisé en trois fonctions principales *XMLParser*, *getOrdPathId et treeBrowser*.

8.2.1. Fonction XMLParser

La Figure 14 représente la fonction principale (main) de l'algorithme XMLtoRDB qui permet de faire le mapping d'un document XML vers une base de données relationnelle.

Tout d'abord, la fonction charge le fichier XML en mémoire en utilisant le parseur DOM [ligne 3] pour extraire l'élément racine [ligne 4], puis elle initialise les propriétés *ordPath*, *idPath* et *order* de la racine par 1 [lignes 6-8]. Ensuite elle ajoute le nom et l'identifiant du chemin de cet élément dans le vecteur des nœuds(*stackVertex*) [ligne 9] et le vecteur des chemins(*stackPath*) [ligne 10] respectivement. Après elle fait appel à la fonction *treeBrowser* qui prend l'élément racine en paramètre [ligne 11]. L'objectif de la fonction *treeBrowser* est de parcourir récursivement la liste de tous les éléments descendants de l'élément racine. Dans les lignes 12 et 13, les fonctions *insertPath* et *insertVertex* vont insérer dans les tables *Path* et *Vertex* le contenu des vecteur *stackPath* et *stackVertex* respectivement. Ceux-ci seront remplies au cours d'exécution de la fonction *treeBrowser*.

Algorithm 1: XMLtoRDB Mapping

Input: XML Document.
Output: XML data stored in relational database.

```
1  /* Main function                                                    */
2  Function XMLParser():
3      Document doc ← domParser.loadDocument(XMLDocument)
4      Element root ← doc.getRootElement()
5      globale order
6      root.ordPath ← '1'
7      root.idPath ← '1'
8      root.order ← '1'
9      stackVertex.add(root.NodeName)
10     stackPath.add('1')
11     treeBrowser(root)
12     insertPath()                    // to store infomation of distinct paths in RDB
13     insertVertex()                  // to insert infomation of distinct vertex in RDB
14 End Function
```

Figure 14: La fonction XMLParser

La Figure 15 représente un organigrame permettant d'illustrer la logique de la fonction XMLParser

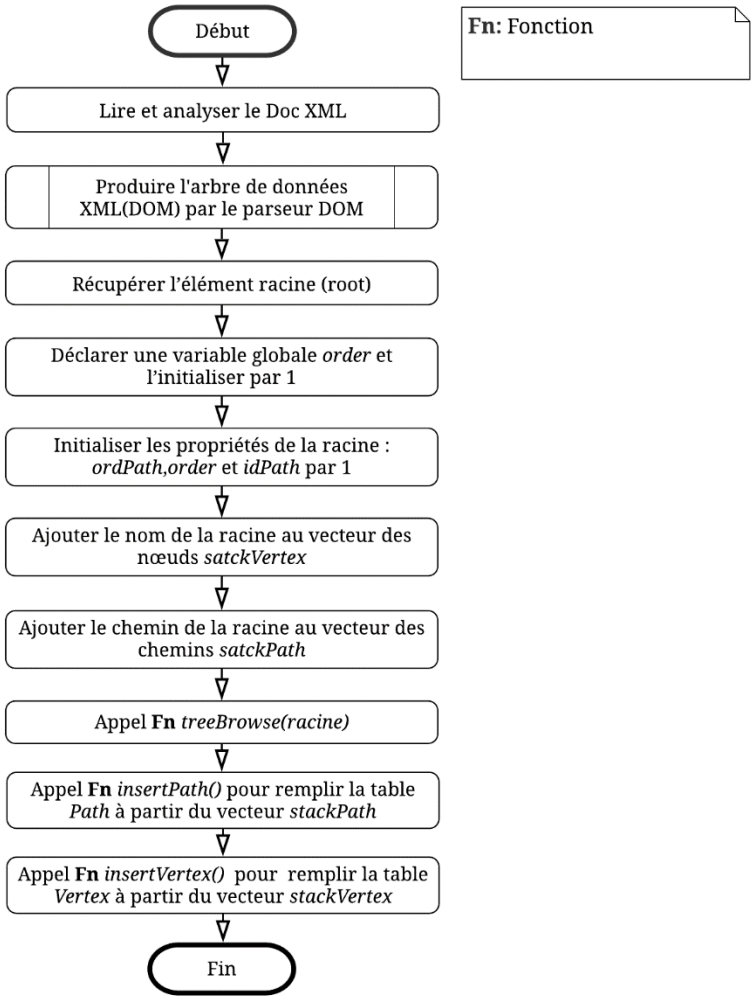


Figure 15: Organigramme de la fonction XMLParser

8.2.2. Fonction getOrdPathId

Algorithm 1: XMLtoRDB Mapping

Input: XML Document.
Output: XML data stored in relational database.

```
1 Function getOrdPathId(String parentOrdPath, int previousLabledSiblingsNum):
2 |   ordPathId ← concat(parentOrdPathId, '.', ((previousLabledSiblingsNum * 2) + 1))   // in initial
  |    load we use only positive odd (ordPath technique)
3 |   return ordPathId
4 End Function
```

Figure 16: Fonction getOrdPathId

Cette fonction implémente le concept de la technique *ORDPATH* [87] permettant l'étiquetage des nœuds d'un arbre des données XML. En effet, le principe de cette technique repose essentiellement sur l'utilisation des nombres positifs et impaires lors de l'étiquetage initiale des nœuds dans la phase du *Data Mapping* tout en réservant les nombres négatifs et paires pour une éventuelle insertion. La fonction *getOrdPathId* prend en entrée deux paramètres à savoir l'*ordPath* de parent (*parentOrdPathId*) et le nombre de nœuds précédents de l'élément courant qui sont déjà étiquetés (*previousLabledSiblingsNum*). Elle retourne à la fin un identifiant *ordPath*. La Figure 16 présente le pseudo-code de la fonction *getOrdPathId.*

L'identifiant *ordPath* est attribué en concaténant l'*ordPath* de parent, le séparateur "." et l'étiquette local du nœud courant.

Il convient de noter que l'étiquette local du nœud courant est calculée par la formule suivante :

$$etiquetteLocaleNœud = (previousLabledSiblingsNum * 2) + 1 \quad (1)$$

Nous allons donner dans ce qui suit, deux exemples typiques pour faire concevoir une idée plus claire à propos de la logique de la fonction en question.

- **Étiquetage d'un nœud précédé par des nœuds frères**

La figure 13 montre un sous-arbre composé par un nœud parent A et des nœuds enfants B, C et D.

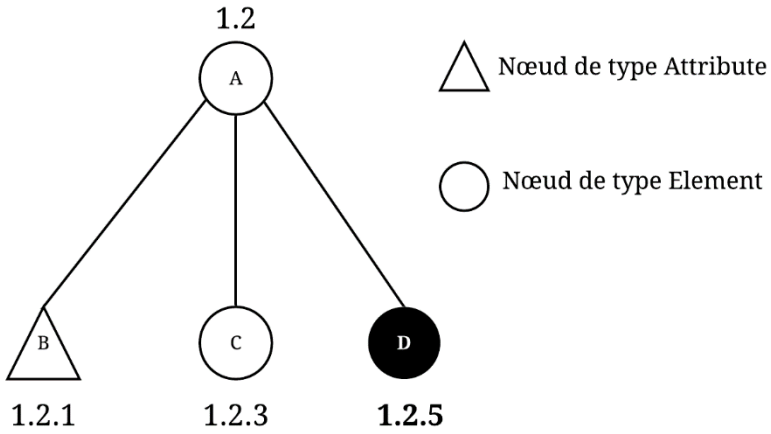


Figure 17 : Étiquetage d'un nœud précédé par des nœud frères

La fonction *getOrdPathId* attribue une étiquette au nœud *D* ayant des nœuds frères précédents (*B* et *C*) en concaténant l'*ordPath* du parent ("1.2"), le séparateur "." et l'étiquette local du nœud *D*. Celle-ci est calculée en utilisant la formule (1), sachant que :

$$previousLabledSiblingsNum = 2$$

Puisque le nœud *D* a deux nœuds frères précédents. On obtient alors :

$$etiquetteLocaleNœud = (2 * 2) + 1$$

Ce qui nous donne :

$$etiquetteLocaleNœud = 5$$

Par conséquent l'*ordPath* du nœud *D* est "1.2.5".

- **Étiquetage d'un nœud non précédé par des nœuds frères**

Nous allons supposer dans cet exemple que le nœud *B* ne possède pas des nœuds frères précédents comme il est illustré dans la Figure 18.

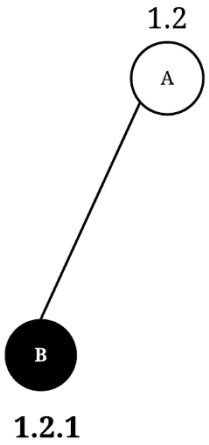


Figure 18 : Étiquetage d'un nœud non précédé par des nœud frères

Dans ce cas on obtient :

$$etiquetteLocaleNœud = 0 + 1$$

Ce qui nous donne "1.2.1" comme *ordPath* de ce nœud.

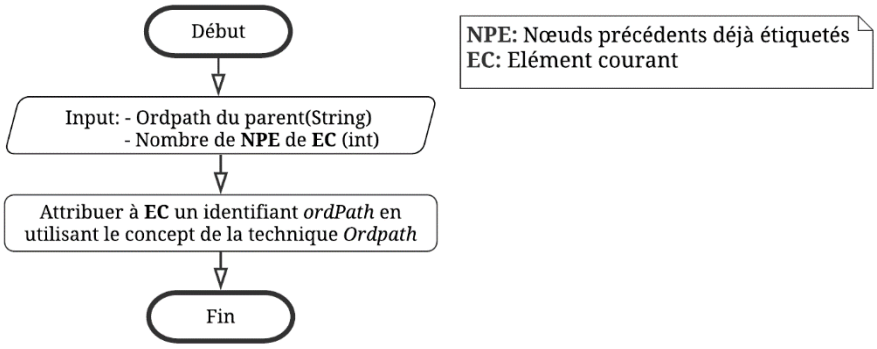


Figure 19: Organigramme de la fonction getOrdPathId

La Figure 19 illustre une présentation graphique de la logique de la fonction *getOrdPathId* sous forme d'un organigramme.

8.2.3. Fonction treeBrowser

```
Algorithm 1: XMLtoRDB Mapping
   Input: XML Document.
   Output: XML data stored in relational database.
 1 Function treeBrowser(Element elm):
 2     listChildNodes ← elm.childNodes
 3     nbrElm ← 0
 4     foreach node ∈ listChildNodes do
 5         if node.type = ELEMENT then
 6             nbrElm++
 7             order++
 8             path ← concat(node.parentPath, '.', node.id)
 9             previousLabledSiblingsNum ← node.parentAttrNum + node.previousElemSiblingNum
10             node.ordPath ← getOrdPathId(node.parentOrdPath, previousLabledSiblingsNum)
11             node.order ← order
12             node.idPath ← stackPath.indexOf(path) // we assume that the stackPath index start at 1
13             if node.name ∉ stackVertex then
14                 stackVertex.add(node.name)
15             end if
16             if path ∉ stackPath then
17                 stackPath.add(path)
18             end if
19             if | node.attributes | > 0 then
20                 listAttributes ← node.attributes
21                 i ← 0
22                 foreach attribute ∈ listAttributes do
23                     order++
24                     path ← concat(attribute.parentPath, '.', attribute.id)
25                     attribute.ordPath ← getOrdPathId(attribute.parentOrdPath, i)
26                     attribute.idPath ← stackPath.indexOf(path)
27                     if attribute.name ∉ stackVertex then
28                         stackVertex.add(attribute.name)
29                     end if
30                     if path ∉ stackPath then
31                         stackPath.add(path)
32                     end if
33                     i++
34                     Store attribute.ordPath, attribute.value, order, 0, 0, attribute.idPath in DATA Table
35                 end foreach
36             end if
37             treeBrowser(elm)
38         end if
39     end foreach
40     Store elm.ordPath, elm.value, elm.order, nbrElm, elm.attributesLength, element.idPath in DATA
       Table
41 End Function
```

Figure 20: Fonction treeBrowser

La fonction *treeBrowser* est la pierre angulaire de l'algorithme XMLtoRDB. C'est une fonction récursive qui opère sur un paramètre de type *Element*. Elle permet de parcourir la liste de tous les éléments descendants de l'élément passé en paramètre. Le but principal cette fonction est d'étiqueter les nœuds de l'arbre de données XML (arbre DOM) en utilisant la technique *OrdPath*, remplir les vecteurs *stackPath* et *stackVertex*, et insérer les informations

détaillées concernant les nœuds dans la table *Data*. La Figure 20 présente le pseudo-code de la fonction *treeBrowser.*

Premièrement, la fonction *treeBrowser* récupère la liste des sous-nœuds de l'élément passé en paramètre (ligne 2) et elle la parcourt par la suite (ligne 4). Si le nœud est de type *Element ;* les variables *nbrElm* (nombre de sous-éléments) et *order* sont incrémentées (lignes 6-7), et le chemin (path) est attribué en concaténant le chemin de parent, le séparateur "." et l'identifiant du nœud (ligne 8). La variable *previousLabledSiblingsNum* présente le nombre de nœuds précédents de l'élément courant qui sont déjà étiquetés. D'ailleurs, cette variable est la somme de nombre d'attributs de parent et de nombre de nœuds frères précédents ayant comme type *Element* (ligne 9). La prise en compte des attributs du parent est primordiale, puisqu'ils sont les premiers qui s'étiquettent.

Dans les lignes 10, 11 et 12 les propriétés suivantes sont attribuées au nœud :

- *ordPath* est généré par la fonction *getOrdPathId*.
- *ordre*.
- *idPath* est l'indice du chemin (*path*) dans le vecteur *stackPath.*

Par la suite, la fonction *treeBrowser* vérifie si le nom du nœud et le chemin (*path*) existent respectivement dans les vecteurs *stackVertex* et *stackPath*, sinon elle les ajoute (lignes 13-18).

Si l'élément a des nœuds enfants de type *Attribute*, la fonction les stocke dans la liste *listAttributes* (lignes 19-20). Pour chaque nœud enfant de la liste *listAttributes* la fonction exécute les instructions ci-dessous (lignes 22-35) :

- Incrémenter la variable *order*.
- Attribuer le chemin (*path*) par la concaténation de trois éléments à savoir le chemin de parent, le séparateur "." et l'identifiant de l'attribut.
- Créer l'étiquette *Ordpath* en utilisant la fonction *getOrdPathId*.
- Ajouter respectivement le nom de l'attribut et le chemin (*path*) dans les vecteurs *stackVertex* et *stackPath* au cas où ils n'existent pas.
- Insérer dans la table *Data* les informations de l'attribut à savoir *ordPath*, la valeur textuelle, *order*, le nombre de sous-éléments, le nombre d'attributs et l'identifiant du chemin de l'attribut (*idPath*). Le nombre de sous-éléments et le

nombre d'attributs sont toujours égale à zéro dans le cas des nœuds de type *Attribute*. L'identifiant du chemin (*idPath*) est une clé étrangère de la table *Path*.

Après avoir traité un nœud de type *Element* et ses attributs (lignes 6-36), la fonction *treeBrowser* fait appel à soi-même (ligne 37) pour traiter les nœuds descendants de l'élément passé en paramètre. Lorsque la fonction accomplit le traitement des nœuds descendants d'un élément, elle stocke ses informations dans la table *Data* (ligne 40).

Après l'exécution de l'algorithme *XMLtoRDB*, le document XML présenté dans la Figure 10 est décomposé et stocké dans la base de données relationnelle comme le montre la Figure 12.

La Figure 21 montre l'organigramme de la fonction *treeBrowser*.

8.3. Algorithme RDBtoXML

L'algorithme *RDBtoXML* permet d'assurer la reconstruction d'un document XML à partir d'une base de données relationnelle. Ceci en traduisant le jeu de résultat SQL(*ResultSet*) renvoyé par le composant *Query Mapper* en format de données XML. Le processus de reconstruction d'un document XML à partir d'une base de données relationnelle est utilisé pour:

- S'assurer que la méthode utilisée dans la phase de mapping, maintient efficacement l'ensemble du document XML sans perte d'informations.
- Reconstruire l'ensemble du document XML.
- Reconstruire une partie ou des parties d'un document XML à la suite d'une requête XPath ou XQuery exécutée par l'utilisateur et traduite en instructions SQL par le composant *Query Mapper*.

Cet algorithme opère sur un jeu de résultat SQL renvoyé par le composant *Query Mapper* et renvoie à la fin des données en format XML. Le traitement de cet algorithme est divisé en deux fonctions principales *XMLReconstracter* et *elementWriter*.

La Figure 22 et Figure 23 représentent respectivement le pseudo-code et le logigramme de l'algorithme *RDBtoXML*.

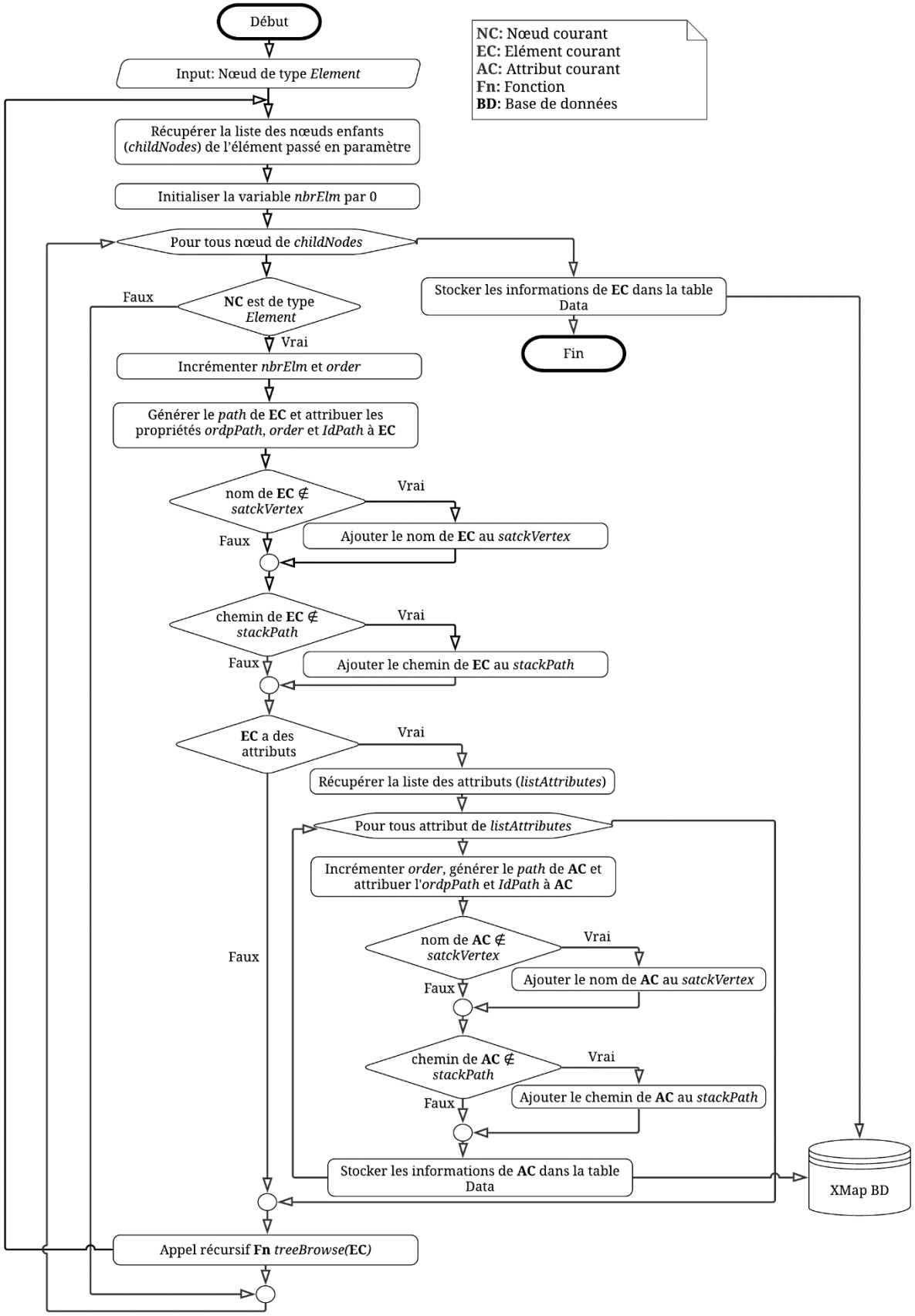


Figure 21: Organigramme de la fonction treeBrowser

Algorithm 2: RDBtoXML Mapping

```
   Input: SQL ResultSet of size n
   Output: XML Document
 1 /* Main function                                                    */
 2 Function XMLReconstracter():
 3 |   global int globalCursor ← 0
 4 |   global document doc
 5 |   if n > 0 then
 6 |   |   elementWriter(null,resultSet[0][subElementsNbr])
 7 |   end if
 8 End Function

 9 Function elementWriter(Element parent, int vSubElementsNbr):
10 |   int i ← 0
11 |   while i < vSubElementsNbr do
12 |   |   Element elem ← doc.createElement(resultSet[globalCursor][name])
13 |   |   elem.createTextValue(resultSet[globalCursor][value])
14 |   |   int locSubElemNbr ← resultSet[globalCursor][subElementsNbr]
15 |   |   int locAttrNbr ← resultSet[globalCursor][attributesNbr]
16 |   |   globalCursor++
17 |   |   int lastAttr ← globalCursor + locAttrNbr
18 |   |   for j ← globalCursor to lastAttr do
19 |   |   |   elem.setAttribute(resultSet[j][name], resultSet[j][value])
20 |   |   |   globalCursor++
21 |   |   end for
22 |   |   if parent ≠ null then
23 |   |   |   parent.appendChild(elem)
24 |   |   else
25 |   |   |   doc.appendChild(elem)
26 |   |   |   i++
27 |   |   end if
28 |   |   if locSubElemNbr > 0 then
29 |   |   |   elementWriter(elem, locSubElemNbr)
30 |   |   end if
31 |   |   i++
32 |   end while
33 End Function
```

Figure 22: Algorithme RDBtoXML

Afin d'exporter le document tout entier, il faut remplir le tableau de données *resultSet* utilisé dans l'algorithme *RDBtoXML* en utilisant la requête de la Figure 27.

8.3.1. Fonction XMLReconstracter

C'est la fonction "main" de l'algorithme *XMLReconstracter*. En premier lieu, elle crée deux variables globales ; *globalCursor* de type entier et *doc* de type *Document* (lignes 3 et 4) et vérifie ensuite si le jeu de résultat SQL n'est pas vide (lignes 5-7). Si c'est le cas, elle fait appel à la fonction *elementWriter* et passe en paramètre les informations de l'élément racine (*root*) (ligne 6).

8.3.2. Fonction elementWriter

La fonction *elementWriter* est la clef de voûte de l'algorithme *RDBtoXML*. Elle opère sur deux paramètres ; un élément parent et le nombre de ses éléments enfants, et elle renvoie à la fin un fichier XML. C'est une fonction récursive permettant d'associer les attributs et les éléments enfant à leurs éléments parents. Cette fonction utilise la variable globale *globalCursor* pour incrémenter la position du pointeur interne du tableau de données afin de parcourir séquentiellement une seule fois le jeu de résultat SQL.

Premièrement, la fonction crée un objet de type *Element* à partir des informations stockées dans la ligne sur laquelle le pointeur interne du tableau est positionné et elle récupère les informations de cet élément (lignes 12-15) puis elle incrémente la position de pointeur interne du tableau de données (lignes 16). Ensuite elle associe l'élément créé avec ses attributs (lignes 18-21). Si le parent (l'élément passé en paramètre) est nul cela signifie que l'élément courant est un élément racine(*root*) alors la fonction *elementWriter* va l'associer au variable *doc* (lignes 24-27), sinon elle va associer l'élément courant à son parent (lignes 22-24). Finalement la fonction fait appel à soi-même si l'élément courant a des éléments enfants (lignes 28-30). L'objectif de ceci est de parcourir récursivement le tableau de données et associer chaque élément avec ses attributs et ses éléments enfants et renvoie à la fin le fichier XML résultant.

8.3.3. Complexité

Le tableau de données *resultSet* utilisé dans l'algorithme *RDBtoXML* est remplit par les informations des nœuds de l'arbre de données d'un document XML donné. Sachant que :

$$n = NE + NA$$

Où *n* est le nombre des nœuds de l'arbre de données XML, *NE* est le nombre des nœuds de type *Element* et *NA* est le nombre des nœuds de type *Attribut*.

L'algorithme *RDBtoXML* parcourt séquentiellement une seule fois le tableau de données afin de reconstruire le document XML. La boucle *while* (ligne 11) est exécutée *NE* et la boucle *for* (ligne 18) est exécutée *NA* fois. Alors la complexité de l'algorithme *RDBtoXML* est linéaire *O*(*n*), sachant que *n* est le nombre des nœuds de l'arbre de données XML.

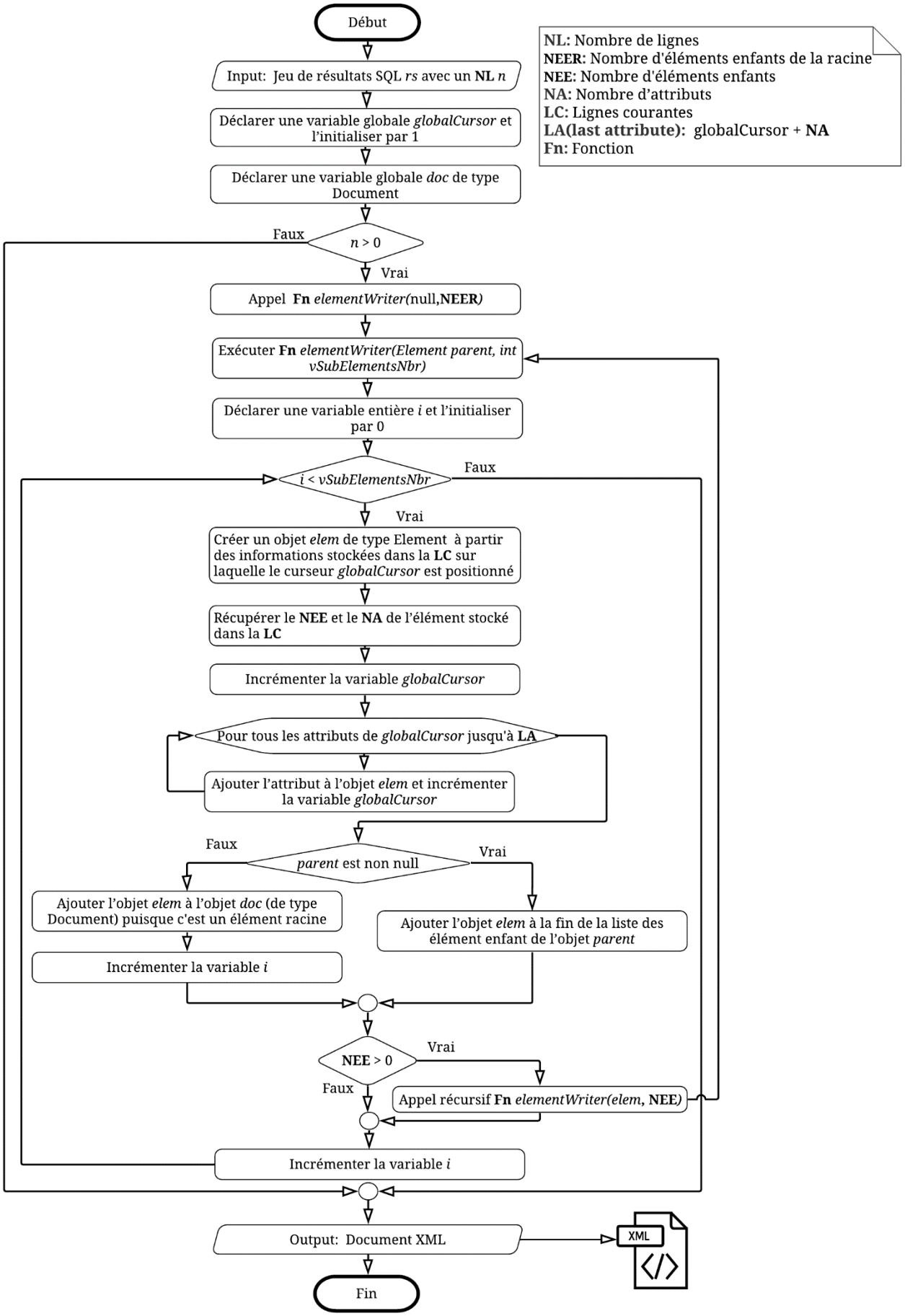


Figure 23: Logigramme de l'algorithme RDBtoXML

8.4. Mapping des requêtes XPath en requêtes SQL

Le schéma de base de données relationnelle proposé pour notre approche est modélisé pour bénéficier à la fois des avantages des chemins pour accélérer la recherche en utilisant les requêtes XPath et aussi pour minimiser le stockage de ces chemins. Pour cela, nous avons choisi de stocker le chemin comme illustré dans la Figure 12 (table *Path*), et nous avons implémenté des fonctions MySQL pour convertir le chemin utilisé dans les requêtes XPath en format de chemins utilisés dans notre table *Path*.

8.4.1. Fonction getPath()

Cette fonction (cf. Figure 24) opère sur un chemin en mode texte (chemin XPath) et renvoie à la fin un chemin converti au format utilisé dans notre approche.

```
DELIMITER $$
CREATE function getPath(parmPath TEXT)
RETURNS VARCHAR(160)
READS SQL DATA
BEGIN
  DECLARE varId VARCHAR(10);
  DECLARE varIds VARCHAR(160);
  DECLARE varValue VARCHAR(60);
  DECLARE instrValue INT;
  SET varIds = '';
  SET varId  = '';

  REPEAT
    SET instrValue = INSTR(parmPath,'/');
    IF instrValue <> 0 THEN
      SET varValue = SUBSTRING(parmPath, 1, instrValue-1);
      SET parmPath = SUBSTRING(parmPath, instrValue+1);
      SET varId = concat( getVertexId(varValue), '.');
    ELSE
      SET varValue = parmPath;
      SET varId = getVertexId(varValue);
    END IF;
    SET varIds = CONCAT(varIds, varId);
  UNTIL instrValue = 0
  END REPEAT;
  RETURN varIds;
END
```

Figure 24: Fonction *getPath()*

- **Exemple d'utilisation:**

```
SELECT getPath("/menu/food/price/currency");
```

- **Resultat:**

```
1.2.4.5
```

8.4.2. Fonction getVertexId()

Le rôle de cette fonction (cf. Figure 25) est d'extraire l'ID de chaque élément d'un chemin à partir de la base de données. Elle est utilisée par la fonction *getPath()*.

```
DELIMITER $$
CREATE function getVertexId(varValue TEXT)
RETURNS VARCHAR(10) READS SQL DATA
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE varId VARCHAR(10);
  DECLARE getId CURSOR FOR
    SELECT id
    FROM vertex
    WHERE name = varValue;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done =
RUE;
  SET varId = '';
  OPEN getId;
  REPEAT
    FETCH getId INTO varId;
  UNTIL done = TRUE
  END REPEAT;
  CLOSE getId;
  RETURN varId;
```

Figure 25: Fonction *getVertexId()*

8.4.3. Fonction getParent()

La fonction *getParent*() (cf. Figure 26) retourne l'identifiant du parent d'un identifiant *ordPath*.

```
DELIMITER $$
 CREATE function getParent(v TEXT)
 RETURNS TEXT READS SQL DATA
 BEGIN
   DECLARE theParent TEXT;
   SET theParent = (SELECT LEFT(v, LENGTH(v) - LOCATE('.', REVERSE(v))));
   RETURN theParent;
 END
```

Figure 26: Fonction *getParent()*

- **Exemple d'utilisation:**

```
SELECT getParent("1.5.1");
```

- **Resultat:**

```
1.5
```

8.4.4. Traduction des requêtes XPath en requêtes SQL

Dans cette partie, nous mettons l'accent sur quelques exemples de la traduction des requêtes XPath en requêtes SQL. Le Tableau 2 présente quelques exemples des requêtes que nous allons traiter.

Tableau 2 : Requêtes XPath

Numéro de La requête	Requêtes XPath
R1	/menu
R2	/menu/food[price>4]/name
R3	/menu/food[calories/@total =700]/price

- **R1:** /menu

Cette requête sélectionne tous les nœuds du document XML. La requête SQL correspondante à cette requête XPath est :

```
SELECT ordpath, name, value, subElementsNbr, attributesNbr
FROM data,path,vertex
WHERE data.pathId=path.id
AND vertex.id= cast(SUBSTRING_INDEX(path.path,  '.', -1) AS UNSIGNED)
ORDER BY order ASC
```

Figure 27 : Requête SQL pour sélectionner tous les nœuds du document XML

- **R2:** /menu/food[price>4]/name

Cette requête sélectionne les noms (*name*) des aliments (*food*) dont le prix (*price*) est supérieur à 4. Afin que le résultat soit correct, nous devons vérifier au niveau de la requête SQL que la valeur de l'élément *price* est supérieur à 4 et que l'élément *name* et l'élément *price* ont le même parent. La Figure 28 représente la requête SQL correspondante à la requête XPath **R2**.

- **R3:** /menu/food[calories/@total=700]/price

Cette requête sélectionne le prix (*price*) des aliments (*food*) dont le total (*total*) des calories est égal à 700. Nous devons vérifier que l'attribut *total* de l'élément *calories* est égal à 700, et nous devons vérifier aussi que l'élément *price* et l'élément *calories* (le parent de

l'attribut *total*) ont le même parent. La Figure 29 représente la requête SQL correspondante à la requête XPath **R3**.

```
SET @pathName =getPath("/menu/food/name");
SET @pathPrice =getPath("/menu/food/price");
SELECT d2.value
FROM Data d1, Data d2, Path p1, Path p2
WHERE  p1.path = @pathPrice
AND p2.path =@pathName
AND cast(d1.Value AS DECIMAL(10,2)) > 4
AND d1.pathId = p1.id
AND d2.pathId = p2.id
AND getParent(d1.ordpath) = getParent(d2.ordpath)
```

Figure 28: Requête SQL correspondante à la requête XPath R2

```
SET @pathPrice =getPath("/menu/food/price");
SET @pathTotal =getPath("/menu/food/calories/total");

SELECT d1.Value
FROM Data d1, Data d2, Path p1, Path p2
WHERE p1.path = @pathPrice
AND p2.path =@pathTotal
AND cast(d2.Value AS DECIMAL(10,2)) = 700
AND d1.pathId = p1.id
AND d2.pathId = p2.id
AND getParent(d1.ordpath) = getParent(getParent(d2.ordpath))
```

Figure 29: Requête SQL correspondante à la requête XPath R3

7. Conclusion

Le XML est récemment devenu le principal support de stockage et de transfert de données sur le web grâce à sa structure adaptable et à sa souplesse de définition des balises. De nombreuses organisations ont adopté le XML comme la pierre angulaire de leurs applications en ligne. De plus, la simplicité, la stabilité et la maturité de la technologie des bases de données relationnelles existantes ajoutent grandement à l'extensibilité et à la robustesse du système de stockage XML. C'est ce qui a motivé beaucoup d'organisations de s'orienter vers le stockage et la gestion des données XML à l'aide de bases de données relationnelles. Par conséquent, une technique efficace de mapping est certainement nécessaire pour établir un pont entre le XML et les bases de données relationnelles. Dans cette optique, nous avons proposé une nouvelle approche pour stocker et interroger un document XML dans une base de données relationnelle. Dans cette approche, nous avons utilisé trois tables pour stocker toutes les informations relatives à un document XML. Notre approche utilise la technique d'étiquetage *OrdPath*, ce qui la permet de supporter efficacement les modifications

structurelles. Dans notre approche, nous avons focalisé sur quatre types de nœuds à savoir racine, élément, attribut et texte.

Chapitre III

L'Application de XMap dans le Domaine de la Conception des Systèmes d'Apprentissage Adaptatifs

1. Introduction

L'apprentissage électronique est considéré comme une option largement reconnue pour remédier aux inconvénients de l'environnement d'apprentissage traditionnel [99]. Les systèmes d'apprentissage adaptatifs sont conçus dans le but de permettre aux apprenants de choisir librement les objets d'apprentissage qu'ils souhaitent voir apparaître dans leurs cours. Ces objets d'apprentissage présentent plusieurs caractéristiques telles que : l'interopérabilité, la réutilisabilité, l'accessibilité, la durabilité et l'adaptabilité [100]. L'accessibilité, la réutilisation et l'interopérabilité peuvent être garanties en utilisant le langage XML . Ce dernier a été largement utilisé comme la norme de facto pour communiquer des informations entre les applications sur le Web en utilisant un balisage structurel et sémantique[101]. Le langage XML est considéré comme la pierre angulaire de la technologie des services Web, L'Intégration d'Applications d'Entreprise (IAE) et les Architectures Orientées Services (AOS), il est utilisé pour le stockage et l'échange des données. Ceci est réalisable vu que le XML est une technologie multiplateforme par nature et il est capable de combler les différences entre les systèmes et les appareils ce qui offre l'avantage de l'indépendance de la plate-forme et permet d'éviter les changements de format pour différents systèmes informatiques. La simplicité de la syntaxe de XML permet à l'homme et à la machine de comprendre facilement ce langage.

Il existe quatre approches pour stocker les données XML[9] : Les bases de données XML native (NXD), les bases de données relationnelles (RDB), les bases de données orientées objet (OODB) et les fichiers. Les bases de données XML native souffrent de plusieurs limitations comme la sécurité, les transactions et la gestion d'accès concurrents, etc. D'autre coté, le stockage des données XML dans fichiers présente des lacunes au niveau d'interrogation des données XML et les bases de données et les bases de données orientées objet souffrent également de nombreuses insuffisances telles que l'absence d'un langage de requête standard pour OODB. Pour de surmonter ces limitations, plusieurs approches ont été proposées [14], [18]–[26], pour tirer parti des technologies matures fournies par les systèmes de gestion de bases de données relationnels actuels à savoir les index, les déclencheurs, l'intégrité des données, la sécurité et l'optimisation des requêtes SQL, et aussi pour exploiter toute la puissance de cette technologie.

Dans ce chapitre, nous proposons une approche permettant d'aider les apprenants dans leur parcours d'apprentissage en leur fournissant des supports de cours adaptés à leurs

préférences. Il existe plusieurs technologies pour représenter un cours structuré. Dans notre approche, nous avons opté pour le langage XML. Afin de surmonter les limitations des bases de données XML et pour exploiter les avantages des bases de données relationnelles, le document XML du support de cours sera stocké dans une base de données relationnelle à l'aide de notre approche E-XMap et nous allons utiliser les techniques d'étiquetage XML pour identifier les relations entre les nœuds et accélérer le traitement des requêtes.

2. Concepts de base

2.1. Systèmes d'Apprentissage Adaptatifs

Les systèmes d'apprentissage adaptatifs consistent à offrir une expérience d'apprentissage efficace et personnalisée aux apprenants. Ceci en adaptant de manière dynamique le contenu d'apprentissage en fonction des caractéristiques spécifiques de chaque apprenant. Celles-ci représentent une combinaison des propriétés à savoir les connaissances, le niveau, les expériences, le style d'apprentissage, les compétences, l'objectif, les préférences, etc. En tant que modalité d'apprentissage éprouvée, les systèmes d'apprentissage adaptatifs sont utilisés dans de nombreux environnements différents pour enseigner et former plus efficacement les apprenants.

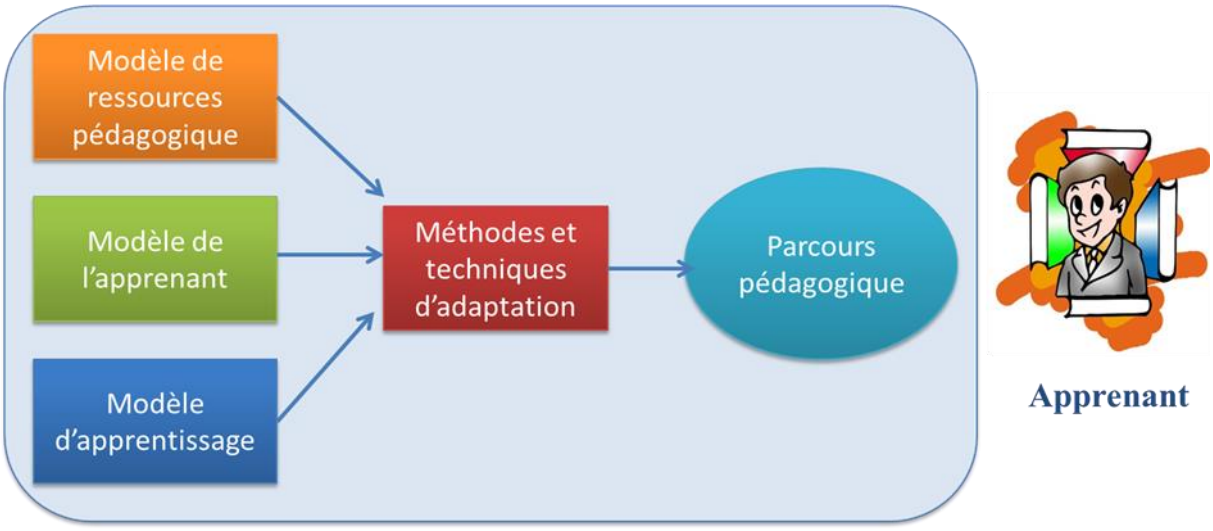


Figure 30: Composants des systèmes d'apprentissages adaptatifs [102]

Chaque système d'apprentissage adaptatif est constitué de quatre composants principaux comme le montre la Figure 30:

- Modèle des ressources pédagogiques : Représente l'ensemble des connaissances qui doivent être transmises à l'apprenant.

- Modèle de l'apprenant : Représente les informations sur les intérêts, les connaissances, les compétences, les préférences, le niveau et le style d'apprentissage de chaque apprenant. Ces informations sont utilisées pour la construction de parcours individualisés.
- Modèle d'apprentissage : C'est la partie du système qui permet de prendre les décisions sur les éléments de contenu (définitions, exemples, illustrations, exercices, etc.) qui seront présentés à l'apprenant afin qu'il puisse acquérir les connaissances contenues dans le modèle des ressources pédagogiques. Ce modèle permet de combler le fossé entre l'apprenant et la connaissance.
- Méthodes et techniques d'adaptation : Ce sont les méthodes et techniques utilisées pour générer des contenus adaptés aux préférences et caractéristiques de chaque apprenant.

Il existe aujourd'hui de nombreux systèmes d'apprentissage adaptatifs, issus de courants de recherche très différents. Nous citons par exemple METADYNE[103], HYPERGAP[104] et HYDAYA[105].

2.2. Les techniques d'étiquetage

2.2.1. Les techniques d'étiquetage XML : Une vue d'ensemble

Avec la popularité croissante du XML dans l'échange de données, de nombreuses approches ont été proposées pour faciliter le traitement des requêtes, d'obtenir une indexation efficace et de préserver les relations structurelles entre les nœuds (parent, enfant, frères, ancêtre et descendant) [106]. Ces approches peuvent être classées en deux types [107]–[109] :

- Les systèmes d'étiquetage statiques : ils traitent les documents XML statiques [25], [110]–[112].
- Les systèmes d'étiquetage dynamiques : ils sont adaptés aux documents XML qui sont fréquemment mis à jour [8], [87], [113]–[122].

2.2.2. Taxonomie des techniques d'étiquetage XML

Les techniques d'étiquetage XML peuvent être divisées en quatre catégories [108], [123]–[125] à savoir les techniques d'étiquetage basées sur l'intervalle, les techniques d'étiquetage basées sur le préfixe, les techniques d'étiquetage multiplicatives et les techniques d'étiquetage hybrides.

- **Techniques d'étiquetage basées sur l'intervalle**

Les techniques d'étiquetage basées sur l'intervalle [110], [126]–[130] sont également appelées techniques d'étiquetage par inclusion, techniques d'étiquetage par plage ou techniques d'étiquetage par région [108]. D'ailleurs, elles sont caractérisées par leur simplicité et leur facilité. Ces techniques consistent particulièrement à générer pour chaque nœud trois informations importantes <position de début, position de fin, niveau> [131]:

  - Position de début (*start*) : Elle peut être générée soit en comptant le nombre de mots ou de lettres depuis le début du document jusqu'au début du nœud, ou en utilisant l'algorithme de parcours en profondeur
  - Position de fin (*end*) : Elle peut être obtenue en calculant le nombre de mots ou de lettres depuis le début du document jusqu'à la fin du nœud. Sinon, elle peut être générée tout en utilisant l'algorithme de parcours en profondeur.
  - Niveau du nœud dans le document (*level*) : C'est la profondeur du nœud dans l'arbre de données XML.

L'avantage principale de cette catégorie est que l'étiquette du nœud est généralement compacte. Cependant, ces techniques d'étiquetage souffrent d'une baisse de performance dans un environnement avec une quantité importante des requêtes de mise à jour [124].

Considérons deux nœuds *N1* et *N2* appartenant à un arbre de données XML. On peut extraire les relations structurelles entre ces deux nœuds en utilisant les règles suivantes :

  - Ancêtre-descendant : *N2* est le descendant de *N1* si et seulement si *N1.start* < *N2.début* et *N2.fin* < *N1.fin*
  - Parent-enfant : *N2* est un nœud enfant de *N1* si et seulement si *N2* est un descendant de *N1* et *N2.level = N1.level +1*.

- **Techniques d'étiquetage basées sur le préfixe**

Les techniques d'étiquetage basées sur le préfixe [23], [33], [51], [52], [61], [64], [71]–[74] sont également appelées techniques d'étiquetage basées sur le chemin[124]. Elles consistent à encode l'étiquette du nœud parent comme préfixe de l'étiquette du nœud enfant en utilisant l'algorithme de parcours en profondeur [25], [125]. Après le processus d'étiquetage, l'étiquette finale du nœud est composée d'un ensemble de sous-étiquettes

séparées par un délimiteur ("." ou " ;"), chacune d'entre elles représentant l'ordre local du nœud parmi les nœuds de son ancêtre. Ainsi, Pour vérifier si un nœud *N1* est un ancêtre ou parent d'un autre nœud *N2*, il suffit de vérifier si l'étiquette de *N1* est un préfixe de l'étiquette de *N2* [8], [116], [136].

- **Techniques d'étiquetage multiplicatives**

Les techniques d'étiquetage multiplicatives [117], [118], [122], [137]–[145] implique certaines opérations mathématiques pour la génération d'étiquettes et la détermination des relations entre les nœuds d'un arbre de données XML. Le concept principal de ces techniques est d'attribuer des étiquettes aux nœuds en utilisant des nombres entiers basés sur les propriétés des opérations arithmétiques. Par conséquent, la détermination des relations structurelles entre les nœuds est réalisée en analysant les propriétés arithmétiques des étiquettes numériques à l'aide de principes mathématiques. En plus de la possibilité de résolution des relation structurelles parent-enfant et ancêtre-descendant, les techniques d'étiquetage multiplicatives supportent l'identification des relations supplémentaires telles que le plus petit ancêtre commun (LCA) [146]. La principale limite est que le calcul de l'étiquetage multiplicatif consomme beaucoup de ressources en raison de la nécessité de calculs complexes. Il est donc inapproprié pour étiqueter les grands documents XML.

- **Techniques d'étiquetage hybrides**

Cette classe de techniques d'étiquetage XML [119], [120], [122], [147] consiste à combiner deux ou plusieurs approches en utilisant leurs points forts et en surmontant leurs points faibles dans le but d'accélérer le traitement des requêtes et de minimiser l'espace de stockage des étiquettes des nœuds [124], [148].

3. Notre approche

3.1. Principe de l'approche

L'idée générale de notre approche est de générer automatiquement un cours pour chaque apprenant en fonction de ses caractéristiques et ses préférences individuelles. Cependant, la grande question qui se pose est de savoir structurer, stocker et indexer les contenus d'apprentissage tout en assurant une meilleure adaptation. Dans ce but, nous avons opté pour le langage XML pour représenter notre support de cours. Par ailleurs, afin d'éviter les faiblesses des bases de données XML et de bénéficier des points forts des bases de données relationnelles, nous avons utilisé notre approche E-XMap pour stocker le document XML du

support de cours dans une base de données relationnelle. Parallèlement, nous utilisons les techniques d'étiquetage XML pour identifier les relations structurelles entre les nœuds et aussi pour accélérer le traitement des requêtes.

E-XMap (Enhanced- XMap) est une version améliorée de notre approche XMap abordée dans le chapitre II. L'objectif du processus du mapping est de stocker les données XML dans une base de données relationnelle tout en préservant la hiérarchisation des données. Dans cette intention, nous avons ajouté un module nommé *NodeLabeling*. La fonction de ce dernier est d'étiqueter les nœuds d'un arbre de données XML de façon à maintenir les relations structurelles entre les nœuds à savoir parent, enfant, frères, ancêtre et descendant. L'étiquette d'un nœud doit être compacte pour minimiser l'espace de stockage de la base de données et aussi pour accélérer l'exécution des requêtes. Nous avons proposé de comparer plusieurs techniques d'étiquetage XML en vue de choisir la technique qui offre plus d'options et les meilleures performances au niveau de temps d'étiquetage, l'espace de stockage et le temps d'exécution des requêtes.

La Figure 31 illustre le diagramme de processus de l'approche E-XMap. Le composant *Data Mapper* permet de faire le mapping du document XML contenant le support de cours en utilisant le parseur DOM (*DOM Parser*) qui génère un arbre de données XML (*XML DOM Tree*). La fonction *TreeBrowser()* parcourt l'arbre de données XML et utilise le module *NodeLabeling* pour étiqueter les nœuds en appliquant un des algorithmes des techniques d'étiquetage que nous allons présenter par la suite. Après l'étiquetage d'un nœud, la fonction *TreeBrowser()* exécute des instructions SQL pour insérer et stocker les informations relatives au nœud dans les tables concernées de notre schéma relationnel. Quant au composant *Query Mapper*, il permet de traduire les requêtes XML en requêtes SQL. Le rôle de ces requêtes XML est d'extraire une partie ou un morceau de cours contenant l'objet d'apprentissage adapté aux préférences de l'apprenant. Le composant *Query Mapper* retourne un résultat sous le format relationnel et le composant *RDBtoXML Mapper* s'occupe de reconstruit le document XML en transformant le résultat relationnel en un format de données hiérarchique (XML). Pour affiche le résultat XML à l'apprenant sous forme d'une page web le module de présentation se charge d'exécute des règles de transformation exprimées en XSLT.

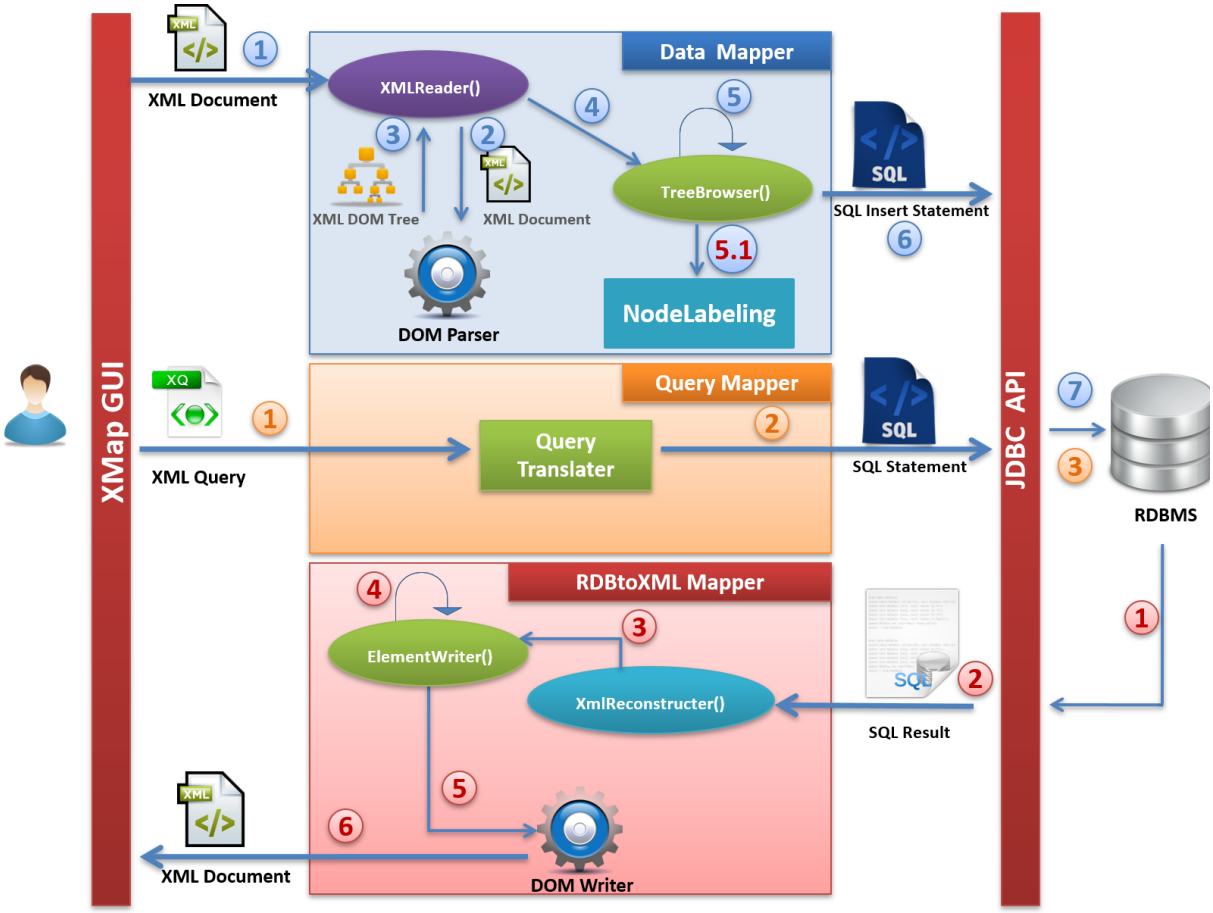


Figure 31: Diagramme de processus de l'approche E-XMap

3.2. Support de cours

Dans cette section, nous allons présenter les composants du cours, qui est un élément essentiel dans tous les systèmes d'apprentissage. Nous créons la structure d'un cours personnalisé en utilisant le langage XML et en spécifiant les différentes couches constituant un cours en se basant sur la hiérarchie du contenu présentée dans la Figure 32. Cette hiérarchie est composée de quatre couches comme décrites ci-dessous :

- **Cours**: Représente les ressources pédagogiques, chaque ressource comporte plusieurs unités.
- **Unité**: Décrit le contenu d'un cours, chaque unité contient plusieurs concepts.
- **Concept**: Représente l'ensemble des concepts pour une unité donnée, les concepts sont liés par des relations.
- **Média**: Représente chaque concept sous différents formats. il comprend quatre formats de contenu à savoir texte, image, vidéo et audio.

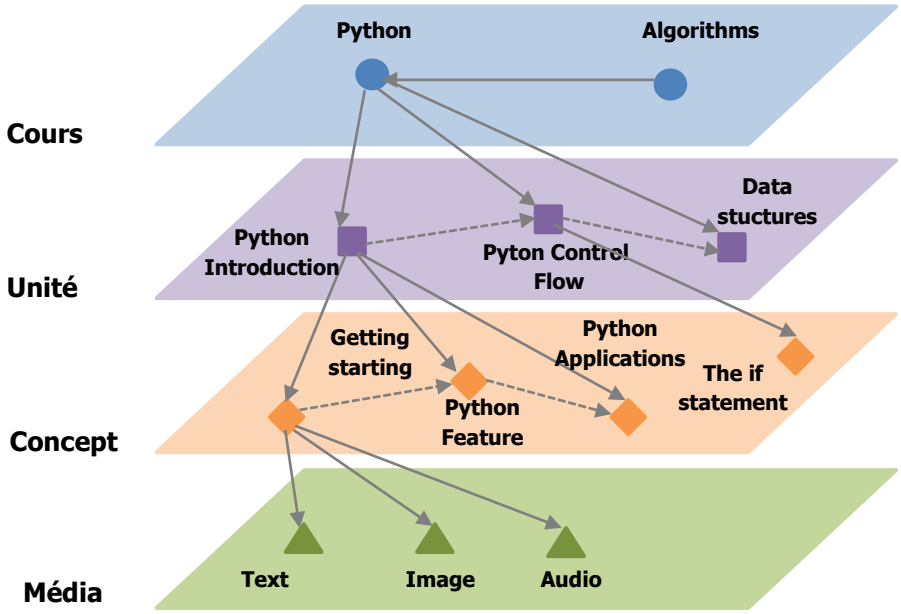


Figure 32: Structure du Cours

3.3. Etiquetage des nœuds d'un arbre de données XML d'un support de cours

Dans cette section, nous présenterons des techniques d'étiquetage dynamiques qui supportent la mise à jour dans un document XML. Dans notre approche, le support de cours est présenté sous forme d'un document XML qui peut être mise à jour et modifié par l'auteur. Nous utilisons, à cet effet notre approche *E-XMap* avec le module *NodeLabeling* qui se charge d'étiqueter les nœuds d'un arbre de données d'un document XML (le support de cours dans notre cas) en utilisant des techniques d'étiquetage dynamiques. Nous appliquons ces techniques d'étiquetage sur l'arbre de données XML (cf. Figure 34) de notre support de cours de la Figure 33. Après l'étiquetage du document XML de support de cours, le document XML généré sera stocké dans une base de données relationnelle comme indiqué dans la Figure 31.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<course>
    <about>
        <title>Python</title>
        <identifier>C1234F</identifier>
        <level>beginner</level>
        <authors>Zakaria BOUSALEM</authors>
        <keywords>Python, Python programming</keywords>
    </about>
    <unit title="Python introduction" number="1">
        <concept title="Getting starting">
                <text>Python is a cross-platform programming ...</text>
                <image>whatIsPython.png</image>
                <audio>whatIsPython.mp3</audio>
        </concept>
    </unit>
    <unit title="Python control flow " number="2">
    <concept title="The if statement">
                <text>The if statement is used to check a condition: ...</text>
                <image>ifStatement.png</image>
                <audio>ifStatement.mp3</audio>
    </concept>
    </unit>
</course>
```

Figure 33: Document XML du cours de Python

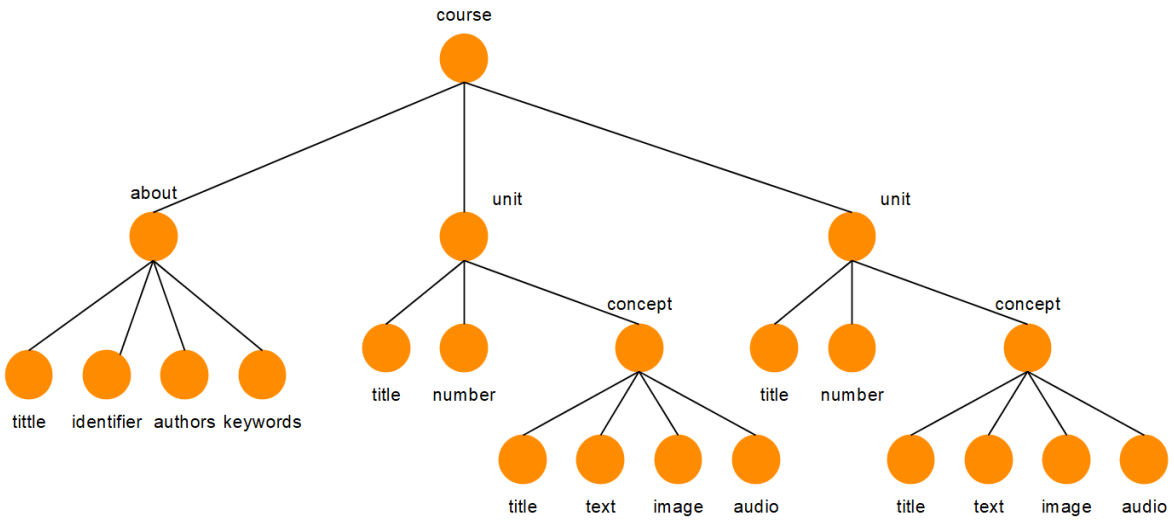


Figure 34: Arbre de données XML du cours de Python

3.2.1. ORDPATH

ORDPATH [87] est une technique d'étiquetage hiérarchique basée sur le préfix, elle est mise en œuvre et implémentée dans Microsoft SQL Server. Contrairement à la méthode d'étiquetage Dewey [25], *ORDPATH* supporte l'insertion, la suppression et la modification des nœuds d'un arbre de données XML [89], [124], [136]. Afin d'attribue une étiquette unique pour un nœud enfant lors de la phase d'étiquetage initiale, *ORDPATH* concatène l'étiquette du parent et un entier impair et positif. Ce dernier représente étiquette locale du nœud enfant. Les nombres pairs et négatifs sont réservés pour l'insertion des nouveaux nœuds

[87]. Par exemple, si un nœud doit être inséré à droite de tous les nœuds enfants existants, *ORDPATH* génère son étiquette en ajoutant +2 au dernier ordinal du dernier enfant. Toutefois, si le nœud nouvellement inséré doit être ajouté à gauche de tous les enfants existants, son étiquette est générée en ajoutant -2 au dernier ordinal du premier enfant.

Considérons un nœud parent *N1* ayant "**1.5.3**" comme étiquette et *N2*, *N3* et *N4* sont ses nœuds enfants avec respectivement les étiquettes suivantes : "**1.5.3**.1", "**1.5.3**.3" et "**1.5.3**.5".

- Pour insérer un nœud enfant à l'extrémité droite, *ORDPATH* génère son étiquette en ajoutant +2 à l'étiquette locale du dernier nœud enfant ("**1.5.3**.5"). L'étiquette du nouveau nœud est "**1.5.3**.7".
- Afin d'insérer un nœud à gauche de tous les enfants existants, son étiquette est générée en ajoutant -2 à l'étiquette locale du premier nœud enfant. L'étiquette du nouveau nœud est "**1.5.3**.-1".
- Pour insérer un nouveau nœud entre deux nœuds frères, *ORDPATH* ajoute le nombre pair situant entre les étiquettes locales (impaires) de deux nœuds frères, puis elle le concatène avec un nombre impair, généralement 1. Par exemple, on veut ajouter trois nœuds entre *N2* ("**1.5.3**.1") et *N3* ("**1.5.3**.3"), les étiquettes des nouveaux nœuds sont :"**1.5.3.2**.1", "**1.5.3.2**.3" et "**1.5.3.2**.5". La valeur 2 ajoutée (ou toute valeur paire utilisée) représente uniquement un signe d'insertion, c'est-à-dire qu'elle est ignorée lors de l'évaluation des relations parent-enfant et ancêtre-descendant : "**1.5.3.2**.1" est un nœud enfant de "**1.5.3**" et aussi c'est un nœud descendant de "**1**". Néanmoins, le signe d'insertion a un effet sur l'ordre lexicographique des étiquettes, puisque la comparaison "**1.5.3**.1" < "**1.5.3.2**.1", "**1.5.3.2**.3", "**1.5.3.2**.5" < "**1.5.3**.3" est vraie.

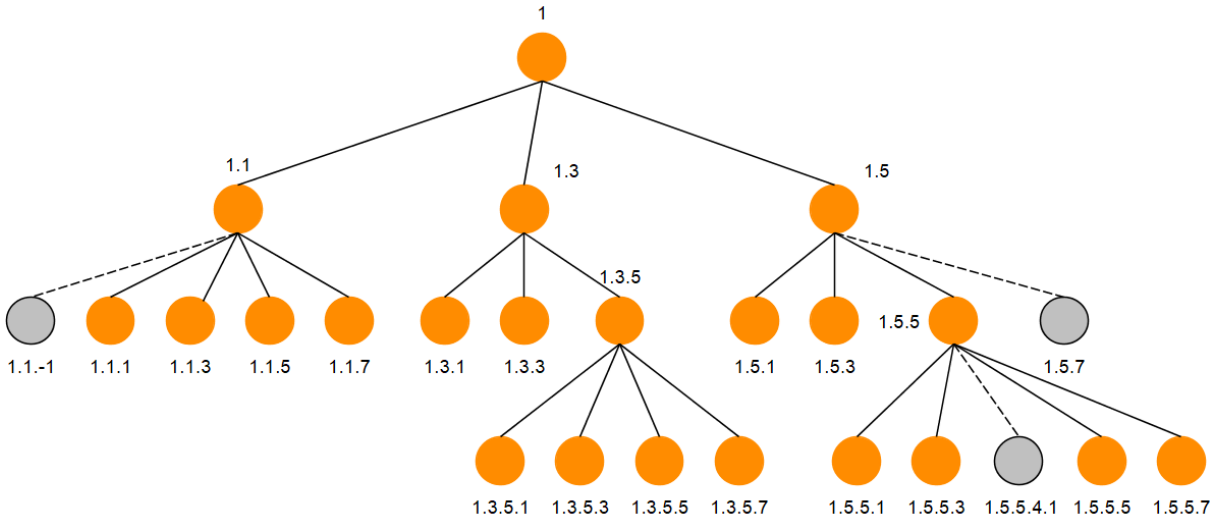


Figure 35: Arbre de données XML du cours de Python étiqueté avec *ORDPATH*

Cependant, cette technique génère des étiquettes moins compactes et gaspille la moitié des nombres possibles pour l'étiquetage des nœuds en raison de son mécanisme d'insertion. C'est pourquoi, *ORDPATH* n'est pas adaptée aux documents XML ayant un arbre de données profond [108], [119], [141]. La Figure 35 montre l'arbre de données XML de notre support de cours (cf. Figure 33) étiqueté avec la technique *ORDPATH*. Les nœuds gris avec un arête en pointillés indiquent les nœuds nouvellement insérés.

Algorithm 3: ORDPATH Labelling Scheme

Input: XML DOM Tree.
Output: XML Nodes labeled using ORDPATH technique.

```
1  root.label ← '1'
2  ORDPATHinitialLabeling(root)
3  Function ORDPATHinitialLabeling(Element parentElm):
4      listChildNodes ← parentElm.childNodes
5      childNumber ← 0
6      parentLabel ← parentElm.label
7      foreach childNode ∈ listChildNodes do
8          childNode.label ← conact(parentLabel, '.', (childNumber*2+1))
9          childNumber++
10         if | childNode.childNodes | > 0 then
11             ORDPATHinitialLabeling(childNode)
12         end if
13     end foreach
14 End Function
```

Figure 36: Algorithme de la technique *ORDPATH*

La Figure 36 présente l'algorithme de la technique *ORDPATH* que nous avons proposé en se basant sur le principe de fonctionnement de cette technique décrit dans l'article [87].

3.2.2. Dynamic XDAS

L'approche *Dynamic XDAS* [119] peut être classée dans la catégorie des systèmes d'étiquetage hybrides. Elle utilise l'approche IBSL [149] pour éviter le réétiquetage des nœuds lors de l'ajout ou la mise à jour de l'arbre de données XML. Pour étiqueter les nœuds, *Dynamic XDAS* utilise l'adressage IP et la technique de subnetting (sous-réseautage) [150] couramment utilisés dans les réseaux informatiques. Dans la phase d'étiquetage, *Dynamic XDAS* attribue à chaque nœud une étiquette sous la forme <*Niveau*, *Nombre*> où *Niveau* est la profondeur du nœud dans le document et *Nombre* est un identifiant unique généré en utilisant la technique de subnetting et l'adressage IP. Cette technique représente *Niveau* avec un seul octet et *Nombre* avec des nombres binaires (0 et 1) et elle le converti ensuite en nombres hexadécimaux afin de minimiser l'espace nécessaire pour le stockage des étiquettes. Pour identifier les relations structurelles parent-enfant, frères et ancêtre-descendant, *Dynamic XDAS* utilise les opérations logiques, ce qui améliore considérablement le temps nécessaire pour déterminer les relations structurelles entre les nœuds.

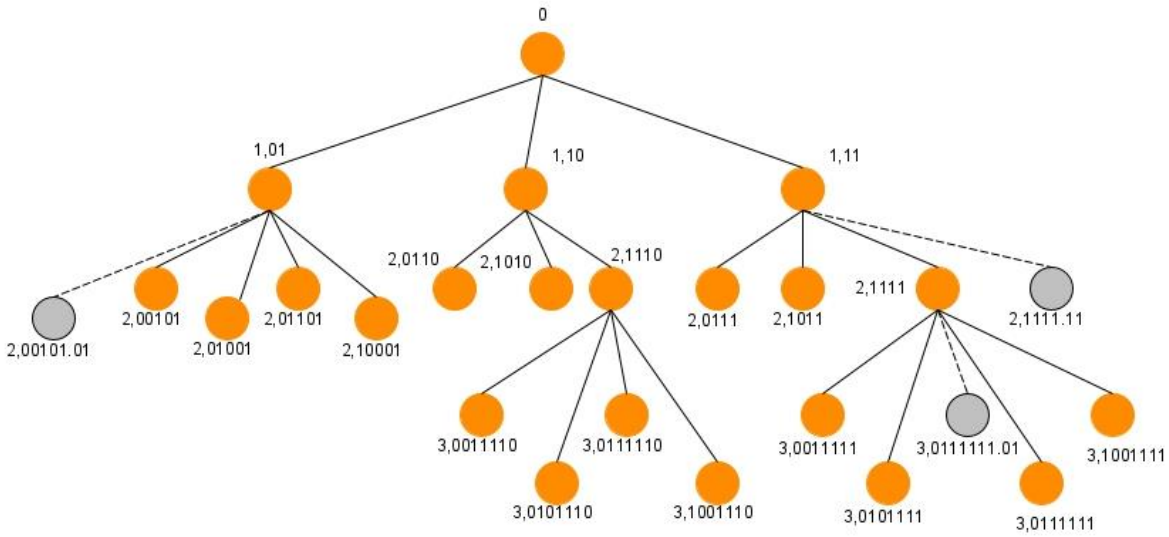


Figure 37: Arbre de données XML du cours de Python étiqueté avec *Dynamic XDAS*

L'équipe de Ghaleb [119] ont modifié l'approche *IBSL* [149] pour qu'elle soit compatible avec la version originale de *XDAS* [151]. Ils ont adopté dans l'approche modifiée quatre cas principaux pour l'opération d'insertion :

- L'insertion d'un nœud à l'extrémité gauche (2 sous-cas).
- L'insertion d'un nœud à l'extrémité droite (2 sous-cas).
- L'insertion d'un nœud entre deux nœuds frères (3 sous-cas).

- L'insertion d'un sous-arbre à n'importe quelle position de l'arbre de données XML (1 sous-cas).

La Figure 37 présente l'arbre de données XML de notre support de cours (cf. Figure 33) étiqueté avec l'approche *Dynamic XDAS*. Les nœuds gris avec un arête en pointillé indiquent les nœuds insérés. Il y a 8 sous-cas d'insertion de nœuds proposés dans l'approche *Dynamic XDAS*, seulement trois sont appliqués dans cette figure.

Nous proposons l'algorithme illustré dans la Figure 38 pour la technique *Dynamic XDAS* on se basant sur sa logique de fonctionnement présentée dans les travaux [119], [120], [151].

Algorithm 4: XDAS Labelling Scheme

```
   Input: XML DOM Tree.
   Output: XML Nodes labeled using XDAS technique.
1  root.label ← '0'
2  XDASinitialLabeling(root, 0, true)
3  Function XDASinitialLabeling(Element parentElm, int parentLevel,
     boolean parentIsRoot):
4      listChildNodes ← parentElm.childNodes
5      parentChildNum ← | listChildNodes |
6      childNumber ← 0
7      parentLabel ← convertHex2Bin(parentElem.label)        // convertHex2Bin is a
         function that convert hexadecimal to binary
8      foreach childNode ∈ listChildNodes do
9          childNumber++
10         childLevel ← parentLevel + 1
11         childNumberBin ← convertDec2Bin(childNumber, parentChildNum)
             // convertDec2Bin is a function that convert decimal to binary
12         if parentIsRoot then
13             childLabel ← concat(childLevel, ',', convertBin2Hex(childNumberBin))
                 // convertBin2Hex is a function that convert binary to hexadecimal
14         else
15             childLabelPart2 ← concat(childNumberBin, parentLabel)
16             childLabel ← concat(childLevel, ',', convertBin2Hex(childLabelPart2))
17         end if
18         childNode.label ← childLabel
19         if | childNode.childNodes | > 0 then
20             XDASinitialLabeling(childNode, childLevel, false)
21         end if
22     end foreach
23 End Function
```

Figure 38: Algorithme de la technique *Dynamic XDAS*

3.2.3. GroupBased

L'objectif de cette approche [141] est de former des groupes d'arbres composés par un nœud parent et ses nœuds enfants, afin d'identifier rapidement les relations structurelles et de faciliter l'insertion de nouveaux nœuds dans le document XML, étant donné que le traitement des petits arbres de données est plus facile et plus rapide que le traitement de l'arbre entier.

Dans cette technique, chaque nœud a deux étiquettes : une étiquette globale et une étiquette locale. L'étiquette globale est utilisée pour identifier facilement les relations entre les nœuds en reliant les petits groupes à l'arbre entier. Le rôle de l'étiquette locale est de faciliter l'identification des relations structurelles et les opérations d'insertion dans le même groupe.

Le processus d'étiquetage se déroule en deux étapes :

- Première étape : consiste à diviser l'arbre de données XML en groupes simples contenants chaque nœud avec ses nœuds enfants. Chaque groupe sera étiqueté par une étiquette globale qui se compose de deux parties. La première partie est une étiquette générée selon la technique Dewey [25]. La deuxième partie est l'ordre du chaque nœud enfant, en commençant de gauche à droite. Cette partie est utilisé pour maintenir l'ordre du document.
- Deuxième étape : consiste à générer l'étiquette locale de chaque nœud.

Cette technique permet d'identifier les relations structurelles suivantes : ancêtre-descendant, parent-enfant, frères et le plus petit ancêtre commun (LCA).

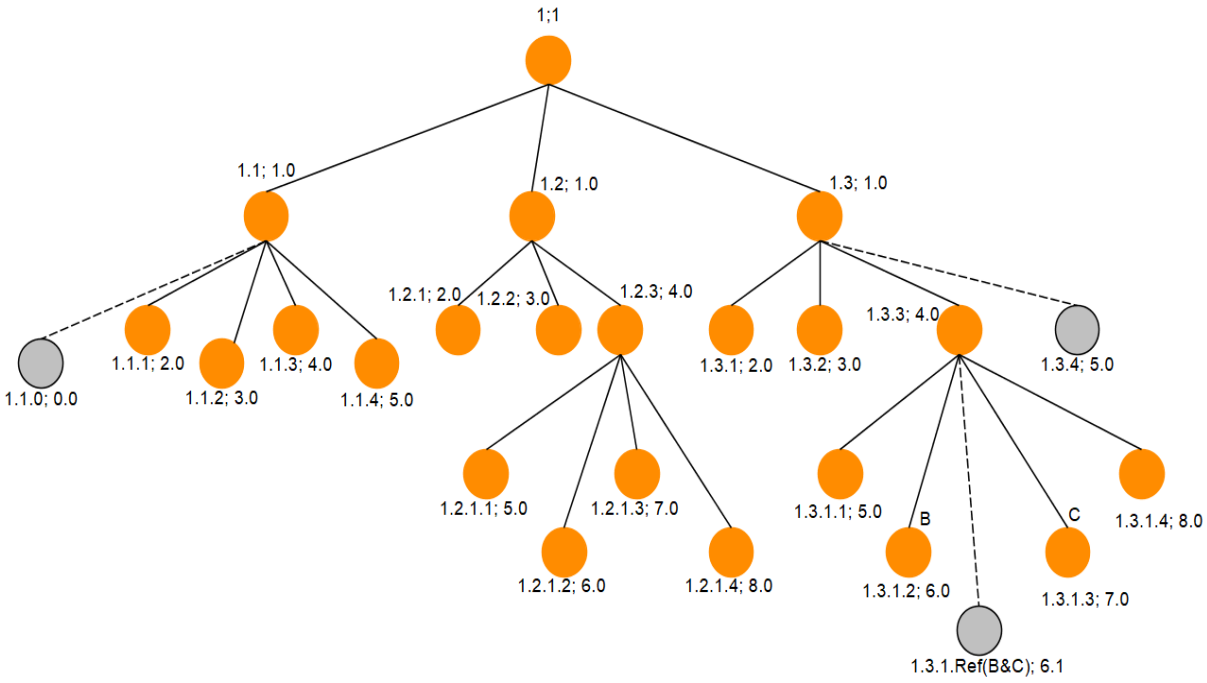


Figure 39: Arbre de données XML du cours de Python étiqueté avec *GroupBased*

L'approche *GroupBased* supporte efficacement les opérations de mise à jour et d'insertion. Quant à l'insertion d'un nouveau nœud, cette technique comprend quatre cas principaux :

- L'insertion d'un nœud à l'extrémité gauche (2 sous-cas).
- L'insertion d'un nœud à l'extrémité droite (2 sous-cas).
- L'insertion d'un nœud entre deux nœuds frères (3 sous-cas).
- L'insertion d'un nœud sous un nœud feuille (nœud enfant) (2 sous-cas).

La Figure 39 présente l'arbre de données XML de notre support de cours (cf. Figure 33) étiqueté avec la technique *GroupBased*. Les nœuds gris avec un arête en pointillé indiquent les nœuds insérés. L'approche *GroupBased* comprend 9 sous-cas d'insertion de nœuds, seulement trois sont illustrés dans cette figure.

Algorithm 5: GroupBased Labelling Scheme

```
   Input: XML DOM Tree.
   Output: XML Nodes labeled using GroupBased technique.
 1 root.globalLabel ← '1'
 2 root.localLabel ← '1'
 3 GroupBasedInitialLabeling(root, true)
 4 Function GroupBasedInitialLabeling(Element parentElm,
     boolean parentIsRoot):
 5 |  listChildNodes ← parentElm.childNodes
 6 |  childNumber ← 0
 7 |  parentGlobalLabel ← parentElm.globalLabel
 8 |  parentFirstPartLocalLabel ← parentElm.firstPartLocalLabel
 9 |  foreach childNode ∈ listChildNodes do
10 |  |  childNumber++
11 |  |  if parentIsRoot then
12 |  |  |  childLocalLabel ← '1.0'
13 |  |  else
14 |  |  |  childLocalLabel ← concat((parentFirstPartLocalLabel + childNumber), '.0')
15 |  |  end if
16 |  |  childNode.globalLabel ← concat(parentGlobalLabel, '.', childNumber)
17 |  |  childNode.localLabel ← childLocalLabel
18 |  |  if | childNode.childNodes | > 0 then
19 |  |  |  GroupBasedInitialLabeling(childNode, false)
20 |  |  end if
21 |  end foreach
22 End Function
```

Figure 40: Algorithme de la technique *GroupBased*

La Figure 40 montre l'algorithme que nous avons établi pour l'approche *GroupBased*[141].

3.2.4. Dynamic Prefix-based Labeling Scheme (DPLS)

La technique *DPLS* [125] peut être classée dans la catégorie des systèmes basés sur le préfixe. C'est la version améliorée de la technique *DFPD* [113]. *DPLS* a une caractéristique distinctive, elle a la capacité de réutiliser les étiquettes de nœuds supprimées dans un contexte dynamique, et donc, elle peut limiter le taux de croissance de la taille des étiquettes en cas d'insertions et de suppressions fréquentes de nœuds. L'objectif de cette approche est de réduire

considérablement le temps de traitement des mises à jour et le temps d'exécution des requêtes, de minimiser l'espace de stockage des étiquettes de nœuds et d'éviter le réétiquetage des nœuds dans un contexte dynamique. L'approche *DPLS* utilise la technique Dewey [25] dans la phase initiale d'étiquetage [46], [62], [125], [75].

Concernant l'opération d'insertion d'un nouveau nœud, l'approche du *DPLS* comporte quatre cas principaux :

- L'insertion d'un nœud à l'extrémité gauche.
- L'insertion d'un nœud à l'extrémité droite.
- L'insertion d'un nœud entre deux nœuds frères.
- L'insertion d'un nœud sous un nœud feuille (nœud enfant).

DPLS utilise les fractions dans l'étiquette locale d'un nœud dans le cas d'insertion d'un nœud entre deux nœuds frères consécutifs. A titre d'exemple, pour insérer un nœud *N3* entre deux nœud frères *N1* et *N2* ayant respectivement les étiquettes suivantes : "1.2.3" et "1.2.4". l'étiquette de *N3* est "1.2.((3+4)/(1+1))" = "1.2.(7/2)". Cependant, les nombres à virgule flottante générés par cette technique ont une précision limitée puisque la mantisse est en fait représentée par un nombre fixe de bits et en pratique elle peut être étendue jusqu'à 2 bits par insertion, ce qui entraîne le problème de dépassement de mémoire (*Overflow problem*). Pour surmonter ce problème, la technique *DPLS* adopte le format de stockage à longueur variable utilisé dans l'approche *ORDPATH* [87], [46], [62], [125], [75].

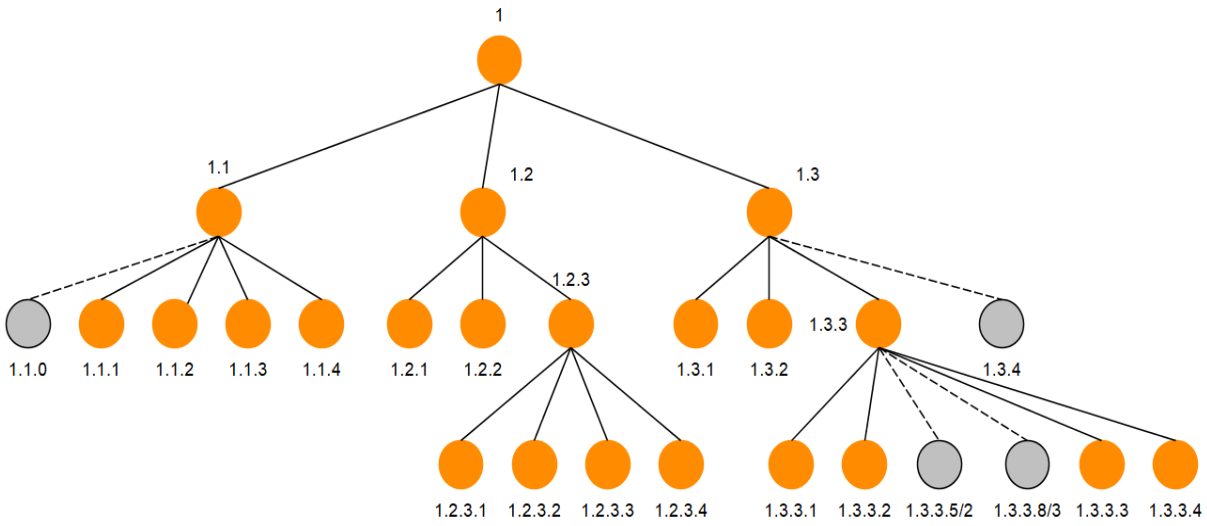


Figure 41: Arbre de données XML du cours de Python étiqueté avec *DPLS*

La Figure 41 présente l'arbre de données XML de notre support de cours (cf. Figure 33) étiqueté avec la technique *DPLS*. Les nœuds gris avec un arête en pointillé indiquent les nœuds insérés.

Algorithm 6: DPLS Labelling Scheme

Input: XML DOM Tree.
Output: XML Nodes labeled using DPLS technique.

```
1  root.label ← '1'
2  DPLSinitialLabeling(root)
3  Function DPLSinitialLabeling(Element parentElm):
4      listChildNodes ← parentElm.childNodes
5      childNodeNumber ← 0
6      parentLabel ← parentElm.label
7      foreach node ∈ listChildNodes do
8          childNodeNumber++
9          node.label ← concat(parentLabel, '.', childNodeNumber)
10         if | node.childNodes | > 0 then
11             DPLSinitialLabeling(node)
12         end if
13     end foreach
14 End Function
```

Figure 42: Algorithme de la technique *DPLS*

La Figure 42 présente l'algorithme que nous avons établi pour l'approche *DPLS* [125].

Les Tableau 3 et Tableau 4 récapitulent respectivement les avantages et les inconvénients de chaque technique d'étiquetage et les relations structurelles supportées.

Tableau 3: Récapitulatif des techniques d'étiquetage

	Avantages	Inconvénients
ORDPATH [87]	Supporte efficacement l'insertion/mise à jour sans le réétiquetage des nœuds existants.	N'est pas adaptée au document XML profonds. Elle génère des étiquettes moins compactes et gaspille la moitié des nombres possibles pour l'étiquetage des nœuds.
Dynamic XDAS [119]	Les nouveaux nœuds peuvent être insérés sans le réétiquetage des nœuds existants. Efficace au niveau de l'identification de relations	Cette technique nécessite un espace de stockage important puisque la taille des étiquettes des nœuds augmente rapidement dans le cas d'insertions répétitifs.

	structurelles car elle utilise les opérations logiques qui sont plus efficaces que la comparaison des paramètres d'inclusion ou l'utilisation des fonctions de traitement des chaines de caractères.	
GroupBased [141]	Supporte l'insertion des nouveaux nœuds sans le réétiquetage des nœuds existants. Identification souple et rapide des différentes relations structurelles.	La taille des étiquettes est grande puisque cette technique utilise deux étiquettes pour chaque nœud ; globale et locale.
DPLS [125]	Recyclez les étiquettes des nœuds supprimés afin de les utilisées pour les nouveaux nœuds insérés.	Nécessite un certain temps pour l'insertion du nœud car il utilise la fraction pour déterminer la valeur du nœud. Le processus de décodage des étiquettes est compliqué puisque cette technique utilise le codage à longueur variable dans la phase d'étiquetage.

Tableau 4: Relations structurelles supportées par chaque technique

	Relations structurelles			
	ancêtre-descendant	Parent-Enfant	Nœuds frères	LCA*
ORDPATH [87]	✓	✓	✓	✓
Dynamic XDAS [119]	✓	✓	✓	✗

GroupBased [141]	✓	✓	✓	✓
DPLS [125]	✓	✓	✓	✓

* LCA (Lowest Common Ancestor) : le plus petit ancêtre commun.

4. Conclusion

Dans ce chapitre, nous avons proposé une approche qui permet de produire un contenu d'apprentissage adaptatif basé sur les caractéristiques et les préférences individuelles des apprenants en utilisant le langage XML pour représenter le support de cours. Le document XML du support de cours sera stocké dans une base de données relationnelle et afin d'identifier les relations entre les nœuds et d'accélérer le traitement des requêtes, nous utilisons les techniques d'étiquetage XML. L'avantage de cette approche est d'éviter les limitations des bases de données XML et de bénéficier des forces des technologies matures fournies par les SGBD relationnels.

Chapitre IV

Migration des Bases de Données Relationnelles vers le NoSQL : Une Etude de Faisabilité

1. Introduction

Edgar F. Codd a proposé l'algèbre relationnelle dans son article « *A relational model of data for large shared data banks* » [2] comme étant une base théorique pour les bases de données relationnelles, en particulier le langage d'interrogation de ces bases de données. Cependant, les systèmes de bases de données relationnelles ont montré leurs limites pour la croissance exponentielle des données générées par la transformation digitale. La migration d'une base de données relationnelle vers une base de données NoSQL nécessite donc une évaluation de faisabilité de la migration. Cette contribution examine cette question, en proposant un modèle relationnel aux bases de données NoSQL, en particulier la base de données orientée colonnes HBase. La modélisation permettra de comparer le modèle des bases de données relationnelles et de HBase, en examinant les opérations de base de l'algèbre relationnelle dans le modèle de données HBase et en citant quelques implémentations des opérations clés de l'algèbre relationnelle dans la base de données de HBase. Pour cela, nous allons comparer les capacités du modèle relationnel et le modèle de HBase en se basant sur la formule suivante :

$$capacités_{HBase} \geq capacités_{Relationnel}$$

Si les capacités du modèle de HBase sont les mêmes de celles du modèle relationnel, ou même si le modèle de HBase a des capacités plus importantes que le modèle relationnel, on peut déduire que les données d'une base de données relationnelle peuvent être migrées vers une base de données HBase.

2. Concepts de base

2.1. Les bases de données orientées colonnes

Les bases de données de cette catégorie sont basées sur le modèle de BigTable [71] (la base de données NoSQL de Google) et c'est la raison pour laquelle elles sont également appelées "les clones de BigTable". Elles sont conçues pour stocker et traiter des téraoctets, voire des pétaoctets de données. Selon Chang *et al*. [71], BigTable est une table triée, multidimensionnelle, distribuée et persistante. Elle est indexée par une valeur dite "clé de ligne", une clé de colonne et un horodatage(timestamp). Le modèle de BigTable a ensuite

inspiré d'autres bases de données orientés colonne, comme HBase[1] [152], Apache Parquet[2], Hypertable[3], Amazon Redshift[4] et Cassandra[5] [153], [154].

Depuis les années 70, plusieurs études ont été réalisées afin de trouver et de montrer les avantages de partitionnement vertical par rapport aux le NSM, le modèle de base des bases de données relationnelles [155]. Cependant, plusieurs facteurs techniques et commerciales ont permis aux systèmes de bases de données traditionnels de garder leur place dans le marché [156].

Le principe général des bases de données orientées colonnes est opposé à celui des bases de données relationnelles. Alors que les SGBDR stockent les données dans des lignes l'une après l'autre, les bases de données orientées colonnes répartit les données selon des colonnes et stockent chaque colonne dans sa propre structure [157], [158]. Cela permet de minimiser l'utilisation des ressources liées aux requêtes sur les grands jeux de données. La Figure 43 illustre la différence entre l'organisation d'une table dans une base de données relationnelle et l'organisation d'une table dans une base de données orientée colonnes.

L'un des principaux avantages d'une base de données orientée colonnes est que les données peuvent être fortement compressées. La compression permet d'effectuer très rapidement les opérations orientées colonnes tel que la mise à jour des colonnes, les opérations d'agrégations (MIN, MAX, SUM, COUNT et AVG), la sélection de quelques colonnes, etc.

Les bases de données orientées colonnes ne sont pas adaptées applications transactionnelles. Selon Moniruzzaman *et al*. [69], Elles sont surtout utiles pour :

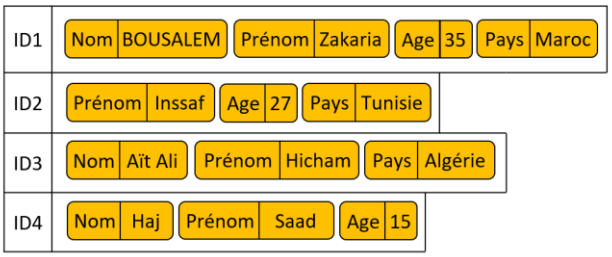
- Le stockage des gros volumes de données qui font l'objet de nombreuses requêtes de lecture et de peu de requêtes d'écriture, en particulier les données versionnées.
- Le traitement par lots des données distribuées y compris le tri, l'analyse syntaxique, et la conversion des données, etc.

[1] http://hbase.apache.org/
[2] https://parquet.apache.org/
[3] https://hypertable.org/
[4] https://aws.amazon.com/redshift/
[5] https://cassandra.apache.org/

- L'analyse statistique des données.

ID	Nom	Prénom	Age	Pays
ID1	BOUSALEM	Zakaria	35	Maroc
ID2		Karima	41	Tunisie
ID3	Aït Ali	Hicham		Algérie
ID4	Haj	Saad	15	

Organisation d'une table dans une base de données relationnelle



Organisation d'une table dans une base de données orientée colonnes

Figure 43: Différence entre l'organisation d'une table dans une BDD relationnelle et l'organisation d'une table dans une BDD orientée colonnes

2.2. HBase

HBase est un système de gestion de bases de données distribué, orienté colonnes et basé sur le Framework Hadoop de la fondation Apache. Il est inspiré de BigTable [71] le système de gestion de bases de données NoSQL de Google. HBase offre un accès aléatoire et en temps réel aux données, il garantit la cohérence de grandes quantités de données non structurées et semi-structurées. Une base de données HBase n'a pas de schéma prédéfini, elle est organisée sous forme des familles de colonnes, elle peut être interrogée en utilisant le langage de requêtes de HBase et aussi par le biais du Framework MapReduce. HBase utilise le HDFS comme système de fichiers pour le stockage des données et Zookeeper, en tant que service de coordination de systèmes distribués.

2.2.1. Architecture et composants

Physiquement, l'architecture de HBase comporte quatre composants principaux : *Région* (*Region*), *HMaster*, Serveur de Région (*Region Server*) et *ZooKeeper* [152] (cf. Figure 44).

- **Région**

Lorsque la taille de la table dépasse la valeur prédéfinie, et pour garantir l'évolutivité, HBase divise automatiquement cette table en différentes zones, chacune d'entre elles

contenant un sous-ensemble de toutes les lignes de la table. Pour l'utilisateur, chaque table est un ensemble des lignes et des colonnes, chaque ligne se distingue par une clé unique (RowKey). Physiquement, une table est divisée en plusieurs blocs, dont chacun est une Région (*Region*) [159]. Une Région stocke une plage contiguë de lignes d'une table. L'ensembles des données d'une table sont stockées dans plusieurs Régions. Ces Régions sont affectées et gérées par des nœuds appelés les Serveurs de Région.

- **Serveur de Région**

Les données de HBase sont généralement stockées dans HDFS, le système de fichiers du Framework Hadoop. Les utilisateurs peuvent obtenir ces données par le biais des nœuds appelés les Serveurs de Région. En général, un seul Serveur de Région fonctionne sur un nœud dans un cluster, et chaque Région n'est gérée que par un seul Serveur de Région. Un Serveur de Région est principalement responsable de la lecture et de l'écriture des données dans le système de fichiers HDFS en réponse aux requêtes des machines clientes [101] . C'est le module de base de HBase. Le Serveur de Région gère en interne un ensemble des Régions, chaque Région correspondant à une plage contiguë de lignes d'une table [160].

- **HMaster**

Le *HMaster* coordonne le cluster HBase, il est responsable des opérations LDD (langage de définition de données), telles que la création, la modification et la suppression des tables. Au démarrage, ce composant assigne chaque Région à un Serveur de Région actifs dans le cluster [159]. Ainsi, le *HMaster* peut déplacer une Région d'un Serveur de Région à un autre dans le cadre d'une opération périodique d'équilibrage de charge. Ce composant gère également les pannes de Serveur de Région en affectant ses régions à un autre Serveur de Région.

- **ZooKeeper**

Pour chercher une donnée dans une table HBase en utilisant un *rowKey*, HBase retrouve la Région contenant le *rowKey* en question dans la table principale de métadonnées de HBase "*hbase:META*". Le rôle de cette table est de mettre en correspondance les régions avec les Serveurs de Région. Cette table est stockée dans un composant nommé *ZooKeeper* [161]. *ZooKeeper* est un projet open-source. HBase utilise *ZooKeeper* comme un service de coordination distribué pour coordonner l'ouverture, la fermeture, la division, le déplacement et

la récupération des Régions entre le *HMaster* et les Serveurs de Région impliqués dans ces opérations. *ZooKeeper* est un serveur de surveillance centralisé qui fournit des services tels que la maintenance des informations de configuration, le nommage, la synchronisation distribuée, la notification des pannes de serveur, etc. Chaque fois qu'un client souhaite communiquer avec des Serveurs de Région, il doit d'abord s'adresser à *ZooKeeper*.

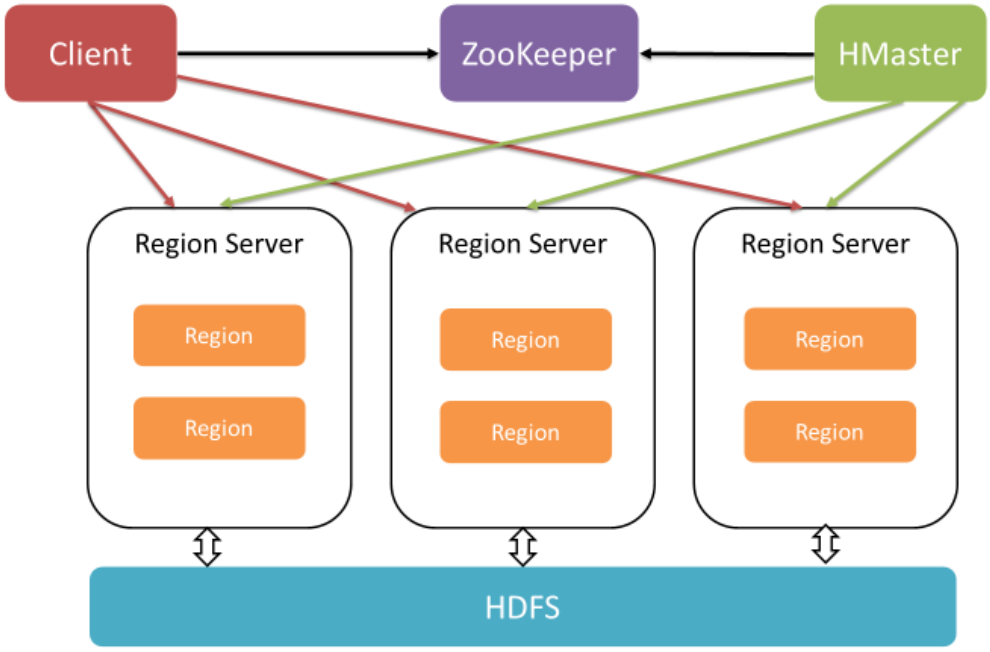


Figure 44: Architecture de HBase

2.2.2. Modèle de donnée de HBase

Comme il est montré dans la Figure 45, le modèle de données de HBase est basé sur six concepts [162] :

- Table : est un ensemble de lignes stockées dans des partitions séparées appelées régions. Les noms de tables ne doivent pas contenir les caractères interdits par le système de fichiers de système d'exploitation utilisé.
- Ligne : Une ligne dans HBase se compose d'une clé unique (RowKey) et d'une ou plusieurs colonnes avec les valeurs qui leur sont associées. Les RowKey sont uniques dans une table et les lignes sont triées lexicographiquement.
- Famille de colonnes : Elle consiste en un ensemble de colonnes physiquement colocataires et stockées dans le même fichier (HFile). Chaque famille de colonnes possède ses propres paramètres de configuration. Chaque ligne d'une table possède la ou les mêmes familles de colonnes, qui peuvent être remplies

ou non. Les noms de familles de colonnes doivent contenir juste les caractères autorisés par le système de fichiers de système d'exploitation utilisé.

- Colonne : Une famille de colonnes est composée d'une ou plusieurs colonnes. Une colonne est identifiée par un qualificatif de colonne qui compose par le nom de famille de colonne concaténé avec le nom de la colonne à l'aide de deux points. Il peut y avoir plusieurs colonnes dans une famille de colonnes et les lignes d'une table peuvent avoir un nombre variable de colonnes.
- Cellule : Une cellule est une combinaison d'un RowKey, d'une famille de colonnes et d'un qualificatif de colonne. Elle permet de stocker les données et elle contient une valeur et un horodatage (timestamp).
- Version : Les données stockées dans une cellule sont versionnées et les versions des données sont identifiées par leurs horodatage.

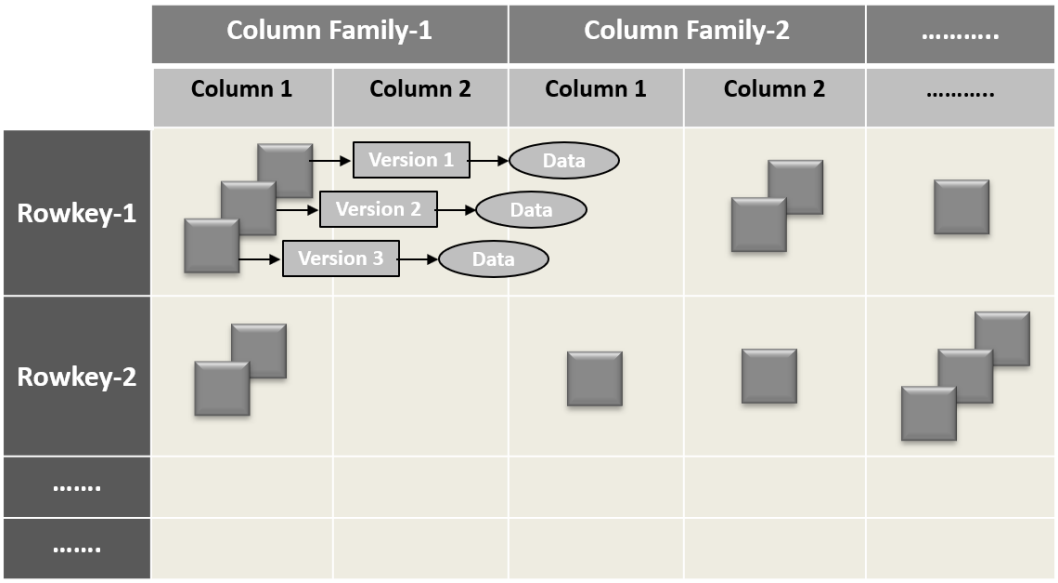


Figure 45: Modèle de données de HBase

2.3. Justification de choix de HBase

Le choix de HBase est motivé par ses points forts qui permettent de surmonter les limitations des bases de données traditionnelles, voire surpasser d'autres bases de données NoSQL dans certains contextes [163]. En effet, HBase est une composante de l'écosystème Hadoop qui utilise le HDFS comme système de fichiers. Cette propriété de HBase fournit automatiquement la fiabilité et la disponibilité aux données grâce à la réplication. A l'instar de HDFS, HBase est également distribué et très évolutif. Elle est utilisée pour stocker et traiter

facilement des téraoctets, voire des pétaoctets de données. Ci-dessous, une liste des avantages clés [164]–[167] qui nous ont amené à choisir HBase :

- La cohérence des données.
- La distribution automatique des données (Auto-sharding).
- L'accès aléatoire aux données (lecture et écriture) même si le HDFS ne supporte pas cette propriété.
- L'évolutivité.
- Le versionnement des données au niveau des cellules.
- Une importante tolérance aux pannes.
- Le support des APIs permettant la programmation des opérations LDD et LMD (Langage de Manipulation de Données) tels que Java API, Thrift, Avro et RESTful.
- Les tables dans HBase ont un schéma flexible contrairement aux tables des bases de données traditionnelles qui ont un schéma fixe et rigide.
- Une intégration facile et souple avec le Framework Hadoop MapReduce, ce qui rendre HBase très adapté au traitement analytique des données.
- Le support des Coprocessors (l'équivalant des déclencheurs pour les bases de données relationnelles).

3. Modélisation d'une base de données NoSQL HBase en utilisant le modèle relationnel

Comme nous l'avons déjà mentionné, une base de données HBase est composée par des lignes et des familles de colonnes. Chaque famille de colonnes contient une ou plusieurs colonnes et les lignes d'une table peuvent avoir un nombre variable de colonnes. Les données sont stockées et versionnées dans des cellules. Une cellule est une combinaison d'une ligne, d'une famille de colonnes et d'un qualificatif de colonne. Pour modéliser une base de données HBase en utilisant le modèle relationnel, nous proposons le modèle présenté dans le Tableau 5.

Tableau 5: Equivalence des concepts du modèle de HBase dans le modèle relationnel

Modèle de HBase	Modèle relationnel
Table	Table

Famille de colonnes	Ensemble de colonnes fortement liées au niveau conceptuel. *p. ex.* : • Informations nominales : nom et prénom. • Informations sur l'adresse : rue, numéro, code postal, localité et pays.
Ligne	Ligne
RowKey	Clé primaire
Colonne	Colonne
Cellule	Valeur d'une colonne
Version	Valeur d'une colonne

4. L'application des opérations de l'algèbre relationnelle sur HBase

Dans cette section nous appliquons les opérations de l'algèbre relationnelle sur le modèle relationnel que nous avons proposé pour une base de données HBase.

4.1. Union

L'union est une opération de base en algèbre relationnelle qui nécessite deux opérandes union-compatibles.

Étant donné deux tables HBase *T1* et *T2*, la définition de l'union est :

$$T1 \cup T2 = \{x | x \in T1 \; ou \; x \in T1\}$$

Où *T1* et *T2* sont union-compatibles.

4.2. Différence

Étant donné deux tables HBase *T1* et *T2*, la définition de la différence est :

$$T1 - T2 = \{x | x \in T1 \; et \; x \notin T2\}$$

Où *T1* et *T2* sont union-compatibles.

4.3. Intersection

L'intersection de deux relations union-compatibles R1 et R2 produit une troisième relation contenant les lignes appartenant conjointement à R1 et à R2.

Dans HBase, nous pouvons définir l'intersection comme suit :

$$T1 \cap T2 = \{r : r \in T1 \; et \; r \in T2\}$$

Où T1 et T2 sont deux Tables HBase et r est une ligne dans ces tables.

4.4. Produit cartésien

Le produit cartésien de deux relations R1 et R2 produit une troisième relation contenante exclusivement toutes les combinaisons possibles d'occurrences des relations R1 et R2, on note R1 × R2.

Le nombre d'occurrences de la relation résultante du produit cartésien est le nombre d'occurrences de R1 multiplié par le nombre d'occurrences de R2.

Dans HBase, nous pouvons définir le produit cartésien comme suit :

$$T1 \times T2 = \{(r,s) : r \in T1 \; et \; s \in T2\}$$

Où *T1* et *T2* sont deux tables HBase, r et s sont des lignes dans ces tables.

4.5. Sélection

La sélection est une opération unaire qui extrait une (ou plusieurs) ligne(s) d'une table HBase si la condition de sélection *P* est remplie.

- Notation : $\sigma_P (T)$
- Paramètres : *T* est une table HBase et *P* est une formule formée d'une combinaison des opérateurs de comparaisons et les opérateurs logiques.
- Résultat : $\sigma_P (T) = \{r \in T : \text{r remplit la condition } P\}$

4.6. Projection

La projection d'une table HBase *T1* produit une table HBase *T2* en sélectionnant les lignes avec les colonnes de l'ensemble C et en éliminant les lignes en double.

- Notation : $\pi_{c_1 \ldots . c_n} T1$

- Paramètres : *T1* est une table HBase et *C* est l'ensemble de colonnes à sélectionner.
- Résultat : une table HBase *T2* avec uniquement les colonnes spécifiées en C.

4.7. Renommage

Le renommage est une opération unaire qui permet de modifie le nom d'une famille de colonnes (ou un qualificatif de colonne). Dans HBase, il n'y a pas une commande pour faire cette opération, mais elle peut être effectuée en utilisant MapReduce ou l'API Java.

- Notation : $\rho_{c_1/c_2}(T)$
- Paramètres : *T* est une table HBase, c_1 est le nouveau nom de la famille de colonnes (ou qualificatif de colonne) et c_2 est une famille de colonnes (ou qualificatif de colonne) dans la table HBase *T*.
- Résultat : Le résultat est une table identique à *T* avec c_2 renommée par c_1.

4.8. θ-jointure

La θ-jointure (thêta-jointure) est une opération binaire qui consiste en toutes les combinaisons de lignes dans deux tables HBase *T1* et *T2* qui répondent à une condition.

- Notation : $T1 \bowtie_{r_1 \theta\ r_2} T2$ OU $T1 \bowtie_{r_1 \theta\ v} T2$
- Paramètres : *T1* et *T2* sont deux tables HBase. r_1 et r_2 sont deux qualificatifs de colonne, θ est un opérateur comparaison qui peut être : >, ≥, =, ≠, < ou ≤, v est une valeur constante.
- Résultat : un sous-ensemble de produit cartésien avec seulement lignes qui remplissent la condition de sélection.

Lorsque le critère de sélection est l'égalité, nous appelons cette opération l'équi-jointure.

4.9. Jointure naturelle

La jointure naturelle est une opération binaire. Elle est le résultat du produit cartésien de deux tables HBase avec une condition qu'il y ait au moins une colonne commune avec le même nom et la même valeur. Si cette condition est omise, et que les deux tables HBase n'ont pas de colonnes communes, la jointure naturelle devient simplement le produit cartésien. Elle est définie comme suit :

$$T1 \bowtie T2 = \pi_{C \cup D}\ \sigma_{T1.a_1 = T2.a_1 \wedge T1.a_2 = T2.a_2 \wedge \ldots \wedge T1.a_n = T2.a_n}\ (T1 \times T2)$$

Où *C* est l'ensemble des noms de colonnes de la table HBase *T1* et *D* est l'ensemble des noms de colonnes de la table HBase *T2*.

4.10. Division

La division est une opération binaire. c'est une opération très puissante et utile, elle est écrite comme suit :

$$T1(k) \div T2(c)$$

Où *T1* et *T2* sont deux tables HBase, *k* et *c* sont les ensembles de noms de colonnes de ces tables :

$$k = \{k_1, \dots, k_m, c_1, \dots, c_n\},$$

$$c = \{c_1, \dots, c_n\}$$

Où $c \subset k$

Le résultat de cette opération est constitué de toutes les lignes *r(x)* de *T1* qui apparaissent dans T1 en combinaison avec chaque ligne de *T2*, où $x = k - c$

La division elle peut être définie comme suit :

$$T1(k) \div T2(c) = \{t \mid t \in \pi\, k - c\,(T1)\; et \;\forall\, u \in T2\,(t \times u \in T1)\}$$

5. L'implémentation des opérations de l'algèbre relationnelle dans HBase

Dans cette section, nous allons citer les implémentations de quelques opérations clés de l'algèbre relationnelle dans HBase.

5.1. Scan

Scan est une commande HBase permettant de lire les données à partir d'une table HBase. La commande *Scan* récupère zéro ou plusieurs lignes d'une table. Par défaut, *Scan* lit la table entière du début à la fin, mais elle peut être utilisées avec des filtres personnalisés pour renvoyer un sous-ensemble de résultats au client. C'est l'équivalent de l'opération de sélection dans l'algèbre relationnelle. Sa syntaxe est la suivante :

$$scan\ '< nom\ de\ la\ table >, \{\ Paramètres\ optionnels\ \}$$

Plusieurs spécifications optionnelles peuvent être passées en paramètre à cette commande pour obtenir des résultats personnalisés. Comme par exemple *COLUMNS*, *FILTER*, *LIMIT*, *STARTROW* et *STOPROW*.

Cependant, l'utilisation de la commande *Scan* via la ligne de commande ou l'API HBase souffre de problèmes de performance surtout pour les données de grande taille. Cela tient au fait que cette commande lit les données d'une manière séquentielle et elle n'utilise pas tous les Serveur de Régions en même temps, ce qui affecte les performances. Pour contourner ce problème, le Framework Hadoop MapReduce peut être utilisé pour implémenter des *Scan* parallèles et distribuées. La Figure 46 présente l'algorithme de la commande *Scan* distribuées en utilisant MapReduce.

Algorithm 7: Distributed HBase Scan command in MapReduce

```
1  class Driver {
2      methode main(){
3          scan ← HBaseTable.scan()
4          foreach columnFamilyName ∈ selectedColumnFamilyNames do
5              foreach columnName ∈ selectedColumnNames do
6                  scan.addColumn(columnFamilyName, columnName)
7              end foreach
8          end foreach
9          foreach filterOption ∈ usedFilterOptions do
10             scan.setFilter(filterOption)
11         end foreach
12         jobStart
13     }
14 }

15 class Mapper {
16     methode map(key, filteredRow){
17         Emit(key, filteredRow)
18     }
19 }
```

Figure 46: Algorithme de la commande Scan distribuées en MapReduce

Bien que l'implémentation de la fonction *Map* et la fonction *Reduce* soit tout ce dont nous avons besoin pour effectuer un « job » MapReduce, il y a un autre morceau de code nécessaire dans MapReduce appelé *Driver*. Ceci communique avec le Framework Hadoop et spécifie les éléments de configuration requis pour exécuter un « job » MapReduce. Dans la classe *Driver* de notre algorithme, nous avons utilisé la fonction *addColumn* de la commande *Scan* (Ligne 6) permettant de choisir les colonnes à afficher dans le résultat, et pour filtrer les lignes, nous avons utilisé la fonction *setFilter* (ligne 10). Un filtre peut être considéré comme la clause *Where* en SQL.

Après l'exécution de la commande *Scan* avec les filtres, le résultat est envoyé au *Mapper*. De cette manière, le nombre de données envoyées au *Mapper* est minimisé par rapport au cas où le filtrage des données est effectué dans la phase *Map*. De plus, le traitement

de la commande *Scan* avec les filtres sera parallélisé. Le *Mapper* prend en entrée un enregistrement sous forme d'une paire (*key*, *filteredRow*) avec :

- *filteredRow* : est une ligne de jeu de données sélectionné par la commande *Scan* avec les filtres.
- *key* : est le rowKey de la ligne traitée.

Le *Mapper* dans notre algorithme permet simplement de recevoir les données et les afficher ensuite ou les stocker dans une autre table HBase (ligne 17). Ainsi, pour une commande *Scan*, nous n'avons besoin que de la phase *Map* pour obtenir le résultat de la requête. Il n'y a pas d'opération pour la phase *Reduce*.

5.2. Get

La commande "*get*" est utilisée pour lire les données d'une table HBase. Cette commande renvoie une seule ligne selon le paramètre d'identification de la ligne (*RowKey*). C'est l'équivalent de la commande *SELECT* avec la clause *WHERE* en SQL. Une commande "*get*" est simplement une commande *Scan* limitée par l'API à une seul ligne. Sa syntaxe est la suivante:

$$get\ '<nom\ de\ la\ table>','<RowKey>'$$

En utilisant cette commande, nous pouvons lire les données d'une table HBase, mais elle retourne seulement l'enregistrement où son *RowKey* égale au second paramètre de la commande.

Nous pouvons également spécifier les colonnes apparaissant dans le résultat en utilisant la syntaxe suivante :

$$get\ '<nom\ de\ la\ table>','<RowKey>',\{COLUMN\ \Rightarrow\ '<famille\ de\ colonnes>:<qualificatif\ de\ colonnes>'\}$$

Quant à l'implémentation de la commande "*get*" en MapReduce pour bénéficier de traitement parallèle et distribué, elle rassemble à l'algorithme de la Figure 46, à l'exception de la partie de filtrage (ligne 10), un *rowKey* doit obligatoirement être renseigné.

5.3. Group by

L'opération *Group by* est souvent utilisée dans les fonctions d'agrégation, elle permet de grouper des lignes selon une(des) colonne(s) et d'appliquer une fonction d'agrégation pour chaque groupe.

Par défaut, HBase ne prend pas en charge l'opération *Group by* et les fonctions d'agrégation, mais elles peuvent être effectuées en utilisant le Framework Hadoop MapReduce.

Dans la phase "*Shuffle and Sort*" [168], MapReduce effectue un tri et un groupement par clé pour s'assurer que le paramètre d'entrée d'un *Reducer* est un ensemble de tuples *t* :

$$t = (k, [v])$$

où [*v*] est la collection de toutes les valeurs associées à la clé *k*. Dans un *Reducer*, les fonctions d'agrégation peuvent être implémentées et appliquées sur la liste des valeurs groupées.

Dans ce qui suit nous allons voir comment HBase implémente l'opération *Group by* et les fonctions d'agrégation par le biais du Framework Hadoop MapReduce.

5.4. Fonctions d'agrégation

Une fonction d'agrégation est un calcul mathématique impliquant un ensemble de valeurs qui aboutit à une valeur unique exprimant une information significative concernant les données à partir desquelles elle est calculée. Les fonctions d'agrégation peuvent être appliquées sur toutes les lignes sélectionnées ou sur chaque groupe de lignes. HBase ne prend pas en charge les fonctions d'agrégation, mais nous pouvons les mettre en œuvre en utilisant le Framework Hadoop MapReduce ou les HBase Coprocessors. Les fonctions d'agrégation communes sont : COUNT, SUM, AVG, MAX et MIN. Nous pouvons présenter ces fonctions comme suit :

$$G_1, \dots, G_k \, \mathrm{g}_{F_1(C_1),\dots F_n(C_n)}(T)$$

Où *G* est un ensemble de colonnes de groupement, *C* est un ensemble de qualificatifs de colonne de la table HBase *T*. Chaque fonction d'agrégation *F* sera appliquée pour chaque groupe de l'ensemble *G*.

Dans HBase, ces fonctions peuvent être mise en œuvre à l'aide du Framework Hadoop MapReduce ou les HBase Coprocessors. Dans ce chapitre nous allons présenter comment le Framework Hadoop MapReduce implémente les fonctions COUNT, SUM et AVG pour être utilisées et appliquées sur les données stockées dans HBase.

5.4.1. COUNT

La fonction HBase COUNT renvoie le nombre de lignes d'une table. Cependant, cette commande est très lente, car elle fonctionne de manière séquentielle. De plus, elle ne supporte pas les filtres pour compter le nombre des lignes qui correspondent à un critère spécifique, et elle ne peut pas être appliquée à un groupe de lignes. Pour surmonter ces limites, nous avons utilisé MapReduce pour implémenter une fonction similaire de la fonction COUNT de langage SQL (cf. Figure 47).

Algorithm 8: count function in MapReduce

```
1  class Driver {
2      methode main(){
3          scan ← HBaseTable.scan()
4          foreach filterOption ∈ usedFilterOptions do
5              scan.setFilter(filterOption)
6          end foreach
7          jobStart
8      }
9  }

10 class Mapper {
11     methode map(key, filteredRow){
12         Emit(groupingColumn, 1)
13     }
14 }

15 class Reducer {
16     methode reduce(groupingColumn, integers){
17         count ← 0
18         foreach one ∈ integers do
19             count ← count + one
20         end foreach
21         Emit(groupingColumn, count)
22     }
23 }
```

Figure 47: Algorithme de la fonction COUNT en MapReduce

La classe *Driver* est chargée de scanner la table HBase (ligne 3), sélectionner des lignes selon des critères spécifiques (lignes 4-6), configurer et exécuter le « job » MapReduce (ligne 7).

La fonction *map* de la classe *Mapper* prend en entrée un enregistrement sous forme d'une paire (*key*, *filteredRow*) avec :

- *filteredRow* : est une ligne de jeu de données sélectionné par la commande Scan avec les filtres.
- *key* : est le rowKey de la ligne traitée.

Le *Mapper* produit une liste de paires (clé, valeur) [(k1 ; v1)], et avant d'être envoyées à la classe Réducteur, elle est automatiquement triée par clé par Hadoop dans la phase « *shuffle* & *sort* ». Pour assurer que le groupement dans la phase « *shuffle* & *sort* » sera effectué selon une colonne spécifique, il faut mettre cette colonne comme clé de la paire envoyée par le *Mapper* au *Reducer* (ligne 12).

Le *Reducer* prend en entrée une liste de paires (clé, valeurs) (k1 ; [v1,v1,....]). Chaque clé représente un groupe. Tout d'abord, le *Reducer* calcule la somme des valeurs associées pour chaque clé (lignes 17-20), et génère en suite une paire (clé, valeur) [(k2 ; v2)], composée de la clé unique et du total obtenu (ligne 21).

5.4.2. SUM

Cette fonction renvoie la somme totale des valeurs d'une colonne numérique d'une table HBase.

Algorithm 9: sum function in MapReduce

```
1  class Driver {
2      methode main(){
3          scan ← HBaseTable.scan()
4          foreach filterOption ∈ usedFilterOptions do
5              scan.setFilter(filterOption)
6          end foreach
7          jobStart
8      }
9  }

10 class Mapper {
11     methode map(key, filteredRow){
12         Emit(groupingColumn, columnToSum)
13     }
14 }

15 class Reducer {
16     methode reduce(groupingColumn, values){
17         sum ← 0
18         foreach value ∈ values do
19             sum ← sum + value
20         end foreach
21         Emit(groupingColumn, sum)
22     }
23 }
```

Figure 48: Algorithme de la fonction SUM en MapReduce

La Figure 48 présente l'algorithme de la fonction *SUM* en MapReduce qui a le même principe de la fonction *COUNT*. La seule différence se situe au niveau de *Mapper*, qui associe à chaque clé intermédiaire (i.e. colonne de groupement) les valeurs qu'on envisage calculer leur somme (ligne 12).

5.4.3. Avg

La fonction *AVG* permet de calculer la moyenne des valeurs d'une colonne numérique dans une table HBase.

L'implémentation de cette fonction dans *MapReduce* (cf. Figure 49) a la même classe *Driver* de la fonction *SUM* et *COUNT*. Dans la classe *Mapper*, la méthode *map* envoie une série de paires (*groupingColumn, columnToAvg*) (ligne 12) avec :

- *groupingColumn* : est la valeur de la colonne de groupement.
- *columnToAvg* : est la valeur de la colonne qu'on envisage calculer leur moyenne.

Ensuite, dans la phase « *shuffle & sort* », Hadoop effectue le tri et le groupement des données par clé, puis ces données sont envoyées au *Reducer* qui additionne les valeurs associées à chaque clé et calcule la moyenne en divisant la somme par le nombre des valeurs (lignes 17 -23).

Algorithm 10: avg function in MapReduce

```
1  class Driver {
2      methode main(){
3          scan ← HBaseTable.scan()
4          foreach filterOption ∈ usedFilterOptions do
5              scan.setFilter(filterOption)
6          end foreach
7          jobStart
8      }
9  }

10 class Mapper {
11     methode map(key, filteredRow){
12         Emit(groupingColumn, columnToAvg)
13     }
14 }

15 class Reducer {
16     methode reduce(groupingColumn, values){
17         sum ← 0
18         count ← 0
19         foreach value ∈ values do
20             sum ← sum + value
21             count ← count + 1
22         end foreach
23         avg ← sum / count
24         Emit(groupingColumn, avg)
25     }
26 }
```

Figure 49: Algorithme de la fonction AVG en MapReduce

Il convient de noter que s'il existe plusieurs colonnes de groupement, la clé dans la phase *Map* est alors la liste des valeurs de tous ces colonnes appartenant à la même ligne. Dans la même veine, s'il y a plusieurs fonctions d'agrégation, la fonction *Reduce* applique chacun d'eux à la liste des valeurs associées à une clé donnée et produit une paire composée de la clé, suivi des résultats de chacune des fonctions d'agrégation.

5.5. Jointure

La jointure dans le modèle relationnel permet d'associer plusieurs tables dans une même requête. Elle permet d'exploiter la puissance des bases de données relationnelles pour obtenir des résultats qui combinent efficacement les données de plusieurs tables. HBase ne prend pas en charge nativement la jointure, mais elle peut être mise en œuvre en utilisant les outils développés sur Hadoop comme Pig [169], Hive [170] et Impala [171] qui exécutent implicitement des « job » *MapReduce* pour établir une jointure entre des tables HBase.

Les algorithmes de jointure utilisés par MapReduce sont différents de celles des SGBD conventionnels, car l'exécution de jointure dans *MapReduce* utilise les fonctions *Map* et *Reduce* pour obtenir des résultats. Il existe plusieurs algorithmes de traitement de jointure

dans *MapReduce* comme Standard Repartition Join [172], Broadcast Join [172] et Trojan Join [173]. La Figure 50 présente le pseudo-code de l'algorithme Standard Repartition Join, l'un des algorithmes de jointure les plus utilisés en Mapreduce [174].

Algorithm 11: Repartition Join algorithm in MapReduce

```
1  methode map(K: null, V : a record from a split of either R or L){
2  |  join_key ← extract the join column from V
3  |  tagged_record ← add a tag of either R or L to V
4  |  emit (join_key, tagged_record)
5  }

6  methode reduce( K': a join key, LIST_V': records from R and L with join key K'){
7  |  create buffers B_R and B_L for R and L, respectively
8  |  append t to one of the buffers according to its tag
9  |  foreach record t in LIST_V' do
10 |  |  append t to one of the buffers according to its tag
11 |  end foreach
12 |  foreach each pair of records (r, l) in B_R × B_L do
13 |  |  emit (null, new_record(r, l))
14 |  end foreach
15 }
```

Figure 50: Algorithme de Repartition Join [172]

Pour établir la jointure entre deux tables HBase *T1* et *T2*, chaque *Mapper* reçoit en entrée une ligne de *T1* ou *T2*. La première étape de la phase *Map* consiste à extraire la colonne de jointure de la ligne, puis ajouter une annotation (tag) pour chaque ligne afin d'identifier sa table d'origine dans la phase *Reduce*, et à la fin émettre les paires (*k, v*) où *k* est la clé de jointure et *v* est la ligne annotée.

Ensuite, le Framework Hadoop MapReduce se charge des tâches de partitionnement, de tri et de fusion (*Shuffle & Sort*). Dans cette étape, Hadoop MapReduce trie par clé et envoie toutes les lignes avec la même clé de jointure au même *Reducer*.

Pour la phase *Reduce*, le paramètre d'entrée est une paire de (k_1; [r_1, r_2,....]) où k_1 est la clé de jointure et [r_1, r_2,....] est une liste de lignes annotées et associées à la clé k_1. Dans cette phase les lignes associées à chaque clé de jointure sont séparées et mémorisées dans deux listes de données selon le tag de la table d'origine (*T1* ou *T2*) et pour établir la jointure, le produit cartésien est effectué entre les lignes des deux listes de données.

6. Conclusion

La migration d'un type de base de données vers un autre type nécessite d'évaluer la faisabilité et les performances potentielles des nouveaux systèmes. Dans ce chapitre, l'accent

est porté sur l'étude de faisabilité de la migration des bases de données relationnelles vers les bases de données orientées colonnes (HBase). Pour cela, nous avons proposé un modèle pour une base de données NoSQL HBase en utilisant le modèle relationnel. Ensuite, nous avons appliqué les opérations de base de l'algèbre relationnelle sur le modèle proposé pour une base de données HBase. Après nous avons cité les implémentations de quelques opérations clés de l'algèbre relationnelle dans HBase. En conclusion, compte tenu de ce qui précède, nous avons déduit que HBase a des capacités équivalentes aux bases de données relationnelles au niveau de la manipulation et l'interrogation des données, ce qui nous permet de dire que la migration entre une base de données relationnelle et une base de données NoSQL HBase peut théoriquement être gérée efficacement.

Chapitre V

Etude Comparative des Performances du SGBD Relationnel MySQL et le SGBD NoSQL HBase

1. Introduction

Depuis plus de trois décennies, les bases de données relationnelles sont la norme de facto sur le marché des systèmes de gestion de bases de données grâce à leur maturité [15], [175]. Aujourd'hui, avec une croissance constante des données générées par les applications web modernes telles que les réseaux sociaux, les sites de commerce électronique et les applications mobiles, alors, la gestion, l'interrogation et l'analyse des données sont devenues un véritable défi pour les systèmes de gestion de bases de données relationnelles (SGBDR). En outre, ces données sont générées sous plusieurs formats (structurée, semi-structurée et non structurée), alors que les systèmes de gestion de bases de données traditionnels sont basés sur un schéma rigide. Ces limites du modèle relationnel ont conduit les leaders de l'Internet tels que Google, Amazon, eBay, Alibaba et Facebook à développer un nouveau modèle appelé bases de données NoSQL [27], afin de surmonter la faiblesse des systèmes de gestion de bases de données relationnelles face à la variété, la vélocité et le grand volume de nouvelles données capturées. Généralement, les bases de données NoSQL ne sont pas un remplacement, mais plutôt un complément aux SGBDR et au SQL. Le modèle NoSQL est basé sur le modèle BASE (*Basically Available*, *Soft State* and *Eventually Consistent*) par opposition au SGBDRs qui sont basés sur les propriétés ACID (*Atomicity*, *Consistency*, *Isolation and Durability*).

Les bases de données NoSQL peuvent être classés en quatre catégories : bases de données clé-valeur, bases de données orientées documents, bases de données orientées colonnes et bases de données orientées graphe. Cette classification est due au fait que chaque type de bases de données se présente dans un contexte spécifique et basé sur des architectures différentes [77]. La comparaison de différents modèles permet d'avoir une vision claire pour choisir le modèle le plus approprié à un contexte donné. L'objectif de cette contribution est de comparer les performances de modèle relationnel (MySQL) et le modèle NoSQL (HBase) et révéler comment le nombre d'opérations et le nombre d'enregistrements affectent la performance en termes de latence et en termes de temps d'exécution en utilisant le Framework YCSB. Afin d'élaborer une approche efficace pour la migration d’une base de données relationnelle vers une base de données HBase, nous avons commencé par une étude de faisabilité dans le chapitre précèdent, ensuite, dans ce chapitre, nous allons établir une comparaison expérimentale des performances entre la base de données relationnelle MySQL et la base de données NoSQL HBase. Le but de cette comparaison est d'identifier clairement les points forts et les points faibles de chaque modèle pour les prendre en considération dans une éventuelle migration. De plus, il est important de comprendre quel type de base de

données est le plus adapté et le plus efficace pour une telle application. Par conséquent, nous pensons qu'il est nécessaire de comparer le modèle relationnel (MySQL) et non relationnel (HBase) en se basant sur la capacité de traitement de différents types de requêtes.

2. Préliminaires

2.1. Benchmarks pour les systèmes de gestion de bases de données

Un benchmark est un banc d'essai qui permet d'évaluer les performances par l'expérimentation sur un système réel [176]. C'est un outil important pour les chercheurs, les concepteurs et les praticiens de bases de données. Son rôle est de générer des charges de travail (workloads) spécifiques aux applications et de tester les bases de données afin d'évaluer les différentes métriques de performance et faciliter le processus de comparaison entre les différentes spécifications des bases de données. L'équipe de Han [177] indique que le processus d'évaluation comparative des mégadonnées se compose de cinq étapes à savoir la planification, la génération de données, la génération de tests, l'exécution et l'analyse, et l'évaluation (cf. Figure 51).

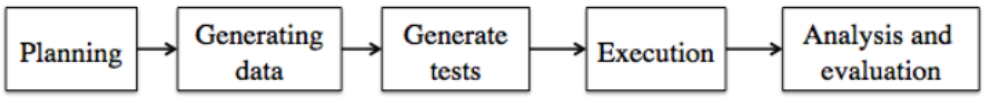


Figure 51: Processus d'évaluation comparative des mégadonnées [177]

Il existe de nombreux outils pour le benchmark des systèmes de bases de données [178] comme BigBench [179], TPC-C[6], TPC-E[7], TPC-H[8], TPC-D[9], Bigdatabench [180] et YCSB[181]. Dans cette contribution, nous allons utiliser le Framework YCSB (Yahoo ! Cloud Serving Benchmark) car il est le choix le plus populaire actuellement pour l'analyse comparative des performances des bases de données NoSQL [182].

2.2. YCSB

Pour comparer les performances des bases de données relationnelles et les bases de données NoSQL, nous avons choisi le banc d'essai open source Yahoo ! Cloud Serving

[6] http://www.tpc.org/tpcc

[7] http://www.tpc.org/tpce

[8] http://www.tpc.org/tpch

[9] http://www.tpc.org/tpcds

Benchmark (YCSB) [181]. C'est l'un des outils les plus populaires pour effectuer le benchmarking des bases de données [183]. YCSB a été développé par Yahoo dans le but de mesurer la performance de divers types de systèmes de bases de données tels que les bases de données relationnelles (avec le pilote JDBC), les bases de données NoSQL (HBase, Mongo, Cassandra, Redis, HyperTable, Couchbase, DynamoDB, Accumulo, etc) et autres [184], [185]. YCSB compare la performance des bases de données en utilisant des opérations simples comme l'insertion, la lecture, le mise à jour et le scan. Il fournit un ensemble de charges de travail prédéfinies qui peuvent être modifiées selon les besoins de l'utilisateur, comme le ratio des opérations de lecture/écriture [184], [185]. Il fournit également des paramètres pour configurer le nombre de threads à utiliser, la durée du test et le nombre d'enregistrements à traiter. En outre, le code source de YCSB est flexible pour ajouter de nouvelles charges de travail.

YCSB supporte de nombreuses bases de données comme MySQL, PostgreSQL, HBase, Cassandra, MongoDB et bien d'autres, pour lesquelles le module Client est déjà implémenté et disponible. Il est également facile d'ajouter des interfaces pour de nouvelles bases de données par une simple implémentation de quelques classes abstraites [184]–[186].

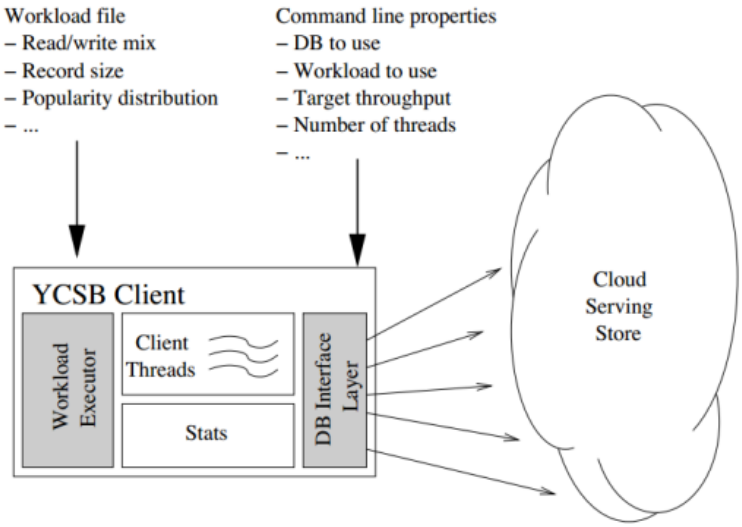


Figure 52: Architecture de YCSB[181]

YCSB est composé d'un générateur des données et un ensemble des charges de travail (cf. Figure 52) pour évaluer la performance des opérations de base (Insertion, lecture, mise à jour et scan). Le benchmark YCSB varie la proportion des opérations de lecture, d'écriture, de mise à jour, d'insertion et de scan dans une série de requêtes appelées charges de travail. Le

benchmark YCSB fournit six charges de travail prédéfinis. Néanmoins l'utilisateur peut définir d'autres charges de travail personnalisées selon le besoin. Ci-dessous la liste des charges de travail prédéfinis dans YCSB.

- **Charge de travail A :** 50% lecture et 50% écriture.
- **Charge de travail B :** 95% lecture et 5% écriture.
- **Charge de travail C :** lecture seule.
- **Charge de travail D :** 5% insertion et 95% lecture des enregistrement récemment insérés.
- **Charge de travail E :** 95% scan et 5% insertion.
- **Charge de travail F :** 50% lecture et 50% lecture- mise à jour-enregistrement.

3. Méthodologie de l'expérimentation

De nombreuses études ont été publiées concernant la comparaison des performances des bases de données en utilisant le Framework YCSB [187]–[189]. Abramova *et al*. [187] comparent cinq bases de données NoSQL (Redis, Cassandra, HBase, MongoDB, et OrientDB) en termes de temps d'exécution des opérations de lecture et écriture. Ils affirment que MongoDB, Redis, et OrientDB sont meilleures en lecture et Cassandra et HBase sont optimisées pour l'écriture. Yassien et Desouky [188] comparent MySQL, MongoDB, et HBase en utilisant YCSB dans le but d'étudier l'effet de la variation du nombre d'opérations et de threads en ce qui concerne le temps d'exécution, le débit d'opération et la latence. Les auteurs déclarent que chaque base de données fonctionne au mieux dans des circonstances différentes. Ils recommandent d'utiliser HBase pour les applications qui nécessitent des opérations de mise à jour et d'insertion importantes, MySQL pour les applications qui effectuent principalement des opérations de lecture et MongoDB pour les applications qui requièrent à la fois des performances de lecture et d'écriture. Matallah *et al*. [189] comparent MongoDB et HBase afin d'évaluer le temps d'exécution du chargement des données et de cinq charges de travail. D'après leurs résultats, lors de la lecture, MongoDB a montré de bonnes performances, contrairement à HBase qui a montré de bonnes performances pour les opérations d'écriture.

La latence est le temps de réponse qui peut obtenir un utilisateur lors de l'envoi d'une requête. C'est l'une des mesures essentielles pour évaluer les performances des bases de

données[190]. Dans cette contribution, nous allons comparer MySQL et HBase en termes de temps d'exécution et de latence des opérations les plus utilisées à savoir la lecture et l'écriture[187]. Nous avons proposé trois scénarios d'évaluation. Le premier consiste à évaluer le chargement de données de différentes tailles de bases de données. Le deuxième scénario consiste à augmenter progressivement le nombre d'enregistrements tout en fixant le nombre d'opérations à 10000. Ce scénario nous permet de vérifier la variation des performances en fonction du nombre d'enregistrement de la base de données. Le troisième scénario consiste à augmenter le nombre d'opérations tout en fixant le nombre total d'enregistrements à 1 million. L'objectif de ce scénario est de révéler comment le changement de nombre d'opérations affectent les performances en termes de temps d'exécution et de latence.

4. Environnement de l'expérimentation

Notre étude comparative a été réalisée dans une machine physique avec le système d'exploitation Ubuntu, le benchmark YCSB, Cloudera Hadoop, Cloudera HBase et MySQL. Toutes les spécifications sont récapitulées dans le Tableau 6.

Tableau 6: Caractéristiques de l'environnement de l'expérimentation

CPU	Intel® Xeon(R) CPU E5504 @ 2.00GHz × 8
RAM	16 GB
Disque dur	237 GB SSD
Système d'exploitation	Ubuntu 14.04 (64-bit)
Version de Java	1.8.0
Version de YCSB	0.14
Version de CDH	5.14.1
Version de Cloudera HBase	1.2.0
Version de Cloudera Hadoop	2.6.0
Version de MySQL	5.7.17

L'objectif principal de cette étude est d'évaluer les performances des opérations les plus utilisées à savoir la lecture et l'écriture [187]. Par conséquent, cette comparaison consiste

principalement en trois charges de travail, à savoir A et C qui sont prédéfinis dans le projet YCSB et nous créons une nouvelle charge de travail G pour évaluer les performances de l'opération de l'écriture seule. Cette charge de travail G est proposée par l'équipe de Abramova [191]. Le Tableau 7 présente les charges de travail utilisés dans cette étude. Il convient de noter que afin de réduire l'impact de variation des performances du processeur et des opérations d'entrées/sorties, nous avons exécuté chaque test trois fois avec un redémarrage de l'ordinateur entre les tests. Toutes les valeurs indiquées dans cette étude représentent la valeur moyenne des trois exécutions.

Tableau 7: Charges de travail utilisés

Charges de travail	Opérations
Charge de travail A	50% lecture et 50% écriture
Charge de travail C	100% lecture : lecture seule
Charge de travail G	100% mise à jour : mise à jour seule (écriture seule)

Le dataset utilisé par ce benchmark est généré par le générateur de données qui fait partie du client YCSB. Les enregistrements du dataset sont composés de 10 champs. Chaque champ est rempli par une chaîne de caractère aléatoire de 100 octets, ce qui donne 1 Ko par enregistrement. YCSB_KEY est la clé primaire de chaque enregistrement [181]. Le Tableau 8 illustre la structure du dataset de YCSB.

Tableau 8: Structure du dataset de YCSB

	YCSB_KEY	FIELD1	FIELD2	FIELD3	FIELD4	FIELD5	FIELD6	FIELD7	FIELD8	FIELD9
ligne 1										
ligne 2										
.....										
.....										
ligne N										

5. Résultats expérimentaux

Nous avons effectué trois tests pour évaluer le temps d'exécution et la latence des bases de données MySQL et HBase. Le premier consiste à générer les données et les chargées dans la base de données. Le deuxième test permet d'exécuter les charges de travail A, C et G tout en augmentant le nombre d'enregistrements et fixer le nombre d'opérations à 10000 par

second. Le dernier test permet exécuter les charges de travail sur une base de données d'un million d'enregistrements tout en variant le nombre d'opérations.

5.1. Chargement des données

- **Temps d'exécution (Plus il est faible, mieux c'est) :** Comme illustré dans la Figure 53, HBase a le temps d'exécution le plus faible. De plus, à mesure que la taille des données augmente, le temps de chargement des données pour MySQL et HBase augmente. HBase a connu une augmentation normale, cependant, MySQL montre une augmentation spectaculaire à partir de 100000 enregistrements. Dans le premier test (nombre d'enregistrements = 1000), MySQL et HBase ont un temps d'exécution presque identique. À mesure que le nombre d'enregistrements augmente, le temps de chargement des données de MySQL varie de 2 fois plus lent que celui de HBase pour le deuxième test (nombre d'enregistrements = 10000), à plus de 4 fois plus lent pour le troisième test et à plus de 5 fois pour le quatrième test.

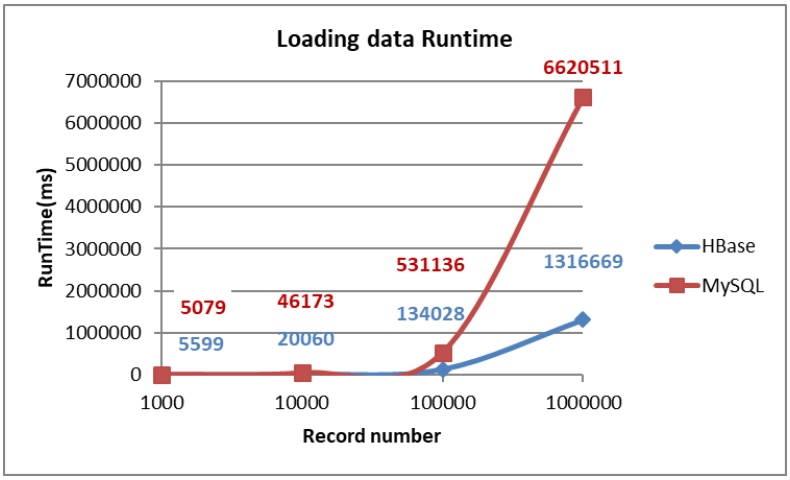


Figure 53: Temps de chargement de données

- **Latence d'insertion (Plus elle est faible, mieux c'est) :** elle diminue à mesure que la taille des données augmente pour HBase. Contrairement à MySQL qui montre une certaine stabilité au début, puis un déclin et une augmentation par la suite. Comme le montre la Figure 54, HBase a la latence d'insertion la plus courte.

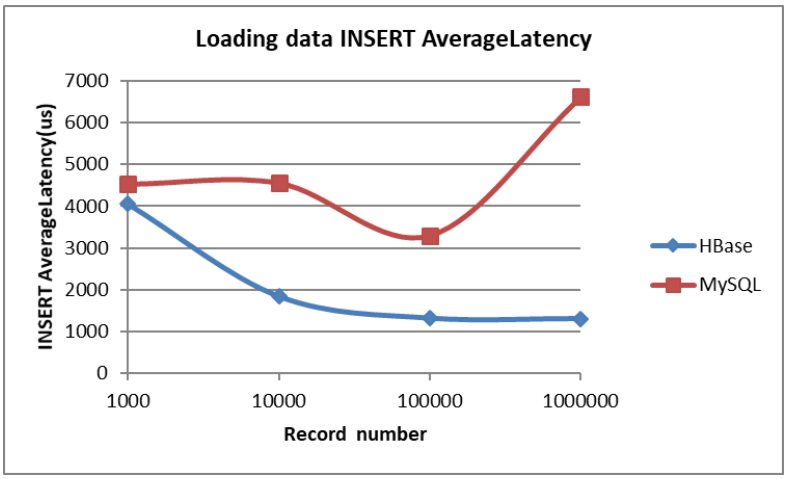


Figure 54: Test 1, Latence Moyenne d'insertion

5.2. Evaluation de la variation des performances en fonction du nombre d'enregistrement de la base de données

5.2.1. Charge de travail A

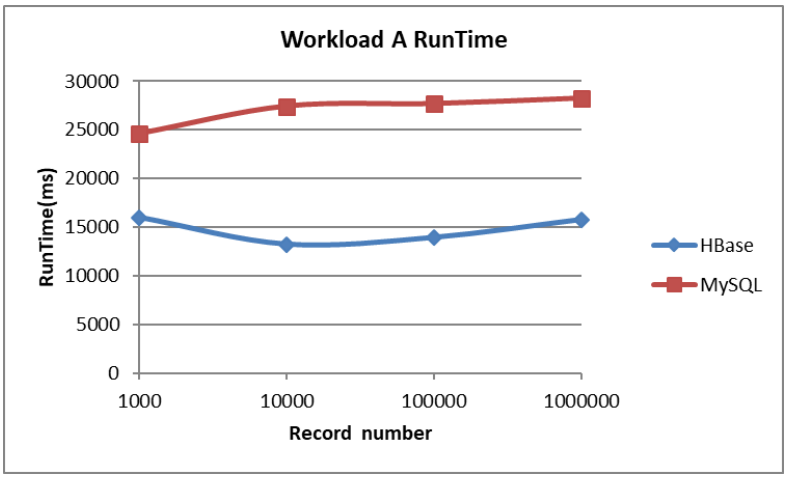


Figure 55: Test 2, Temps d'exécution de charge de travail A

- **Temps d'exécution :** Comme illustré dans la Figure 55, plus la taille des données augmente, plus le temps d'exécution de charge de travail A de MySQL augmente. MySQL affiche une légère augmentation régulière de temps d'exécution, contrairement à HBase qui affiche une légère baisse et une augmentation par la suite. HBase a le temps d'exécution le plus faible.

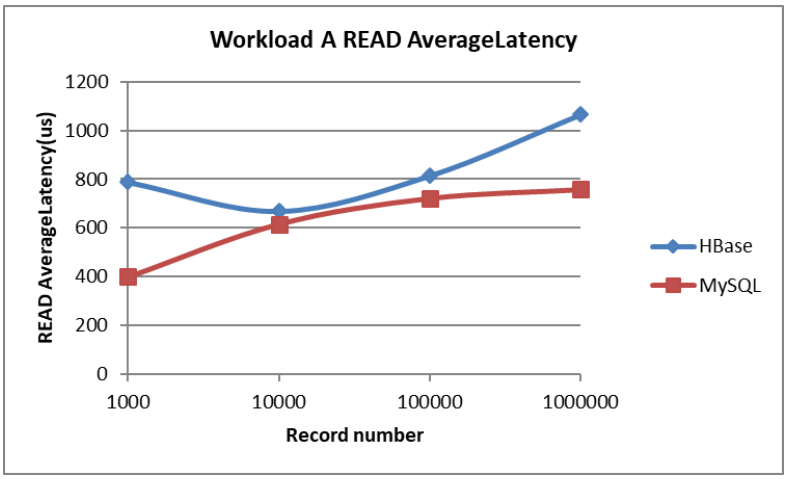


Figure 56: Test 2, Latence de lecture de charge de travail A

- **Latence de lecture (Plus elle est faible, mieux c'est) :** HBase affiche une légère baisse et une augmentation par la suite, contrairement à MySQL qui affiche une légère augmentation régulière comme le montre la Figure 56. MySQL a la latence de lecture la plus courte.

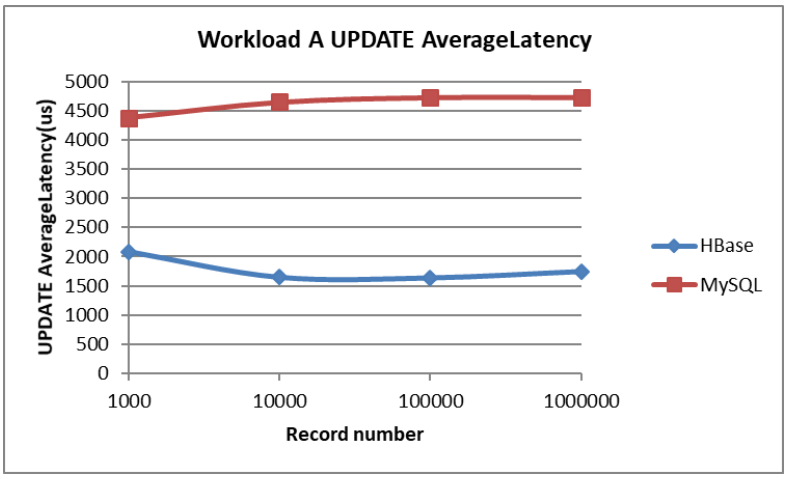


Figure 57: Test 2, Latence de mise à jour de charge de travail A

- **Latence de mise à jour (Plus elle est faible, mieux c'est) :** Comme le montre la Figure 57, MySQL présente une légère augmentation régulière de latence de mise à jour, contrairement à HBase qui présente une légère baisse et une augmentation par la suite. HBase a la latence de mise à jour la plus faible.

5.2.2. Charge de travail C

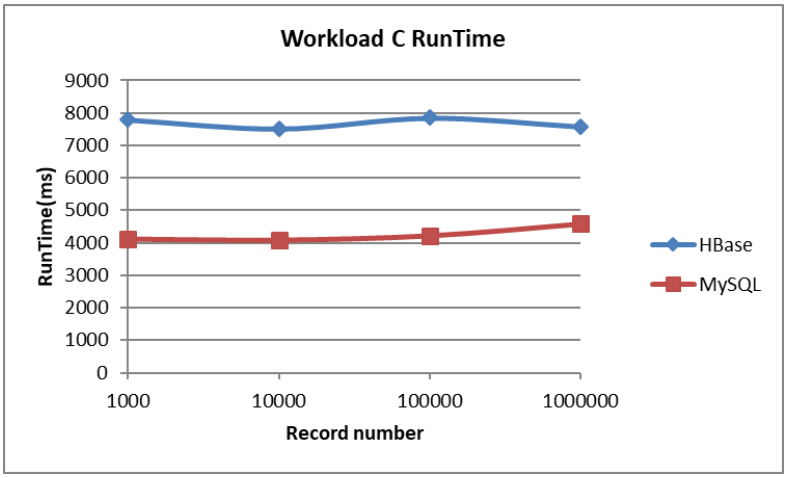


Figure 58: Test 2, Temps d'exécution de charge de travail C

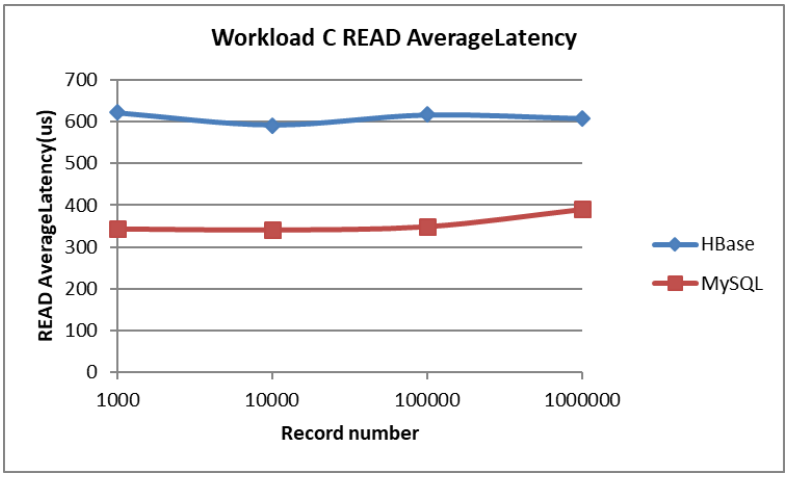


Figure 59: Test 2, Latence de lecture de charge de travail C

Selon les Figures 58 et 59, HBase affiche une légère baisse au départ, puis une augmentation et une baisse par la suite en termes de temps d'exécution et de latence de lecture, contrairement à MySQL qui affiche une stabilité au départ, puis une légère augmentation après avoir atteint 100 000 enregistrements. MySQL a le temps d'exécution et la latence de lecture les plus faibles.

5.2.3. Charge de travail G

Comme illustré dans les Figures 60 et 61, HBase affiche une stabilité pour le temps d'exécution et la latence de mise à jour, contrairement à MySQL qui affiche une légère augmentation. HBase a le temps d'exécution et la latence de mise à jour les plus faibles.

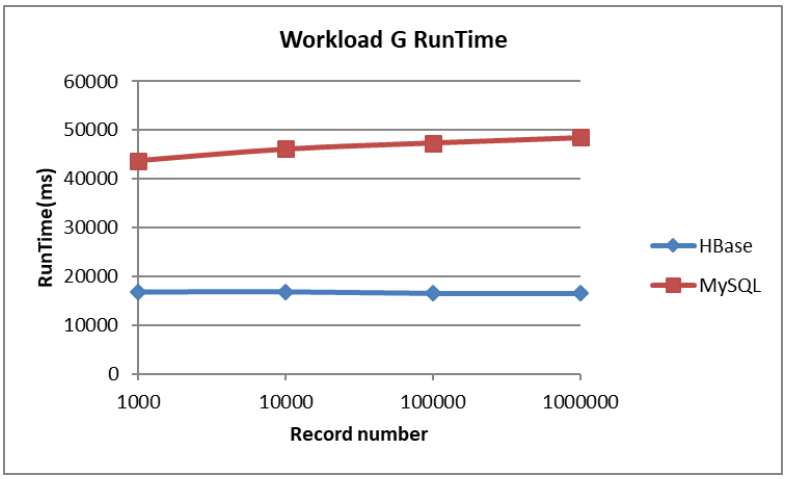


Figure 60: Test 2, Temps d'exécution de charge de travail G

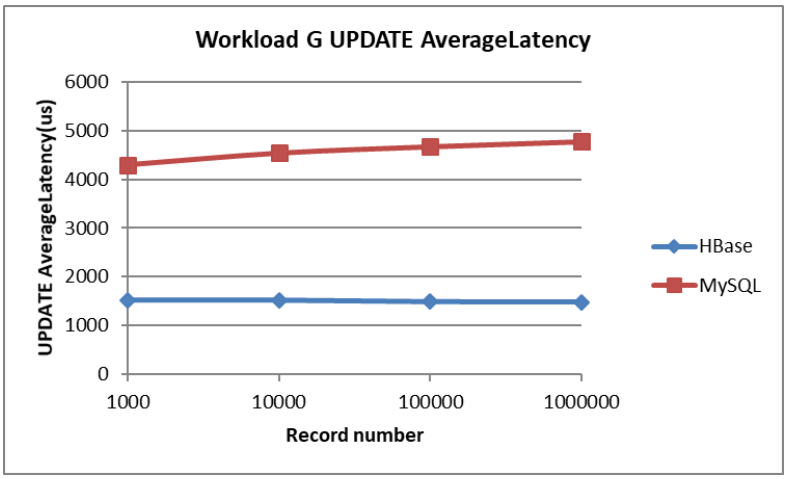


Figure 61: Test 2, Latence de mise à jour de charge de travail G

5.3. Evaluation de la variation des performances en fonction du nombre d'opérations par second

5.3.1. Charge de travail A

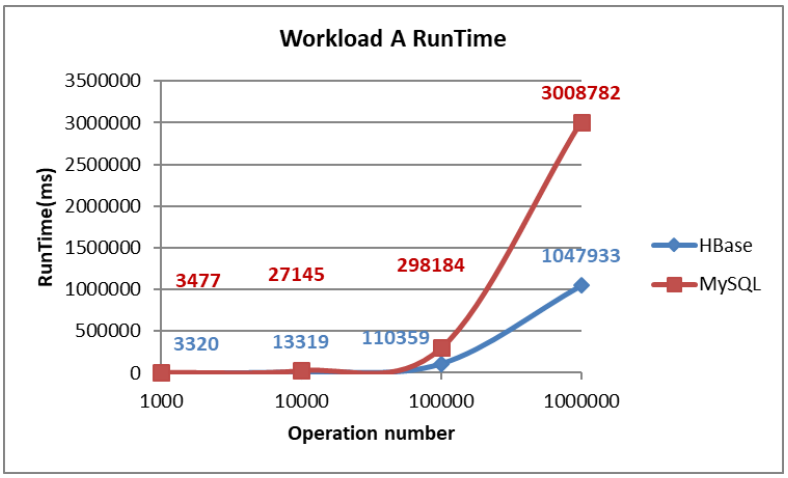


Figure 62: Test 3, Temps d'exécution de charge de travail A

- **Temps d'exécution :** Comme le montre la Figure 62, à mesure que le nombre d'opérations augmente, le temps d'exécution de MySQL et de HBase augmente. HBase affiche une augmentation normale, cependant, MySQL présente une augmentation spectaculaire à partir de 100000 opérations/second. HBase a le temps d'exécution le plus faible.
- **Latence de lecture :** HBase et MySQL affichent un immense déclin jusqu'à atteindre 10000 opérations/second, puis affichent une légère augmentation (cf. Figure 63). MySQL a la latence de lecture la plus faible.

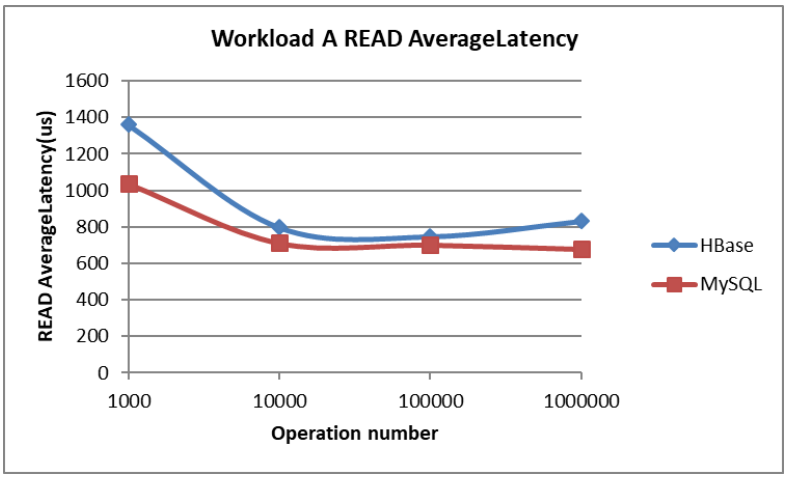


Figure 63: Test 3, Latence de lecture de charge de travail A

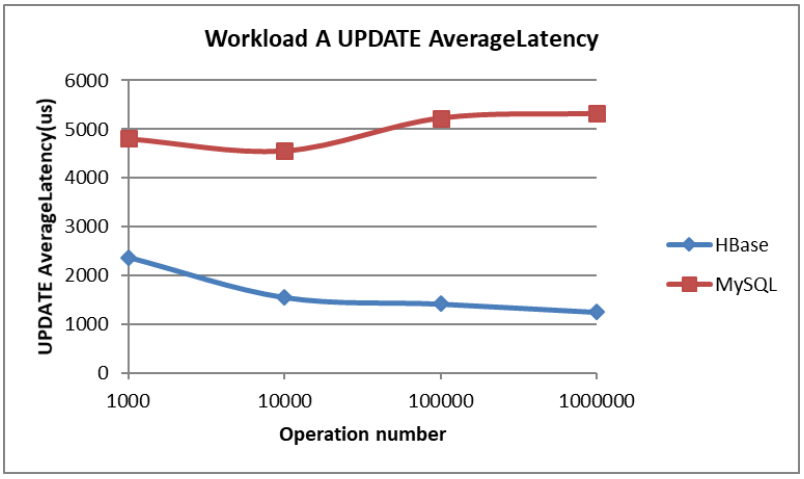


Figure 64: Test 3, Latence de mise à jour de charge de travail A

- **Latence de mise à jour :** Comme illustré dans la Figure 64, à mesure que le nombre d'opérations augmente, la latence de mise à jour de HBase diminue. MySQL affiche un léger déclin au début, puis une augmentation alternée et une stabilité par la suite en termes de latence de mise à jour. HBase a la latence de mise à jour la plus faible.

5.3.2. Charge de travail C

- **Temps d'exécution :** Selon la Figure 65, à mesure que le nombre d'opérations augmente, le temps d'exécution de MySQL et de HBase augmente. MySQL a le temps d'exécution le plus faible.
- **Latence de lecture :** HBase affiche une immense diminution, contrairement à MySQL qui affiche une légère baisse et une stabilité par la suite (cf. Figure 66). MySQL a la latence de lecture la plus faible.

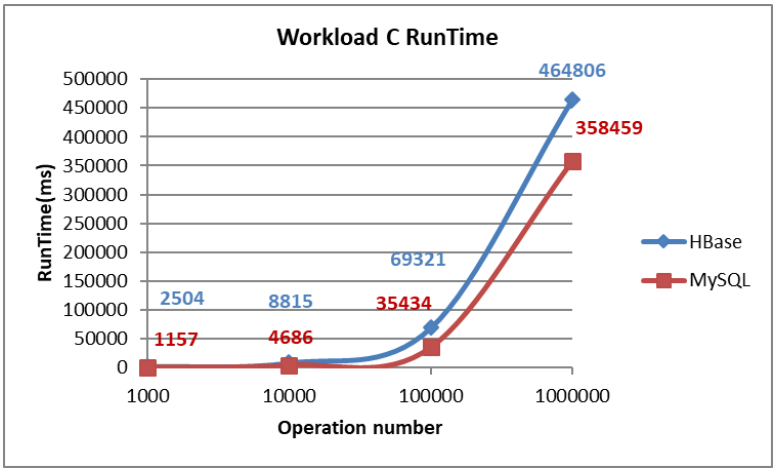


Figure 65: Test 3, Temps d'exécution de charge de travail C

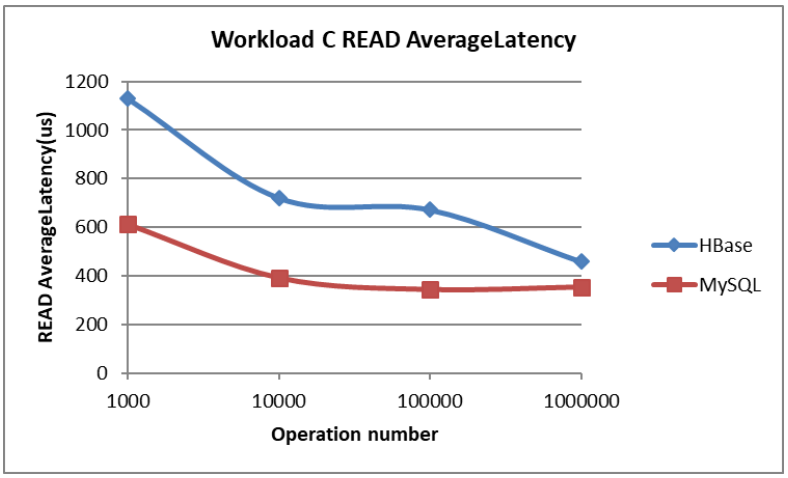


Figure 66: Test 3, Latence de lecture de charge de travail C

5.3.3. Charge de travail G

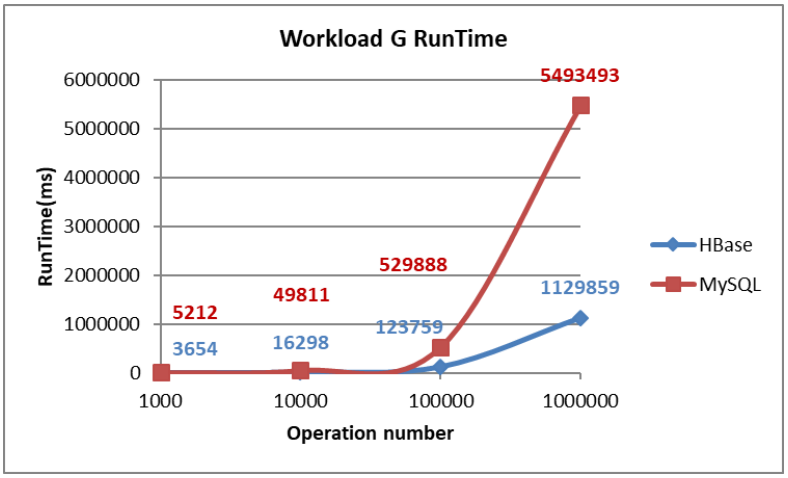


Figure 67: Test 3, Temps d'exécution de charge de travail G

- **Temps d'exécution :** Comme le montre la Figure 67, à mesure que le nombre d'opérations augmente, la durée d'exécution de MySQL et de HBase augmente. HBase affiche une augmentation normale, en revanche MySQL présente une augmentation spectaculaire à partir de 100000 opérations. HBase a le temps d'exécution le plus faible.

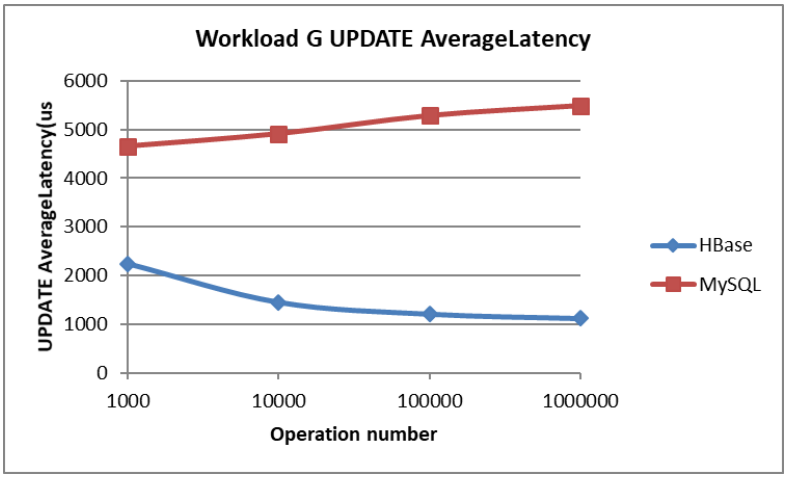


Figure 68: Test 3, Latence de mise à jour de charge de travail G

- **Latence de mise à jour :** HBase montre une diminution régulière, et MySQL affiche une augmentation constante (cf. Figure 68). HBase a la latence de mise à jour la plus faible.

6. Synthèse des résultats

Selon les résultats expérimentaux de notre étude, nous pouvons observer que le temps d'exécution de chargement de données de MySQL est plus élevé pour tous les tests et nous pouvons dire que HBase est bien plus performant que MySQL au niveau de chargement de

données. En ce qui concerne les latences de lecture/écriture, nous pouvons affirmer que la latence de MySQL est plus faible pour les opérations de lecture et que celle de HBase est plus faible pour les opérations d'écriture. En ce qui concerne la durée d'exécution des charges de travail, HBase est le plus performant dans tous les cas, sauf pour la charge de travail en lecture seule. De plus, d'après nos résultats expérimentaux, nous pouvons affirmer que :

- L'augmentation du nombre d'enregistrements a des effets médiocres sur la latence de lecture pour MySQL et HBase. Elle n'a aucune influence en termes de latence d'écriture. Cependant, Elle a un grand impact en ce qui concerne le temps d'exécution de chargement des données (en tenant compte du fait que le nombre d'enregistrements testés n'était pas vraiment important).
- L'augmentation du nombre d'opérations a un impact significatif sur la latence de lecture et d'écriture pour MySQL et HBase, et des effets immenses sur le temps d'exécution des charges de travail.

En générale, les résultats obtenus ont montré que HBase surpasse MySQL pour les opérations d'écriture mais, il est à la traîne pour les opérations de lecture en termes de temps d'exécution et de latence. HBase affiche de bonnes performances concernant le chargement initial des données grâce sa gestion de mémoire qui optimise l'exécution les opérations d'insertion. HBase montre aussi sa performance au niveau des opérations de mise à jour, ceci expliqué par le fait que HBase écrite au début les mises à jour des données dans le journal d'écriture anticipée (WAL). Après la validation et le stockage de la mise à jour dans le journal WAL, elle est écrite dans le MemStore en mémoire tampon. Lorsque les données en mémoire atteignent leur capacité maximale, elles sont écrites sur le disque, ce qui réduit les opérations d'entrée/sortie, contrairement à MySQL qui stocke les données directement sur le disque. Concernant les performances des opérations de lecture, HBase est à la traîne, car il compare toutes les versions enregistrées pour une données afin de renvoyer la version la plus récente, ce qui affecte les performances de la base de données.

Nous citons ci-dessous une liste des cas d'utilisation adaptés pour le SGBD relationnelle MySQL et une autre pour le SGBD NoSQL HBase :

- **MySQL**

- Applications qui favorisent les charges de travail utilisant les opérations de lecture comme par exemple l'étiquetage des photos(tag); l'ajout d'un tag est une opération d'écriture, mais la plupart des opérations consistent à lire les tags.
- Applications qui utilisent des données structurées.
- Applications avec des requêtes complexes et qui ne sont pas destinées pour l'analyse de données en temps réel.
- Applications utilisant des volumes de données modérés.
- Utile pour les applications qui favorisent la Cohérence et la Disponibilité (théorème de Brewer).
- Applications qui nécessitent l'utilisation des traitements des transactions.
- Applications utilisant des données avec une vélocité modérée.
- Applications ayant une seule/quelques source(s) de données.

- **HBase**
  - Destiné pour les contextes favorisant les charges de travail qui sont constitués majoritairement par des opérations d'écriture comme par exemple les applications dans le domaine de l'e-commerce qui peuvent utiliser HBase pour stocker les fichiers log concernant l'historique de recherche des clients, ainsi que pour effectuer des analyses et ensuite cibler la publicité adaptée pour chaque client.
  - Applications qui utilisent les données non-structurées et/ou semi-structurées.
  - Applications utilisant les données massives.
  - Génération des rapports d'activité "Reporting" à partir des données massives.
  - Utile pour les applications favorisant la Cohérence et la Tolérance au partitionnement (théorème de Brewer).
  - Applications qui utilisent des données avec une vélocité élevée comme les données issues des capteurs utilisées dans l'Internet des Objets.
  - Applications ayant plusieurs sources de données.

- Applications Décentralisées basées sur le système Pair-à-Pair comme les applications de vote en ligne décentralisé, les jeux sur le Blockchain et les plateformes de *crowdfunding*.

7. Perspectives

Dans le cadre de comparaison de performances de la base de données relationnelle MySQL et la base de données NoSQL HBase et en perspectives de notre étude, nous proposons ci-dessous d'autres tests pour mieux comprendre les différences entre ces deux types de bases de données.

- **Test 1** : Nous allons fixer le nombre d'enregistrements à 4M (4 000 000) d'enregistrements et nous allons augmenter le nombre d'opérations de 5K (5000), 10K, 100K, 1M à 4 M afin d'évaluer les performances de la base de données lorsque tous les enregistrements pourront être chargés en RAM. Nous allons effectuer cette expérience en mode monoposte et aussi en mode distribué (4 nœuds).
- **Test 2 – (Size-up)** : Dans cette expérience nous allons augmenter la taille de la base de données de 500K (500000), 1M, 3M, 6M à 10M et le nombre d'opérations sera 10% de la taille de la base de données de chaque expérience. L'objectif est d'évaluer les performances de la base de données lorsque des opérations supplémentaires d'entrée/sortie sont utilisées. Nous allons effectuer cette expérience en mode monoposte et aussi en mode distribué (4 nœuds).
- **Test 3 – (Speed-up)** : Afin de tester la première métrique de la scalabilité de MySQL et HBase, nous allons fixer le nombre d'enregistrements à 10M, le nombre d'opérations à 4M et nous allons augmenter progressivement la taille du cluster (1 nœud, 2 nœuds et 4 nœuds) ; La taille de la base de données sera divisée sur les nœuds du cluster.
- **Test 4 – (Scale-up)** : Afin de tester la deuxième métrique de la scalabilité de MySQL et HBase, nous allons fixer le nombre d'enregistrements à 10M par nœud, le nombre d'opérations à 4M et nous allons augmenter progressivement la taille du cluster (1 nœud, 2 nœuds, et 4 nœuds)
- **Test 5** : Le but de ce test est d'évaluer la capacité des bases de données MySQL et HBase à servir un nombre élevé des clients. Dans cette

expérience nous allons augmenter progressivement le nombre de threads (1, 12, 24, 48, 96, 192, 384 et 768) avec un nombre d'opérations de 2M et une base de données de 4M lignes en mode monoposte et aussi en mode distribué (4 nœuds).

Nous avons développé les commandes YCSB nécessaires pour implémenter les tests ci-dessus. Quelques exemples de celles-ci sont reportés dans la section des annexes.

8. Conclusion

Dans cette contribution, nous avons présenté une comparaison expérimentale entre la base de données relationnelle MySQL et la base de données NoSQL HBase en termes de temps d'exécution et de la latence dans différents scénarios en utilisant le Framework YCSB. Selon les résultats expérimentaux de notre étude, nous avons déduit que HBase a obtenu des meilleures performances que MySQL pour le chargement de données, vu que le temps d'exécution de MySQL est plus élevé dans tous les scénarios pour ce type d'opération. En outre, nous avons constaté que HBase surpasse MySQL dans les opérations d’écriture mais il est à la traîne concernant le temps d'exécution et la latence des opérations de lecture.

Chapitre VI

Une Approche de Conversion d’une Base de Données Relationnelle vers une Base de Données NoSQL HBase

1. Introduction

Au cours de la dernière décennie, un écosystème entier de technologies a émergé pour répondre à la demande des entreprises pour traiter une quantité sans précédent de données. À chaque instant, le rythme de création des données ne cesse de s'accélérer. Selon Domo [12],une société américaine spécialisée dans le cloud BI, pendant chaque minute en 2020, 150 milles messages Facebook sont partagés, les utilisateurs de Facebook uploadent 147 milles photos, 500 heures de vidéos YouTube sont uploadées, les utilisateurs de Instagram poste 347222 Stories et 41,66 millions de messages sont partagés sur WhatsApp. Les giga-octets ne sont plus une unité de mesure de volume des données générées. Les téraoctets ne le sont plus non plus. Les volumes de données passent rapidement des pétaoctets aux zettaoctets(cf. Figure 69).

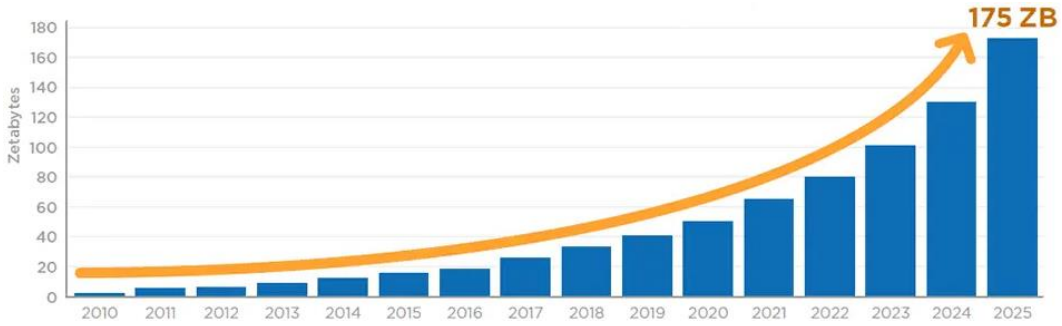


Figure 69: Croissance exponentielle des données [192]

Les systèmes de bases de données relationnelles ont montré leurs limites pour une telle croissance exponentielle des données[193]. Ils stockent les données dans des tables sous une forme normalisée et structurée, ce qui est devenu une limite avec l'évolution rapide des données. Les lacunes des bases de données relationnelles en relation avec les données massives sont traitées par des solutions de Big Data telles que les bases de données NoSQL[194]. Avec l'utilisation croissante de ces dernières, il est important de trouver un moyen pour convertir le schéma d'une base de données relationnelle au schéma d'une base de données NoSQL. Dans cette contribution nous utilisons HBase, qui est l'un des bases de données NoSQL orientées colonnes les plus populaires[195]. Nous allons présenter dans ce chapitre, le processus de migration d'une base de données relationnelle vers HBase avec l'explication de chaque étape de migration, ensuite nous allons proposer un ensemble de règles de transformation de schéma d'une base de données relationnelle vers HBase.

2. Outils de migration

De nombreux outils ont été développés pour migrer les données d'une base de données relationnelle vers une base de données HBase par exemple Apache Sqoop[10] et DataX[11] [196]:

- Apache Sqoop : est un outil développé par Apache Corporation. Il permet de transférer des données en masse à partir d'une base de données relationnelle vers Hadoop et par la suite les importer dans HBase ou Hive. Par conséquent, un utilisateur peut transférer des données et effectuer des opérations MapReduce avec Hadoop facilement. L'outil a été conçu pour fonctionner dans les deux sens afin que les données puissent être transférées d'une base de données relationnelle vers Hadoop et vice versa.
- DataX : est un outil de synchronisation à très grande vitesse des sources de données hétérogènes par exemple MySQL, Oracle, SQL Server, PostgreSQL, Hadoop, HBase, Hive, MongoDB, Cassandra, etc.

L'inconvénient majeur de ces outils réside dans le fait qu'ils ne fournissent pas des solutions pour importer intelligemment dans HBase les relations entre les tables d'une base de données relationnelle. Ce qui oblige l'utilisateur d'interroger plusieurs tables HBase surtout pour les requêtes de jointure, la chose qui affecte les performances des requêtes.

3. Travaux connexes

Serrano *et al.* [197] ont proposé une méthode de migration des schémas de données pour les SGBDR vers HBase. Leur méthode consiste en un ensemble de directives pour encadrer le processus de migration et un ensemble des étapes de transformation des schémas de données que les développeurs d'applications HBase peuvent suivre pendant la migration d'une base de données relationnelle vers HBase. Une autre méthode est conçue par Kim et ses collèges [198] pour transformer les données d'une base de données relationnelle (MySQL) en NoSQL (HBase). Ils ont utilisé une technique de dénormalisation appelée *CLDA* [199] qui permet d'éviter les opérations de jointure et supporte l'atomicité en utilisant les notions de dénormalisation au niveau de la colonne. Cette technique a montré une amélioration considérable des performances des requêtes. L'objectif principal des contributions de Ouanouki [200], [201], est de proposer un ensemble de règles de conversion pour soutenir et

[10] https://sqoop.apache.org/
[11] https://github.com/alibaba/DataX

améliorer le processus de conversion des schémas des bases de données relationnelles vers une base de données orientée colonnes HBase. Tout d'abord, une expérience est menée pour justifier la nécessité de règles de conversion en demandant aux participants de convertir une base de données relationnelle existante vers HBase sans aucune directive. Cette expérience a démontré que tous les participants praticiens des bases de données relationnelles ne pouvaient pas facilement effectuer une telle conversion. Ensuite, une autre expérience a été effectuée pour découvrir une première liste de règles de conversion en demandant aux participants d'effectuer une conversion d'une base de données relationnelle vers HBase en utilisant trois modèles de conception de conversion possibles. Zhao *et al.* [202] ont utilisé les concepts de l'imbrication et l'imbrication multiple pour représenter la relation de jointure entre les tables d'une base de données relationnelle. Comme HBase ne prend en charge l'imbrication qu'entre les familles de colonnes et les qualificatifs des colonnes, il est difficile de traiter une situation de l'imbrication multiple après la migration d'une base de données relationnelle vers HBase tout en assurant des performances élevées des requêtes. Pour remédier ce problème, les auteurs ont proposé un nouveau schéma pour la base de données HBase cible. Ce schéma prend en charge l'imbrication multiple d'une manière efficace après la migration d'une base de données relationnelle vers une base de données HBase. En utilisant ce schéma pour une base de données HBase, une seule requête de sélection peut remplacer les requêtes de jointure utilisées dans la base de données relationnelle source, et l'efficacité de des requêtes a été grandement améliorée, en particulier pour les requêtes de jointure. De plus, cette approche permet aussi la migration des index, ce qui permet une interrogation plus rapide.

4. Conversion d'une base de données relationnelle vers une base de données orientée colonnes HBase

4.1. Définitions formelles

4.1.1. Modèle relationnel

Dans le modèle Entité/Association, une entité représente une table relationnelle. Une table est un ensemble de données organisées sous forme de lignes et de colonnes. Chaque table a une ou plusieurs colonnes, une colonne est un groupe vertical de cellules dans une table. Une ligne est un ensemble de champs qui constituent un enregistrement.

Une clé primaire est une ou plusieurs colonnes qui ont été configurées comme champ d'identification unique pour la table. La plupart des clés primaires sont composées d'une seule colonne, mais elles peuvent également être composées de plusieurs colonnes.

Ainsi, Une clé étrangère est une clé utilisée pour relier deux tables entre elles. C'est un champ (ou une collection de champs) dans une table qui fait référence à la clé primaire dans une autre table.

Ci-dessous une définition formelle d'une table relationnelle :

Une table relationnelle *T* est définie par (N^T, Cln^T, PK^T, FK^T) où :

- N^T : est le nom de la table *T*.
- $Cln^T = \{ Cln_n^T \}, 1 \leq n \leq l$ est l'ensemble des colonnes de la table *T*, où l est le nombre de colonnes de cette table.
- $PK^T = \{ PK_m^T \}, 1 \leq m \leq p$, est l'ensemble des clés primaires de la table *T*, où *p* est le nombre de clés primaires de cette table.
- $FK^T = \{ FK_k^T \}, 1 \leq k \leq s$, est l'ensemble des clés étrangères de la table *T*, où *s* est le nombre de clés étrangères de cette table.
- $(PK^T \cup FK^T) \subset Cln^T$ et $l \leq (p + s)$

4.1.2. Modèle orienté colonnes

Dans le modèle orienté colonnes de HBase, une base de données peut contenir une ou plusieurs tables. Une table HBase est constituée de plusieurs lignes. Chaque ligne a un *RowKey* unique. Les données contenues dans une ligne sont divisées en familles de colonnes. Elles sont composées d'une ou plusieurs colonnes. Ces colonnes sont identifiées par des qualificatifs de colonnes.

Ci-dessous une définition formelle d'une table HBase :

Une table HBase *TH* est définie par (N^{TH}, FC^{TH}) où :

- N^{TH} : est le nom de la table *TH*.
- FC^{TH} : $\{ FC_n^{TH} \}, 1 \leq n \leq l$ est l'ensemble des familles de colonnes de la table *TH*, où l est le nombre de familles de colonnes de cette table. Une famille de colonne FC_n^{TH} est définie par (N^{FC}, Cln^{FC}) où :
  - N^{FC} : est le nom de la famille de colonne.
  - Cln^{FC} : $\{ Cln_m^{FC} \}, 1 \leq m \leq p$ est l'ensemble des colonnes constituant les lignes d'une famille de colonne, où *p* est le nombre de colonnes de cette dernière. Chaque ligne est identifiée par un *RowKey RK*.

4.2. Processus de migration d'une base de données relationnelle vers une base de données orientée colonnes HBase

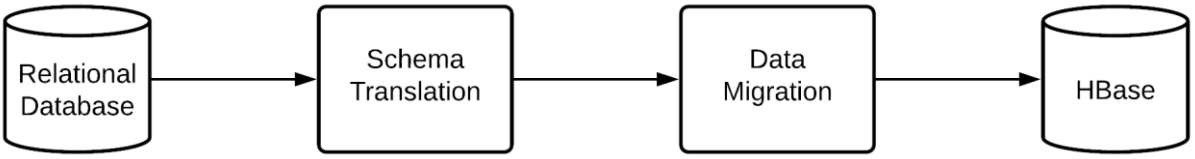


Figure 70: Processus de migration d'une base de données relationnelle vers une base de données orientée colonnes HBase

La Figure 70 présente le processus de migration d'une base de données relationnelle vers une base de données orientée colonnes HBase. Nous proposons deux phases principales comme décrit ci-dessous :

- Transformation de schéma : cette phase consiste à appliquer les règles de transformation de schéma que nous allons présenter dans la section suivante. En premier lieu, nous proposons d'effectuer une analyse des données et des fichiers journaux de la base de données. Cela dit pour bien comprendre la logique de l'application et identifier les méthodes d'accès aux données et les opérations les plus utilisée lors de l'interrogation de la base de données. En deuxième lieu, nous utilisons des règles de conversion des différents types d'associations entre les tables relationnelles, afin de minimiser les relations et les jointures entres les tables dans la base de données cible HBase.
- Migration des données : l'objectif de cette étape est d'effectuer le transfert de données d'une base de données relationnelle vers une base de données HBase en se basant sur le schéma généré dans la première étape.

Ces deux phases peuvent être implémenter en utilisant le Java API, REST API ou Apache Thrift API pour interagir avec HBase.

4.3. Règles de transformation de schéma

Le modèle Entité/Association définit la vue conceptuelle d'une base de données relationnelle. Il représente principalement les entités, les attributs, les relations et les associations entre les entités. Nous proposons dans cette section des règles de transformation d'un schéma relationnel vers une base de données HBase.

4.3.1. Règle 1 : Bien comprendre les données, les méthodes d'accès aux données et le domaine d'activité

La migration des données relationnelles vers une base de données HBase doit être effectuée en tenant compte les méthodes d'accès aux données et de la logique de l'application cible.

Avant de définir le modèle de la base de données HBase, il faut comprendre les données et le domaine d'activité. Au début, il ne faut pas se concentrer sur la manière de transformer ou de définir le modèle de données. Il s'agit plutôt de comprendre les données, indépendamment de la façon dont elles sont stockées dans la base de données relationnelle. Par la suite, il faut identifier comment les données sont manipulées par l'application en utilisant par exemple les fichiers journaux de la base de données relationnelle pour savoir les types d'opérations exécutées par l'application, la façon de sélection des données et les tables fréquemment utilisées dans les jointures. L'objectif de tout ceci est de définir un modèle de la base de données HBase basé sur une compréhension approfondie des charges de travail de l'application.

Table relationnelle Etudiant

id	nom	prenom	image
1	EL Haddad	Ali	haddad.jpg
2	Samiri	Inssaf	samiri.jpg
3	Haj	Salim	haj.jpg
4	Benali	Rim	benali.jpg
5	Karim	Adil	karim.jpg
6	Dada	Fouad	dada.jpg

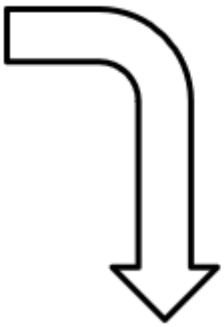


Table HBase Etudiant

rowKey	valeur
1	cf_etudiant: {'cq_nom':'EL Haddad', 'cq_prenom':'Ali', 'cq_image':'haddad.jpg'}
2	cf_etudiant: {'cq_nom':'Samiri', 'cq_prenom':'Inssaf', 'cq_image':'samiri.jpg'}
3	cf_etudiant: {'cq_nom':'Haj', 'cq_prenom':'Salim', 'cq_image':'haj.jpg'}
4	cf_etudiant: {'cq_nom':'Benali ', 'cq_prenom':'Rim', 'cq_image':'benali.jpg'}
5	cf_etudiant: {'cq_nom':'Karim', 'cq_prenom':'Adil', 'cq_image':'karim.jpg'}
6	cf_etudiant: {'cq_nom':'Dada', 'cq_prenom':'Fouad', 'cq_image':'dada.jpg'}

Figure 71: Méthode simple de transformation d'une table relationnelle à une table HBase

4.3.2. Règle 2 : Regrouper dans une famille de colonnes toutes les données accessibles ensemble.

Table relationnelle Etudiant

id	nom	prenom	image
1	EL Haddad	Ali	haddad.jpg
2	Samiri	Inssaf	samiri.jpg
3	Haj	Salim	haj.jpg
4	Benali	Rim	benali.jpg
5	Karim	Adil	karim.jpg
6	Dada	Fouad	dada.jpg

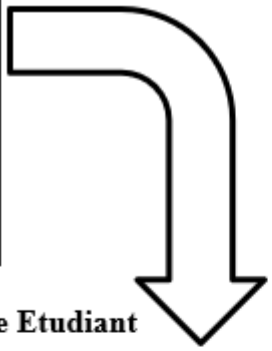


Table HBase Etudiant

rowKey	valeur
1	cf_info_perso: {'cq_nom':'EL Haddad', 'cq_prenom':'Ali'} cf_image: {'cq_image':'haddad.jpg'}
2	cf_etudiant: {'cq_nom':'Samiri', 'cq_prenom':'Inssaf'} cf_image: {'cq_image':'samiri.jpg'}
3	cf_etudiant: {'cq_nom':'Haj', 'cq_prenom':'Salim'} cf_image: {'cq_image':'haj.jpg'}
4	cf_etudiant: {'cq_nom':'Benali ', 'cq_prenom':'Rim'} cf_image: {'cq_image':'benali.jpg'}
5	cf_etudiant: {'cq_nom':'Karim', 'cq_prenom':'Adil'} cf_image: {'cq_image':'karim.jpg'}
6	cf_etudiant: {'cq_nom':'Dada', 'cq_prenom':'Fouad'} cf_image: {'cq_image':'dada.jpg'}

Figure 72: Méthode améliorée de transformation d'une table relationnelle à une table HBase

HBase est une base de données orientée colonnes, elle regroupe plusieurs colonnes de sa chaque table sous forme des ensembles appelés familles de colonnes. En outre, HBase stocke les données sur le disque en fonction de la famille de colonnes, de sorte que la définition appropriée des familles de colonnes peut améliorer l'efficacité des opérations de lecture/l'écriture[152], [203]. Pour cela, les colonnes peuvent être regroupées en fonction du type de contenu des données, c'est-à-dire que les attributs peuvent être divisés en plusieurs

familles de colonnes en fonction de leur importance. Dans la Figure 71, la table relationnelle *Etudiant* stocke toutes les informations d'un étudiant à savoir le nom, le prénom et le nom de son image. Donc un simple schéma HBase peut être dérivé directement du schéma relationnel ; la table *Etudiant* dans la base de données relationnelle correspond à une table HBase porte le même nom avec une seule famille de colonnes et la clé primaire est traitée comme un *RowKey* (cf. Figure 71). Cependant, l'utilisation habituelle des données des étudiants consiste à utiliser seulement le nom et le prénom pour fournir la liste des étudiants d'une classe et l'image ne s'utilise que lorsqu'un utilisateur demande l'affichage les détails d'un étudiant. Alors, pour améliorer l'efficacité des opérations de lecture/l'écriture dans la table HBase *Etudiant*, les données de l'étudiant peuvent être divisées en deux familles de colonnes puisqu'elles ne sont pas accessibles ensemble (cf. Figure 72).

Dans ce qui suit nous proposons des règles de conversion des relations entre les tables relationnelles vers HBase. En effet, le modèle relationnel se base sur l'utilisation d'un schéma normalisé pour assurer l'intégrité des données et éviter la redondance et il utilise les jointures entre les tables pour répondre aux opérations SQL. Tandis que HBase utilise le principe de conception de Dénormalisation, Duplication et Clés intelligentes ou en anglais *DDI* (*Denormalization, Duplication, and Intelligent Keys*) [204]. Tout d'abord, la dénormalisation et la duplication consistent à réagréger les tables relationnelles en une grande table HBase, même si certaines données doivent être dupliquées ou redondantes dans la table HBase. Ensuite, une conception intelligente du RowKey doit être établie pour une identification unique des lignes de la table HBase. Comme ça, les requêtes de jointure seront minimisées puisque les tables relationnelles connexes sont fusionnées en une grande table HBase. Dans ce chapitre nous allons utiliser le principe *DDI* pour convertir les relations entre les tables relationnelles vers une base de données HBase.

4.3.3. Règle 3 : Transformation des relations de type « 1-1 »

Dans la base de données relationnelle de l'application de gestion des ventes, un produit peut avoir des informations supplémentaires facultatives telles que l'image, une description détaillée et un commentaire. Si ces informations facultatives sont enregistrées dans la table « Produits », on obtient donc de nombreuses valeurs nulles pour les enregistrements ne comportant pas ces données facultatives. Ceci peut dégrader les performances de la base de données.

Pour remédier ce problème, nous pouvons créer une autre table nommée «DetailsProduits» pour stocker les données facultatives (cf. Figure 73). Un enregistrement ne sera créé dans cette table que pour les produits comportant des données facultatives. Les deux tables, « Produits » et « DetailsProduits » sont reliées par une relation de type 1-1. C'est-à-dire que pour chaque ligne de la table mère, il y a au maximum une ligne (éventuellement zéro) dans la table enfant.

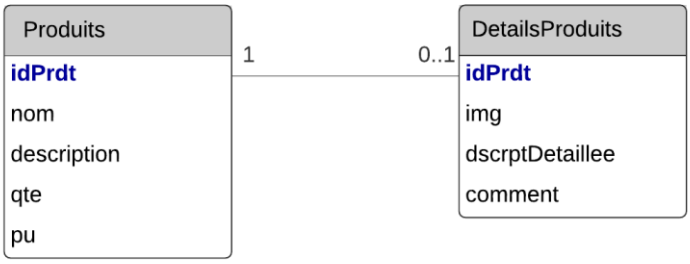


Figure 73: Exemple d'une relation « 1-1 » dans le model relationnel

Table Relationnelle Produits

idPrdt	nom	description	qte	pu
1	chandail	Gros tricot	25	180
2	robes	robes longues	15	250
3	manteaux	manteaux homme	27	450

Table Relationnelle DetailsProduits

idPrdt	img	dscrptDetaillee	comment
3	3.jpg	Cette parka droite mi longue Canada	Pressing...

Table HBase Produits

rowKey	valeur
1	cf_produit : {'cq_nom':'chandail', 'cq_description':'Gros tricot', 'cq_qte':'25', 'cq_pu':'180'}
2	cf_produit : {'cq_nom':'robes', 'cq_description':'robes longues', 'cq_qte':'15', 'cq_pu':'250'}
3	cf_produit : {'cq_nom':'manteaux', 'cq_description':'manteaux homme', 'cq_qte':'27', 'cq_pu':'450'} cf_detailsProduits : {'cq_img':'3.jpg, 'cq_dscrptDetaillee':'Cette parka droite mi longue Canada....', 'cq_comment':' Pressing...'}

Figure 74: Transformation d'une relation de type « 1-1 »

Pour convertir le schéma relationnel présenté dans la Figure 73, une seule table HBase sera créée avec le nom de la table mère « Produits », puis nous allons créer une famille de

colonnes pour chaque table. La clé primaire de la table mère est convertie comme un *RowKey*, et les colonnes de chaque table relationnelle sont converties en qualificatifs de colonnes dans la famille de colonnes correspondante (cf. Figure 74). L'objectif principal de cette conversion est de minimiser le maximum les liens entre les tables HBase puisque HBase ne supporte pas nativement les jointures et les clés étrangères.

La Figure 74 présente un exemple d'application de notre règle de transformation de d'une relation de type « 1-1 ». Dans cet exemple, nous avons créé une famille de colonnes pour chaque table relationnelle puisqu'habituellement les données de deux tables ne sont pas accessible ensembles. De cette façon, nous pouvons améliorer l'efficacité des opérations de lecture/l'écriture dans la base de données HBase.

Une autre solution possible pour convertir la relations « 1-1 », c'est de créer une table HBase avec une seule famille de colonne, et toutes les colonnes de deux tables relationnelles sont converties en qualificatif de colonnes. Finalement, la clé primaire de la table mère est convertie comme un *RowKey*. Cette solution peut être utilisée dans le cas des données qui sont fortement liées et accessible en même temps.

D'une manière générale, deux solutions sont proposées pour convertir une relation de type « 1-1 » entre une table relationnelle mère T_1 et table relationnelle fille T_2 :

- Solution 1 :
  - Une table HBase *TH* est créée avec le nom de la table mère.
  - Deux familles de colonnes FC_1^{TH} et FC_2^{TH} sont créées et portent respectivement les noms des tables relationnelles T_1 et T_2.
  - L'ensembles des colonnes Cln^{T_1} et Cln^{T_2} des tables relationnelles T_1 et T_2 sont converties respectivement à des ensembles des qualificatifs de colonnes Cln^{FC_1} et Cln^{FC_2} dans les familles de colonnes correspondantes.
  - La clé primaire de la table mère T_1 est convertie en *Rowkey RK* des lignes de la table HBase *TH*.
- Solution 2 : les mêmes étapes doivent être suivis. La seule différence est de créer une seule famille de colonnes FC_1^{TH} qui est composée de l'ensemble des qualificatifs de colonnes Cln^{FC_1}issu de la conversion de l'ensembles des colonnes Cln^{T_1} et Cln^{T_2} des tables relationnelles T_1 et T_2.

4.3.4. Règle 4 : Transformation des relations de type « 1-n »

Dans une base de données relationnelle, une relation de type « 1-n » se produit lorsque chaque enregistrement de la table *A* peut avoir de nombreux enregistrements liés dans la table *B*, mais chaque enregistrement dans la table *B* ne peut avoir qu'un seul enregistrement correspondant dans table *A*. Par exemple, dans une base de données de « gestion d'une école », un enseignant peut dispenser plusieurs cours, tandis qu'un cours est dispensé par un (et un seul) enseignant.

Une relation de type « 1-n » ne peut pas être représentée dans une seule table. Par exemple, dans une base de données de « gestion d'une école », nous pouvons commencer par une table appelée « enseignants », qui stocke les informations des enseignants (telles que le nom, le bureau, le téléphone et l'email). Pour stocker les informations des cours dispensés par chaque enseignant, nous pourrions créer des colonnes *cours1*, *cours 2*, *cours3*, mais nous sommes immédiatement confrontés à un problème sur le nombre de colonnes à créer. D'autre part, si nous commençons avec une table appelée « cours », qui stocke les informations sur un cours (Id du cours, titre, date, heure de début et heure de fin), nous pourrions créer des colonnes supplémentaires pour stocker les informations sur l'enseignant (comme nom, le bureau, le téléphone et l'email). Cependant, comme un enseignant peut dispenser plusieurs cours, ses données seraient dupliquées sur de nombreuses lignes de la table « cours ».

Pour implémenter une relation de type « 1-n », il faut créer deux tables :

- Une table « cours » pour stocker les informations sur les cours avec le « idCours » comme clé primaire.
- Une table « enseignants » pour stocker les informations sur les enseignants avec le «« idEnseig » comme clé primaire.

Ensuite la relation de type « 1-n » peut être implémentée en stockant la clé primaire de la table « enseignants » (idCours) dans la table « cours » comme illustré dans la Figure 75.

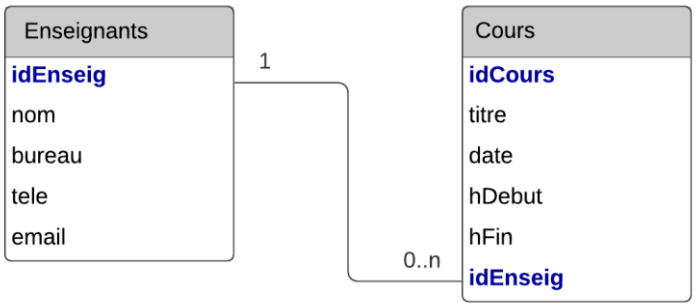


Figure 75: Exemple de d'une relation de type « 1-n » dans le model relationnel

Pour convertir le schéma relationnel présenté dans la Figure 75 nous proposons trois solutions :

- **Solution 1 : Méthode verticale**

Une table HBase est créée avec un nom composé par la concaténation des noms des deux tables relationnelles, puis nous allons créer une famille de colonnes pour chaque table. Les colonnes de chaque table relationnelle sont converties en qualificatifs de colonnes dans la famille de colonnes correspondante. Deux types de *RowKey* seront utilisés :

a) *RowKey* pour les lignes contenant les données des enseignants ; tous simplement c'est l'identifiant (clé primaire) de la ligne dans la table relationnelle.

b) *RowKey* pour identifier les informations d'un cours d'un enseignant. Nous concaténons l'identifiant de l'enseignant avec l'identifiant du cours.

La Figure 76 présente deux tables relationnelles avec une relation de type « 1-n » et la Figure 77 illustre le résultat de la conversion de ces deux tables vers HBase.

Table Relationnelle Enseignants

idEnseig	nom	bureau	tele	email
1	Cherti	48	451	cherti@uh1.ma
2	Bousalem	94	489	bousalem@uh1.ma
3	Haj	25	384	haj@uh1.ma

Table Relationnelle Cours

idCours	titre	date	hDebut	hFin	idEnseig
1	Java – les bases	11/01/20	15:00	17:00	1
2	Java - POO	15/01/20	14:00	16:00	1
3	HBase - Intro	13/01/20	09:30	11:00	2
4	UML - Intro	29/01/20	10:30	12:00	3

Figure 76: Deux tables relationnelles avec une relation « 1-n »

Table HBase Enseignants_Cours

rowKey	valeur
1	cf_enseignants : {'cq_nom':'Cherti', 'cq_bureau':'48', 'cq_tele':'451', 'cq_email':'cherti@uh1.ma'}
1_1	cf_cours : {'cq_titre':'Java – les bases', 'cq_date':'11/01/20', 'cq_hDebut':'15:00', 'cq_hFin':'17:00'}
1_2	cf_cours : {'cq_titre':'Java – POO', 'cq_date':'15/01/20', 'cq_hDebut':'14:00', 'cq_hFin':'16:00'}
2	cf_enseignants : {'cq_nom':'Bousalem', 'cq_bureau':'94', 'cq_tele':'489', 'cq_email':'bousalem@uh1.ma'}
2_3	cf_cours : {'cq_titre':'HBase - Intro', 'cq_date':'13/01/20', 'cq_hDebut':'09:30', 'cq_hFin':'11:00'}
3	cf_enseignants : {'cq_nom':'haj', 'cq_bureau':'25', 'cq_tele':'384', 'cq_email':'haj@uh1.ma'}
3_4	cf_cours : {'cq_titre':'UML - Intro', 'cq_date':'29/01/20', 'cq_hDebut':'10:30', 'cq_hFin':'12:00'}

Figure 77: Le résultat de la conversion des tables relationnelles de la Figure 76 vers HBase en utilisant la méthode verticale

- **Solution 2 : Méthode horizontale**

L'idée de cette méthode est de stocker tous les cours d'un enseignant sur une seule ligne, ce qui permet d'accéder à plus de données avec un seul *RowKey*. Pour cela, nous proposons de créer une table HBase avec un nom généré par la concaténation des noms des deux tables relationnelles, ensuite nous allons créer deux familles de colonnes. La première pour la table mère et la deuxième pour la table fille. Les colonnes de la table mère sont converties en qualificatifs de colonnes dans la première famille de colonnes. Les noms des qualificatifs de colonnes dans la deuxième famille de colonnes seront l'identifiant du cours concaténé avec le nom de chaque colonne dans la table fille (cf. Figure 78). Le *RowKey* des lignes est l'identifiant de de la table mère.

Table HBase Enseignants_Cours

rowKey	valeur
1	cf_enseignants : {'cq_nom':' Cherti', 'cq_bureau':'48', 'cq_tele':'451', 'cq_email':'cherti@uh1.ma'} cf_cours : {'cq_1_titre':'Java – les bases', 'cq_1_date':'11/01/20', 'cq_1_hDebut':'15:00', 'cq_1_hFin':'17:00', 'cq_2_titre':'Java – POO', 'cq_2_date':'15/01/20', 'cq_2_hDebut':'14:00', 'cq_2_hFin':'16:00'}
2	cf_enseignants : {'cq_nom':'Bousalem', 'cq_bureau':'94', 'cq_tele':'489', 'cq_email':'bousalem@uh1.ma'} cf_cours : {'cq_3_titre':'HBase - Intro', 'cq_3_date':'13/01/20', 'cq_3_hDebut':'09:30', 'cq_3_hFin':'11:00'}
3	cf_enseignants : {'cq_nom':'haj', 'cq_bureau':'25', 'cq_tele':'384', 'cq_email':'haj@uh1.ma'} cf_cours : {'cq_4_titre':'UML - Intro', 'cq_4_date':'29/01/20', 'cq_4_hDebut':'10:30', 'cq_4_hFin':'12:00'}

Figure 78: Le résultat de la conversion des tables relationnelles de la Figure 76 vers HBase en utilisant la méthode horizontale

- **Solution 3 : Méthode horizontale par colonnes**

L'idée est de regrouper pour un enseignant toutes les valeurs d'une colonne de la table « cours » dans une même famille de colonnes. En effet, les données enregistrées dans la même famille de colonnes sont stockées ensemble dans le système de fichiers alors que les données de familles de colonnes différentes peuvent être réparties sur plusieurs machines dans le cluster. L'objectif de cette méthode est de créer un schéma qui améliore l'efficacité de l'accès aux données lorsqu'il s'agit de sélectionner les valeurs de certaines colonnes de la table fille. Pour y parvenir, nous proposons de créer une table HBase avec un nom composé de la concaténation des noms des deux tables relationnelles. Concernant les familles de colonnes, en premier lieu, nous allons créer une famille de colonne pour la table mère « enseignants » , et les colonnes de cette dernière seront converties en qualificatifs de colonnes. En second lieu, des nouvelles familles de colonnes seront créées, et les noms de celles-ci sont les noms des colonnes de la table fille « cours ». Ensuite, les noms des qualificatifs de colonnes de chaque famille de colonnes seront définis à partir des identifiants des lignes de la table fille. La Figure 79 présente un exemple de conversion des tables relationnelles de la Figure 76 vers HBase en utilisant la méthode horizontale par colonnes.

Table HBase Enseignants_Cours

rowKey	valeur
1	cf_enseignants : {'cq_nom':'Cherti ', 'cq_bureau':'48', 'cq_tele':'451', 'cq_email':'cherti@uh1.ma'} cf_titre : {'cq_1':'Java – les bases', 'cq_2':'Java - POO'} cf_date : {'cq_1':'11/01/20', 'cq_2':'15/01/20'} cf_hDebut : {'cq_1':'15:00', 'cq_2':'14:00'} cf_hFin : {'cq_1':'17:00', 'cq_2':'16:00'}
2	cf_enseignants : {'cq_nom':'Bousalem', 'cq_bureau':'94', 'cq_tele':'489', 'cq_email':'bousalem@uh1.ma'} cf_titre : {'cq_3':'HBase - Intro'} cf_date : {'cq_3':'13/01/20'} cf_hDebut : {'cq_3':'09:30'} cf_hFin : {'cq_3':'11:00'}
3	cf_enseignants : {'cq_nom':'haj', 'cq_bureau':'25', 'cq_tele':'384', 'cq_email':'haj@uh1.ma'} cf_titre : {'cq_4':'UML - Intro'} cf_date : {'cq_4':'29/01/20'} cf_hDebut : {'cq_4':'10:30'} cf_hFin : {'cq_4':'12:00'}

Figure 79: Le résultat de la conversion des tables relationnelles de la Figure 76 vers HBase en utilisant la méthode horizontale par colonnes

D'une manière générale, pour convertir une relation « 1-n » entre une table relationnelle mère T_1 et table relationnelle fille T_2, nous avons proposé trois solutions :

- Solution 1 : Méthode verticale
  - Une table HBase TH est créée avec un nom composé de la concaténation des noms des tables relationnelles T_1 et T_2.
  - Deux familles de colonnes FC_1^{TH} et FC_2^{TH} sont créées et portent respectivement les noms des tables relationnelles T_1 et T_2.
  - L'ensembles des colonnes Cln^{T_1} et Cln^{T_2} des tables relationnelles T_1 et T_2 sont converties respectivement à des ensembles des qualificatifs

de colonnes Cln^{FC_1} et Cln^{FC_2} dans les familles de colonnes correspondantes.

- La clé primaire de la table mère *T1* est convertie en *Rowkey RK1* des lignes de la table HBase *TH* qui enregistrent les informations de la table mère *T1*. Concernant les lignes de la table HBase *TH* qui stockent les informations de la table fille *T2*, elles sont indexées par un *Rowkey RK2* généré par la concaténation de la clé primaire et la clé étrangère de chaque ligne la table fille *T2*.

- Solution 2 : Méthode horizontale
  - Une table HBase *TH* est créée avec un nom généré en concaténant les noms des tables relationnelles *T1* et *T2*.
  - Deux familles de colonnes FC_1^{TH} et FC_2^{TH} sont créées et portent respectivement les noms des tables relationnelles *T1* et *T2*.
  - L'ensemble des colonnes Cln^{T_1} de la table relationnelle *T1* est convertie à un ensemble des qualificatifs de colonnes Cln^{FC_1} dans la famille de colonnes FC_1^{TH}. Les noms des qualificatifs de colonnes de la deuxième famille de colonnes seront l'identifiant du cours concaténé avec le nom de la colonne dans la table fille.
  - Les noms des qualificatifs de colonnes de l'ensemble Cln^{FC_2} sont générés par concaténation de chaque ligne de la table *T2* avec le nom de chaque colonne de l'ensemble Cln^{T_2}. Le nombre des qualificatifs de colonnes de la famille de colonnes FC_2^{TH} est calculé comme suit : $|Cln^{FC_2}| = |T_2| \times |Cln^{T_2}|$.
  - La clé primaire de la table mère *T1* est convertie en *Rowkey RK* des lignes de la table HBase *TH*.

- Solution 3 : Méthode horizontale par colonnes
  - Une table HBase *TH* est créée avec un nom généré en concaténant les noms des tables relationnelles *T1* et *T2*.
  - Des familles de colonnes sont créées dont la première (FC_1^{TH}) porte le nom de la table mère *T1* et les noms des autres sont générés à partir des noms des colonnes de l'ensemble Cln^{T_2}.
  - L'ensemble des colonnes Cln^{T_1} de la table relationnelle *T1* est convertie à un ensemble des qualificatifs de colonnes Cln^{FC_1} dans la famille de colonnes FC_1^{TH}. Les noms des qualificatifs de colonnes des

autres familles de colonnes seront la clé primaire de chaque ligne de la table fille T_2.

- La clé primaire de la table mère T_1 est convertie en *Rowkey RK* des lignes de la table HBase *TH*.

4.3.5. Règle 5 : Transformation des relations de type « n-m »

Dans une base de données de gestion des ventes, la commande d'un client peut contenir un ou plusieurs produits ; et un produit peut apparaître dans plusieurs commandes. Ce type de relations est connu sous le nom de « n-m » ou « plusieurs à plusieurs ». Au début, nous commençons par deux tables relationnelles « *Produits* » et « *Commandes* ». La table « *Produits* » contient des informations sur les produits (le nom, la quantité, etc.) avec la colonne « *idPrdt* » comme clé primaire. La table « *Commandes* » contient les informations sur les commandes des clients (l'identifiant du client, la date de la commande, etc.). Là encore, nous ne pouvons pas stocker les articles commandés dans la table « *Commandes* », car nous ne savons pas combien de colonnes à réserver pour les articles, et aussi nous ne pouvons pas stocker les informations relatives à une commande dans la table Produits.

Pour implémenter la relation « plusieurs à plusieurs », nous devons créer une troisième table (appelé table de jonction), par exemple « *lignesCommandes* », où chaque ligne représente un article d'une commande particulière (cf. Figure 80). Pour la table « *lignesCommandes* », la clé primaire se compose de deux colonnes : « *idCmd* » et « *idPrdt* », qui identifient chaque ligne de manière unique.

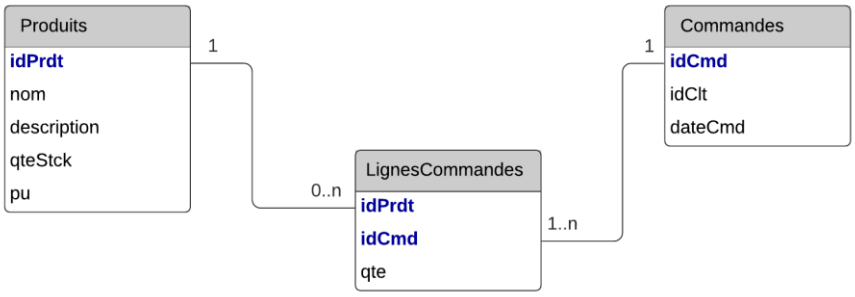


Figure 80: Exemple d'une relation « n-m » dans le model relationnel

Pour convertir le schéma relationnel présenté dans la Figure 80, là aussi nous utilisons le principe de conception *DDI* qui est basé sur la technique de dénormalisation, duplication et les clés intelligentes. L'objectif de notre solution est de stocker toutes les combinaisons possibles entre un produits et une commande avec leur détails(lignes de commandes). Pour ce faire, une table HBase est créée avec un nom composé par la concaténation des noms des

deux tables relationnelles principales, ensuite nous allons créer deux familles de colonnes. La première pour la table « *Commandes* » et la deuxième pour la table « *Produits* » et ses détails stockées dans la table « *lignesCommandes* ». Les colonnes de la table relationnelle « *Commandes* » sont converties en qualificatifs de colonnes dans la famille de colonnes « *Commandes* ». Concernant les qualificatifs de la deuxième famille de colonnes, nous allons les générer en convertissant les colonnes de la table « *Produits* » et les colonnes qui stockent les détails d'une commande dans la table « *lignesCommandes* » . Nous allons utiliser un *RowKey* composé par l'identifiant du produit et l'identifiant de la commande pour indexer tous les détails d'une commande.

Table Relationnelle Produits

idPrdt	nom	description	qteStck	pu
1	chandail	Gros tricot	25	180
2	robes	robes longues	15	250
3	manteaux	manteaux homme	27	450

Table Relationnelle Commandes

idCmd	idClt	dateCmd
1	2	05/08/2020
2	1	06/08/2020
3	6	09/08/2020

Table Relationnelle lignesCommandes

idCmd	idPrdt	qte
1	3	3
1	1	5
2	3	2
2	1	1
2	2	7
3	2	4

Figure 81: Exemple de tables relationnelles avec une relation « n-m »

La Figure 81 présente des tables relationnelles avec une relation de type « n-m » et la Figure 82 illustre le résultat de la conversion de ces tables vers HBase.

Table HBase Produits_Commandes

rowKey	valeur
1	cf_produit : {'cq_nom':'chandail', 'cq_description':'Gros tricot', 'cq_qteStck':'25', 'cq_pu':'180'}
1_3_1	cf_commandes : {'cq_idClt ':'2', 'cq_dateCmd ':'05/08/2020', 'cq_3_qte':'1', 'cq_1_qte':'5'}
2	cf_produit : {'cq_nom':'robes', 'cq_description':'robes longues', 'cq_qteStck':'15', 'cq_pu':'250'}
2_3_1_2	cf_commandes : {'cq_idClt ':'1', 'cq_dateCmd ':'06/08/2020', 'cq_3_qte':'2', 'cq_1_qte':'1', 'cq_2_qte':'7'}
3	cf_produit : {'cq_nom':'manteaux', 'cq_description':'manteaux homme', 'cq_qteStck':'27', 'cq_pu':'450'}
3_2	cf_commandes : {'cq_idClt ':'6', 'cq_dateCmd ':'09/08/2020', 'cq_1_qte':'4'}

Figure 82: Le résultat de la conversion des tables relationnelles de la Figure 81 vers HBase en utilisant la méthode hybride

5. Conclusions

Ces dernières années, grâce à l'augmentation spectaculaire du volume, de la variété et de la vitesse des données, nous sommes entrés dans l'ère des données massive [205]. La structure des données est devenue très flexible, ce qui a conduit au développement de nombreux systèmes de stockage différents des bases de données traditionnelles où les données sont stockées dans des tables avec un schéma rigide. Bien que les bases de données relationnelles soient toujours prédominantes dans l'industrie, il y a un important mouvement vers les systèmes de bases de données alternatifs qui prennent en charge des données non structurées avec une meilleure évolutivité, ce qui a conduit à la popularité des bases de données NoSQL[206]. La migration des bases de données relationnelles vers les bases de données NoSQL est un sujet de recherche qui suscite de plus en plus d'intérêt ces dernières années. Dans ce chapitre nous proposons une approche de migration d'une base de données relationnels vers une base de données HBase en donnant des règles de transformation de schéma. Tous d'abord, nous avons cité quelques outils de migration de données avec leurs limites en relation avec la conversion des dépendances et les relations entre les tables. Ensuite, nous avons présenté un schéma qui explique le processus de migration. Finalement,

nous avons proposé un ensemble des règles qui aident à réussir l'opération de conversion de schéma d'une base de données relationnelle vers une base de données HBase.

Conclusion Générale et Perspectives

Les travaux de recherche effectués dans cette thèse s'inscrivent dans le domaine de la conception des architectures de migration des bases de données divers. Cet axe de recherche est valorisé par l'intégration de plusieurs domaines de recherche. La problématique abordée est celle de la migration des bases de données XML vers les bases de données relationnelles et des bases de données relationnelles vers les bases de données NoSQL.

Notre première contribution s'articule essentiellement autour du stockage et l'interrogation des documents XML à l'aide des bases de données relationnelles. Notre approche décompose le document XML en trois tables sans avoir besoin d'utiliser le XML Schema ou DTD. Cette approche supporte toutes les modifications structurelles de l'arbre XML à savoir l'insertion, la modification, et la suppression. Nous finissons cette contribution par la proposition de deux algorithmes. Le premier permet de mapper les données XML vers une base de données relationnelle. Et le deuxième algorithme permet d'assurer la reconstruction d'un document XML à partir d'une base de données relationnelle.

L'idée générale de notre deuxième contribution est de générer automatiquement un cours pour chaque apprenant en fonction de ses caractéristiques et ses préférences individuelles. Cependant, la grande question qui se pose est de savoir structurer, stocker et indexer les contenus d'apprentissage tout en assurant une meilleure adaptation. Dans ce but, nous avons opté pour le langage XML pour représenter notre support de cours. Par ailleurs, afin d'éviter les faiblesses des bases de données XML et de bénéficier des points forts des bases de données relationnelles, nous avons utilisé notre approche E-XMap pour stocker le document XML du support de cours dans une base de données relationnelle. Parallèlement, nous utilisons les techniques d'étiquetage XML pour identifier les relations structurelles entre les nœuds et aussi pour accélérer le traitement des requêtes.

Notre troisième contribution consiste essentiellement à étudier la faisabilité de la migration des bases de données relationnelles vers les bases de données NoSQL et plus particulièrement la base de données orientée colonnes HBase. Ceci en proposant un modèle relationnel à la base de données HBase. L'objectif de cette contribution est de comparer le modèle des bases de données relationnelles et de HBase, en appliquant les opérations de base de l'algèbre relationnelle dans le modèle de données HBase et en citant quelques implémentations des opérations clés de l'algèbre relationnelle dans la base de données de HBase.

Dans notre quatrième contribution nous avons comparé les performances du modèle relationnel (MySQL) et le modèle NoSQL (HBase) pour révéler l'impact de l'augmentation du nombre d'opérations et du nombre d'enregistrements sur la performance en termes de la latence et du temps d'exécution en utilisant le Framework YCSB. L'objectif principal de cette comparaison est d'identifier les points forts et les points faibles de chaque modèle afin de les prendre en considération dans une éventuelle migration.

Notre cinquième et dernière contribution met la lumière sur le processus de migration d'une base de données relationnelle vers la base de données orientée colonnes HBase. Ceci en expliquant chaque étape de migration. Nous finissons cette contribution par la proposition d'un ensemble de règles de transformation de schéma d'une base de données relationnelle vers HBase.

Les contributions présentées dans cette thèse ouvrent la voie à un certain nombre de perspectives que nous abordons en trois points.

Le premier point vise à améliorer notre approche XMap pour supporter d'autres types nœuds tels que l'espace de noms, l'instruction de traitement et le commentaire. Une autre piste d'amélioration de l'approche XMap consiste à développer un Framework pour la traduction automatique de requêtes XML en requêtes SQL en supportant tous les types de requêtes (sélection, insertion, mise à jour et suppression).

Le deuxième point est lié à la comparaison des performances du SGBD relationnel MySQL et le SGBD NoSQL HBase. Nous envisageons diversifier les critères de comparaison pour comprendre davantage les différences entre ces deux types de bases de données. Pour cela, nous proposons des tests complémentaires à savoir :

- L'évaluation des performances des bases de données lorsque tous les enregistrements pourront être chargés en RAM.
- L'évaluation des performances des bases de données lorsque des opérations supplémentaires d'entrée/sortie sont utilisées.
- L'évaluation de la capacité de la base de données à servir un nombre élevé des clients.
- La comparaison de la scalabilité de MySQL et HBase.

Nous avons détaillé cette perspective à la fin du chapitre V.

Le troisième point concerne la migration d'une base de données relationnelle vers une base de données HBase. Nous envisageons améliorer notre approche en proposant des règles de transformation pour les autres types de relations à savoir l'agrégation, la composition et la spécialisation. Une autre réflexion consiste à automatiser l'opération de migration tout en appliquant nos règles de transformation de schéma proposées.

Bibliographie

[1] A. Davoudian, L. Chen, and M. Liu, “A survey on NoSQL stores,” ACM Comput. Surv. CSUR, vol. 51, no. 2, pp. 1–43, 2018.

[2] E. F. Codd, “A Relational Model of Data for Large Shared Data Banks,” Commun ACM, vol. 13, no. 6, pp. 377–387, Jun. 1970, doi: 10.1145/362384.362685.

[3] P. Csr, Object–Oriented Database Systems: Approaches and Architectures. PHI Learning Pvt. Ltd., 2005.

[4] J.-K. Chen and W.-Z. Lee, “An Introduction of NoSQL Databases based on their categories and application industries,” Algorithms, vol. 12, no. 5, p. 106, 2019.

[5] A. Meier and M. Kaufmann, “Nosql databases,” in SQL & NoSQL Databases, Springer, 2019, pp. 201–218.

[6] N. Chaudhry and M. M. Yousaf, “Architectural assessment of NoSQL and NewSQL systems,” Distrib. Parallel Databases, vol. 38, no. 4, pp. 881–926, 2020.

[7] I. Astrova, A. Koschel, N. Wellermann, and P. Klostermeyer, “Performance Benchmarking of NewSQL Databases with Yahoo Cloud Serving Benchmark,” in Proceedings of the Future Technologies Conference, 2020, pp. 271–281.

[8] E. Khanjari and L. Gaeini, “A new effective method for labeling dynamic XML data,” J. Big Data, vol. 5, no. 1, Art. no. 1, Dec. 2018, doi: 10.1186/s40537-018-0161-4.

[9] A. Qtaish and M. T. Alshammari, “A Narrative Review of Storing and Querying XML Documents Using Relational Database,” J. Inf. Knowl. Manag., vol. 18, no. 04, p. 1950048, 2019.

[10] T. Kudrass and M. Conrad, “Management of XML documents in object-relational databases,” in International Conference on Extending Database Technology, 2002, pp. 210–227.

[11] T. Naser, R. Alhajj, and M. J. Ridley, “Reengineering XML into object-oriented database,” in 2008 IEEE International Conference on Information Reuse and Integration, 2008, pp. 1–6.

[12] “Domo Inc. Data never sleeps 8.0 infographic — Domo.” https://www.domo.com/learn/data-never-sleeps-8 (accessed Sep. 24, 2020).

[13] S. Mazumdar, D. Seybold, K. Kritikos, and Y. Verginadis, “A survey on data storage and placement methodologies for cloud-big data ecosystem,” J. Big Data, vol. 6, no. 1, p. 15, 2019.

[14] E. Song and S.-C. Haw, “XML-REG: Transforming XML Into Relational Using Hybrid-Based Mapping Approach,” IEEE Access, vol. 8, pp. 177623–177639, 2020, doi: 10.1109/ACCESS.2020.3026006.

[15] Z. Bousalem and I. Cherti, “XMap: A Novel Approach to Store and Retrieve XML Document in Relational Databases.,” JSW, vol. 10, no. 12, pp. 1389–1401, 2015.

[16] A. A. El-Aziz and A. Kannan, “Relational storage for XML rules,” Int. J. Web Serv. Comput., vol. 3, no. 4, p. 33, 2012.

[17] A. M. Saba, E. Shahab, H. Abdolrahimpour, M. Hakimi, and A. Moazzam, “A Comparative Analysis of XML Documents, XML Enabled Databases and Native XML Databases,” ArXiv Prepr. ArXiv170708259, 2017.

[18] M. A. I. Fakharaldien, K. Edris, J. M. Zain, and N. Sulaiman, “Mapping extensible markup language document with relational database management system,” Int J Phys Sci, vol. 7, pp. 4012–4025, 2012.

[19] D. Florescu and D. Kossmann, “Storing and querying XML data using an RDMBS,” IEEE Data Eng. Bull., vol. 22, p. 3, 1999.

[20] J. S. K. T. G. He and C. Z. D. D. J. Naughton, “Relational databases for querying XML documents: Limitations and opportunities,” in Proceedings of VLDB, 2008, pp. 302–314.

[21] W. Kim, “XRel: a path-based approach to storage and retrieval of XML documents using relational databases,” ACM Trans. Internet Technol. TOIT, vol. 1, no. 1, pp. 110–141, 2001.

[22] A. Qtaish and K. Ahmad, “XAncestor: An efficient mapping approach for storing and querying XML documents in relational database using path-based technique,” Knowl.-Based Syst., vol. 114, pp. 167–192, 2016.

[23] P. Suri and D. Sharma, “Schema based storage of XML documents in relational databases,” Int. J. Web Serv. Comput., vol. 4, no. 2, p. 23, 2013.

[24] P. Suri and D. Sharma, “A Model Mapping Approach for storing XML documents in Relational databases,” Int. J. Comput. Sci. Issues IJCSI, vol. 9, no. 3, p. 495, 2012.

[25] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang, “Storing and querying ordered XML using a relational database system,” in Proceedings of the 2002 ACM SIGMOD international conference on Management of data, 2002, pp. 204–215.

[26] H. Zhu, H. Yu, G. Fan, and H. Sun, “Mini-XML: An efficient mapping approach between XML and relational database,” in 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS), 2017, pp. 839–843.

[27] D. J. Abadi, “Data management in the cloud: Limitations and opportunities.,” IEEE Data Eng Bull, vol. 32, no. 1, pp. 3–12, 2009.

[28] R. Schneider, “Research data literacy,” in European Conference on Information Literacy, 2013, pp. 134–140.

[29] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, Extensible markup language (XML) 1.0. W3C recommendation October, 2000.

[30] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "W3C Recommendation: Document Type Definition (DTD)," World Wide Web Consort. Recomm. REC-Xml-19980210 Httpwww W3 OrgTR1998REC-Xml-19980210, vol. 16, p. 16, 1998.

[31] World Wide Web Consortium, XML Schema Part 0: Primer Second Edition. W3C Recommendation 28 October 2004. 2004.

[32] J. Clark and M. Murata, RELAX NG specification, OASIS committee specification, December 2001. 2001.

[33] L. Dodds, "Schematron: validating XML using XSLT," 2001.

[34] C. Frankston and H. S. Thompson, "XML-Data reduced," Unpubl. Draft W3C Note Version 021 Univ. Edinb., 1998.

[35] A. Davidson et al., "Schema for object-oriented XML 2.0," W3C Note July, 1999.

[36] O. Falola, S. Misra, A. Adewumi, and R. Damasevicius, "Evaluation and Comparison of Metrics for XML Schema Languages.," in ICADIWT, 2017, pp. 51–59.

[37] H. Darwen, "The Relational Model: Beginning of an Era," IEEE Ann. Hist. Comput., vol. 34, no. 4, pp. 30–37, Oct. 2012, doi: 10.1109/MAHC.2012.50.

[38] Y. Liu, X. Zeng, K. Zhang, and Y. Zou, "Transforming Entity-Relationship Diagrams to Relational Schemas Using a Graph Grammar Formalism," in 2018 IEEE International Conference on Progress in Informatics and Computing (PIC), Dec. 2018, pp. 327–331. doi: 10.1109/PIC.2018.8706334.

[39] C. Y. Park and K. B. Laskey, "MEBN-RM: A Mapping between Multi-Entity Bayesian Network and Relational Model," Appl. Sci., vol. 9, no. 9, p. 1743, 2019.

[40] M. Levene and G. Loizou, A guided tour of relational databases and beyond. Springer Science & Business Media, 2012.

[41] O. Abahussain and A. Alqaddoumi, "DBMS, NoSQL and Securing Data: the relationship and the recommendation," in 2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT), Dec. 2020, pp. 1–6. doi: 10.1109/3ICT51146.2020.9311958.

[42] J. L. Harrington, Relational database design and implementation: clearly explained. Morgan Kaufmann/Elsevier (Boston), 2009.

[43] P. Suri and M. Sharma, "A Comparative Study Between the Performance of Relational & Object Oriented Database in Data Warehousing," Int. J. Database Manag. Syst., vol. 3, no. 2, pp. 116–127, May 2011, doi: 10.5121/ijdms.2011.3208.

[44] F. Kalis, “Codd’s Rules,” Best SQLServerCentral Com Vol 2, p. 158, 2003.

[45] E. F. Codd, “An evaluation scheme for database management systems that are claimed to be relational,” in 1986 IEEE Second International Conference on Data Engineering, Feb. 1986, pp. 720–729. doi: 10.1109/ICDE.1986.7266284.

[46] F. Ferrarotti, S. Hartmann, H. Köhler, S. Link, and M. Vincent, “The boyce-codd-heath normal form for SQL,” in International Workshop on Logic, Language, Information, and Computation, 2011, pp. 110–122.

[47] T. Haerder and A. Reuter, “Principles of transaction-oriented database recovery,” ACM Comput. Surv. CSUR, vol. 15, no. 4, pp. 287–317, 1983.

[48] D. C. Jamison, “Structured query language (SQL) fundamentals,” Curr. Protoc. Bioinforma., no. 1, pp. 9–2, 2003.

[49] A. I. Din, Structured query language (SQL) A practical Introduction. 2014.

[50] S. Sumathi and S. Esakkirajan, “Structured query language,” Fundam. Relational Database Manag. Syst., pp. 111–212, 2007.

[51] “Definition of Big Data - Gartner Information Technology Glossary,” Gartner. https://www.gartner.com/en/information-technology/glossary/big-data (accessed Mar. 16, 2021).

[52] J. Manyika et al., “Big data: The next frontier for innovation, competition,” and productivity. Technical report, McKinsey Global Institute, 2011.

[53] P. Hitzler and K. Janowicz, “Linked Data, Big Data, and the 4th Paradigm.,” Semantic Web, vol. 4, no. 3, pp. 233–235, 2013.

[54] Y. Demchenko, “The Big Data Architecture Framework (BDAF),” Outcome Brainstorming Sess. Univ. Amst., vol. 17, pp. 1494–1499, 2013.

[55] T. White, Hadoop: The definitive guide. O’Reilly Media, Inc., 2012.

[56] A. Holmes, Hadoop in practice, vol. 3. Manning New York;, 2012.

[57] V. K. Singh, M. Taram, V. Agrawal, and B. S. Baghel, “A literature review on Hadoop ecosystem and various techniques of big data optimization,” Adv. Data Inf. Sci., pp. 231–240, 2018.

[58] R. Vasuja, A. Bhandralia, and K. Chuchra, “Daemons of Hadoop: An Overview,” Int. J. Eng. Res. Technol., pp. 2278–0181, 2018.

[59] N. Kaskatiiski and L. Boyanov, “Acquiring and storing Internet of Things data into Hadoop,” in IOP Conference Series: Materials Science and Engineering, 2021, vol. 1031, no. 1, p. 012060.

[60] O. Azeroual and R. Fabre, “Processing Big Data with Apache Hadoop in the Current Challenging Era of COVID-19,” Big Data Cogn. Comput., vol. 5, no. 1, p. 12, 2021.

[61] D. Borthakur, “The hadoop distributed file system: Architecture and design,” Hadoop Proj. Website, vol. 11, no. 2007, p. 21, 2007.

[62] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” Commun. ACM, vol. 51, no. 1, pp. 107–113, 2008.

[63] W. Zhao, H. Ma, and Q. He, “Parallel k-means clustering based on mapreduce,” in IEEE international conference on cloud computing, 2009, pp. 674–679.

[64] D. Lee, J.-S. Kim, and S. Maeng, “Large-scale incremental processing with MapReduce,” Future Gener. Comput. Syst., vol. 36, pp. 66–79, 2014.

[65] S. Muppidi and M. R. Murty, “Document clustering with map reduce using Hadoop framework,” Int. J. Recent Innov. Trends Comput. Commun., vol. 3, no. 1, pp. 409–413, 2015.

[66] M. A. Rodriguez and P. Neubauer, “The graph traversal pattern,” in Graph Data Management: Techniques and Applications, IGI Global, 2012, pp. 29–46.

[67] S. Amer-Yahia et al., “Databases and Web 2.0 panel at VLDB 2007,” ACM SIGMOD Rec., vol. 37, no. 1, pp. 49–52, 2008.

[68] S. N. Swaminathan and R. Elmasri, “Quantitative analysis of scalable NoSQL databases,” in 2016 IEEE International Congress on Big Data (BigData Congress), 2016, pp. 323–326.

[69] A. B. M. Moniruzzaman and S. A. Hossain, “Nosql database: New era of databases for big data analytics-classification, characteristics and comparison,” ArXiv Prepr. ArXiv13070191, 2013.

[70] A. Oussous, F.-Z. Benjelloun, A. A. Lahcen, and S. Belfkih, “NoSQL databases for big data,” Int. J. Big Data Intell., vol. 4, no. 3, pp. 171–185, 2017.

[71] F. Chang et al., “Bigtable: A distributed storage system for structured data,” ACM Trans. Comput. Syst. TOCS, vol. 26, no. 2, p. 4, 2008.

[72] H. Junno, “Recommendation engine with Neo4j graph database,” 2019.

[73] A. E. Lotfy, A. I. Saleh, H. A. El-Ghareeb, and H. A. Ali, “A middle layer solution to support ACID properties for NoSQL databases,” J. King Saud Univ.-Comput. Inf. Sci., vol. 28, no. 1, pp. 133–145, 2016.

[74] D. Pritchett, “BASE: An Acid Alternative: In partitioned databases, trading some consistency for availability can lead to dramatic improvements in scalability.,” Queue, vol. 6, no. 3, pp. 48–55, 2008.

[75] E. A. Brewer, “Towards robust distributed systems,” in PODC, 2000, vol. 7, no. 10.1145, pp. 343477–343502.

[76] S. Gilbert and N. Lynch, “Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services,” Acm Sigact News, vol. 33, no. 2, pp. 51–59, 2002.

[77] J. R. Lourenço, B. Cabral, P. Carreiro, M. Vieira, and J. Bernardino, “Choosing the right NoSQL database for the job: a quality attribute evaluation,” J. Big Data, vol. 2, no. 1, p. 18, 2015.

[78] A. Simec and M. Maglicic, “Comparison of JSON and XML data formats,” in Central European Conference on Information and Intelligent Systems, 2014, p. 272.

[79] F. Du, S. Amer-Yahia, and J. Freire, “ShreX: Managing XML documents in relational databases,” in Proceedings of the Thirtieth international conference on Very large data bases-Volume 30, 2004, pp. 1297–1300.

[80] M. Ramanath, J. Freire, J. R. Haritsa, and P. Roy, “Searching for efficient XML-to-relational mappings,” in International XML Database Symposium, 2003, pp. 19–36.

[81] M. E. El-Sharkawi and N. E.-H. El Tazi, “LNV: relational database storage structure for XML documents,” in The 3rd ACS/IEEE International Conference onComputer Systems and Applications, 2005., 2005, p. 49.

[82] H. Zafari, K. Hasani, and M. E. Shiri, “Xlight, an efficient relational schema to store and query XML data,” in 2010 International Conference on Data Storage and Data Engineering, 2010, pp. 254–257.

[83] Q. Wang, Z. Ren, L. Dong, and Z. Sheng, “Path-based xml relational storage approach,” Phys. Procedia, vol. 33, pp. 1621–1625, 2012.

[84] M. Atay, Y. Sun, D. Liu, S. Lu, and F. Fotouhi, “Mapping XML data to relational data: A DOM-based approach,” ArXiv Prepr. ArXiv10101746, 2010.

[85] H. Jiang, H. Lu, W. Wang, and J. X. Yu, “Path materialization revisited: an efficient storage model for XML data,” in Australian Computer Science Communications, 2002, vol. 24, no. 2, pp. 85–94.

[86] F. Abduljwad, W. Ning, and X. De, “SMX/R: Efficient way of storing and managing XML documents using RDBMSs based on paths,” in 2010 2nd International Conference on Computer Engineering and Technology, 2010, vol. 1, pp. V1-143.

[87] P. O’Neil, E. O’Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury, “ORDPATHs: insert-friendly XML node labels,” in Proceedings of the 2004 ACM SIGMOD international conference on Management of data, 2004, pp. 903–908.

[88] J. Ying, S. Cao, and Y. Long, “An efficient mapping approach to store and query XML documents in relational database,” in Proceedings of 2012 2nd International Conference on Computer Science and Network Technology, 2012, pp. 2140–2144.

[89] I. M. Dweib, “Automatic mapping of XML documents into relational database,” doctoral, University of Huddersfield, 2010. Accessed: Apr. 04, 2020. [Online]. Available: http://eprints.hud.ac.uk/id/eprint/9701/

[90] A. Qtaish and K. Ahmad, “Model mapping approaches for XML documents: A review,” J. Inf. Sci., vol. 41, no. 4, pp. 444–466, 2015.

[91] S. Subramaniam, S.-C. Haw, and P. K. Hoong, “s-XML: An efficient mapping scheme to bridge XML and relational database,” Knowl.-Based Syst., vol. 27, pp. 369–380, 2012.

[92] M. A. I. Fakharaldien, J. M. Zain, and N. Sulaiman, “XRecursive: An efficient method to store and query XML documents,” ArXiv Prepr. ArXiv12036454, 2012.

[93] J. Clark and S. DeRose, XML Path Language (XPath) 1.0—W3C Recommendation 16 November 1999. Technical Report REC-xpath-19991116, World Wide Web Consortium, 1999.

[94] A. L. Hors et al., “Document Object Model (DOM) Level 2 Core Specification Version 1.0,” W3C Recomm. Nov, 2000.

[95] D. Megginson, Sax 2.0: The simple api for xml. visited 07-01-00. www. megginson. com/SAX, 2001.

[96] E. R. Harold, “An Introduction to StAX,” Internet Available Httpwww Xml Compuba20030117stax Html, 2003.

[97] J. Zhang, “Simplify XML Processing with VTD-XML,” JavaWorld Com, 2006.

[98] R. P. Sonar and M. S. Ali, “iXML-generation of efficient XML parser for embedded system,” in 2016 International Conference on Computing Communication Control and automation (ICCUBEA), 2016, pp. 1–5.

[99] L. J. Deborah, R. Baskaran, and A. Kannan, “Learning styles assessment and theoretical origin in an E-learning scenario: a survey,” Artif. Intell. Rev., vol. 42, no. 4, pp. 801–819, 2014.

[100] S. Chawla, N. Gupta, and R. K. Singla, “LOQES: model for evaluation of learning object,” IJACSA Int. J. Adv. Comput. Sci. Appl., vol. 3, no. 7, 2012.

[101] S. C. Haw and E. Soong, “Performance evaluation on structural mapping choices for data-centric XML documents,” Indones. J. Electr. Eng. Comput. Sci., vol. 18, no. 3, pp. 1539–1550, 2020.

[102] N. Zniber and C. Cauvet, “Systèmes pédagogiques adaptatifs: état de l’art et perspectives,” in MajecSTIC 2005: Manifestation des Jeunes Chercheurs francophones dans les domaines des STIC, 2005, pp. 300–315.

[103] N. Delestre, “Un hypermédia adaptatif dynamique pour l’enseignement,” Thèse Au Lab. PSI Univ. Rouen, 2000.

[104] A. Balla, M. T. Laskri, and S. Laoudi, “HYPERGAP: un hypermédia éducatif dynamique générant des activités pédagogiques,” Doc. Numér., vol. 7, no. 1, pp. 39–57, 2003.

[105] M. Beggas, “Modelisation par Un Systeme Multi-Agents d’un Hypermedia Educatif Adaptatif Dynamique,” Rapp. Stage Master, 2005.

[106] A. Gabillon and M. Fansi, “A new persistent labelling scheme for XML,” 2006.

[107] J. Lu, An Introduction to XML Query Processing and Keyword Search. Springer, 2013.

[108] H. Al Zadjali, “Compressing labels of dynamic XML data using base-9 scheme and Fibonacci encoding,” PhD Thesis, University of Sheffield, 2017.

[109] R. Thonangi, “A Concise Labeling Scheme for XML Data.,” in COMAD, 2006, pp. 4–14.

[110] Q. Li and B. Moon, “Indexing and querying XML data for regular path expressions,” in VLDB, 2001, vol. 1, pp. 361–370.

[111] N. Bruno, N. Koudas, and D. Srivastava, “Holistic twig joins: optimal XML pattern matching,” in Proceedings of the 2002 ACM SIGMOD international conference on Management of data, 2002, pp. 310–321.

[112] J. Lu, T. W. Ling, C.-Y. Chan, and T. Chen, “From region encoding to extended dewey: On efficient processing of XML twig pattern matching,” 2005.

[113] J. Liu, Z. M. Ma, and L. Yan, “Efficient labeling scheme for dynamic XML trees,” Inf. Sci., vol. 221, pp. 338–354, 2013.

[114] M. Duong and Y. Zhang, “LSDX: a new labelling scheme for dynamically updating XML data,” in Proceedings of the 16th Australasian database conference-Volume 39, 2005, pp. 185–193.

[115] M. Duong and Y. Zhang, “Dynamic labelling scheme for xml data processing,” in OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", 2008, pp. 1183–1199.

[116] B. G. Assefa and B. Ergenc, “OrderBased labeling scheme for dynamic XML query processing,” in International Conference on Availability, Reliability, and Security, 2012, pp. 287–301.

[117] L. Xu, T. W. Ling, and H. Wu, “Labeling dynamic XML documents: an order-centric approach,” IEEE Trans. Knowl. Data Eng., vol. 24, no. 1, pp. 100–113, 2010.

[118] M. F. O’Connor and M. Roantree, “SCOOTER: a compact and scalable dynamic labeling scheme for XML updates,” in International Conference on Database and Expert Systems Applications, 2012, pp. 26–40.

[119] T. A. Ghaleb and S. Mohammed, “A dynamic labeling scheme based on logical operators: a support for order-sensitive XML updates,” Procedia Comput. Sci., vol. 57, pp. 1211–1218, 2015.

[120] T. A. Ghaleb and S. A. Mohammed, “XML node labeling and querying using logical operators,” Dec. 27, 2016

[121] D. Gopinathan and K. Asawa, “CBSL - A Compressed Binary String Labeling Scheme for Dynamic Update of XML documents,” J. Comput. Inf. Technol., vol. 26, no. 2, pp. 99–114, Oct. 2018, doi: 10.20532/cit.2018.1003955.

[122] Z. Qin, Y. Tang, F. Tang, J. Xiao, C. Huang, and H. Xu, “Efficient XML query and update processing using a novel prime-based middle fraction labeling scheme,” China Commun., vol. 14, no. 3, pp. 145–157, Mar. 2017, doi: 10.1109/CC.2017.7897330.

[123] X.-T. Nguyen, S.-C. Haw, S. Subramaniam, and C.-K. Pham, “Dynamic Node Labeling Schemes for XML Updates,” in Proceedings of the 6th International Conference On Computing & Informatics (ICOCI), 2017, pp. 505–510.

[124] S.-C. Haw and C.-S. Lee, “Data storage practices and query processing in XML databases: A survey,” Knowl.-Based Syst., vol. 24, no. 8, pp. 1317–1340, 2011.

[125] J. Liu and X. X. Zhang, “Dynamic labeling scheme for XML updates,” Knowl.-Based Syst., vol. 106, pp. 135–149, 2016.

[126] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman, “On supporting containment queries in relational database management systems,” in Proceedings of the 2001 ACM SIGMOD international conference on Management of data, 2001, pp. 425–436.

[127] T. Amagasa, M. Yoshikawa, and S. Uemura, “QRS: A robust numbering scheme for XML documents,” in Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405), 2003, pp. 705–707.

[128] J.-K. Min, J. Lee, and C.-W. Chung, “An efficient XML encoding and labeling method for query processing and updating on dynamic XML data,” J. Syst. Softw., vol. 82, no. 3, pp. 503–515, 2009.

[129] S. Subramaniam, S.-C. Haw, and L.-K. Soon, “Relab: a subtree based labeling scheme for efficient XML query processing,” in 2014 IEEE 2nd International Symposium on Telecommunication Technologies (ISTT), 2014, pp. 121–125.

[130] L. Fu and X. Meng, “Triple code: an efficient labeling scheme for query answering in XML data,” in 2013 10th Web Information System and Application Conference, 2013, pp. 42–47.

[131] S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, D. Srivastava, and Y. Wu, “Structural joins: A primitive for efficient XML query pattern matching,” in Proceedings 18th International Conference on Data Engineering, 2002, pp. 141–152.

[132] E. Cohen, H. Kaplan, and T. Milo, “Labeling dynamic XML trees,” SIAM J. Comput., vol. 39, no. 5, pp. 2048–2074, 2010.

[133] L. Xu, T. W. Ling, H. Wu, and Z. Bao, “DDE: from dewey to a fully dynamic XML labeling scheme,” in Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, 2009, pp. 719–730.

[134] S. Soltan, A. Zarnani, R. AliMohammadzadeh, and M. Rahgozar, “IFDewey: A New Insert-Friendly Labeling Schemafor XML Data,” Int. J. Comput. Inf. Eng., vol. 2, no. 1, pp. 203–205, 2008.

[135] J. Liu, Z. M. Ma, and Q. Qv, “Dynamically querying possibilistic XML data,” Inf. Sci., vol. 261, pp. 70–88, 2014.

[136] S. H. A. Al-khazraji, “A labelling technique comparison for indexing large XML database,” PhD Thesis, University of Sheffield, 2017.

[137] F. Weigel, K. U. Schulz, and H. Meuss, “The BIRD numbering scheme for XML and tree databases–deciding and reconstructing tree relations using efficient arithmetic operations,” in International XML Database Symposium, 2005, pp. 49–67.

[138] Y. Xiao, J. Hong, W. Cui, Z. He, W. Wang, and G. Feng, “Branch code: A labeling scheme for efficient query answering on trees,” in 2012 IEEE 28th International Conference on Data Engineering, 2012, pp. 654–665.

[139] D. An and S. Park, “Group-Based Prime Number Labeling Scheme for XML Data,” in 2010 10th IEEE International Conference on Computer and Information Technology, 2010, pp. 1639–1644.

[140] P. Jayanthi, “Vector based labeling method for dynamic XML documents,” in 2013 International Conference on Information Communication and Embedded Systems (ICICES), 2013, pp. 217–221.

[141] A. Almelibari, “Labelling Dynamic XML Documents: A GroupBased Approach,” PhD Thesis, University of Sheffield, 2015.

[142] S. Subramaniam and S.-C. Haw, “ME labeling: a robust hybrid scheme for dynamic update in XML databases,” in 2014 IEEE 2nd International Symposium on Telecommunication Technologies (ISTT), 2014, pp. 126–131.

[143] R. Al-Shaikh, G. Hashim, A. BinHuraib, and S. Mohammed, “A modulo-based labeling scheme for dynamically ordered XML trees,” in 2010 Fifth International Conference on Digital Information Management (ICDIM), 2010, pp. 213–221.

[144] X. Wu, M.-L. Lee, and W. Hsu, “A prime number labeling scheme for dynamic ordered XML trees,” in Proceedings. 20th International Conference on Data Engineering, 2004, pp. 66–78.

[145] C. Zhuang and S. Feng, “Reuse the Deleted Labels for Vector Order-Based Dynamic XML Labeling Schemes,” in International Conference on Database and Expert Systems Applications, 2012, pp. 41–54.

[146] J.-H. Yun and C.-W. Chung, “Dynamic interval-based labeling scheme for efficient XML query and update processing,” J. Syst. Softw., vol. 81, no. 1, pp. 56–70, Jan. 2008, doi: 10.1016/j.jss.2007.05.034.

[147] Z. Qin, Y. Tang, and X. Wang, “A String Approach for Updates in Order-Sensitive XML Data,” in Proceedings of the 2012 International Conference on Information Technology and Software Engineering, 2013, pp. 153–164.

[148] S.-C. Haw and A. Amin, “Node Indexing in XML Query Optimization: A Review,” Indian J. Sci. Technol., vol. 8, no. 32, pp. 1–9, 2015.

[149] H.-K. Ko and S. Lee, “A Binary String Approach for Updates in Dynamic Ordered XML Data,” IEEE Trans. Knowl. Data Eng., vol. 22, no. 4, pp. 602–607, Apr. 2010, doi: 10.1109/TKDE.2009.87.

[150] J.-F. Pillou and F. Lemainque, Tout sur les réseaux et Internet-5e éd. Dunod, 2020.

[151] T. A. Ghaleb and S. Mohammed, “Novel scheme for labeling XML trees based on bits-masking and logical matching,” in 2013 World Congress on Computer and Information Technology (WCCIT), 2013, pp. 1–5.

[152] L. George, HBase: the definitive guide: random access to your planet-size data. O’Reilly Media, Inc., 2011.

[153] A. Lakshman and P. Malik, “Cassandra: structured storage system on a p2p network,” in Proceedings of the 28th ACM symposium on Principles of distributed computing, 2009, pp. 5–5.

[154] A. Lakshman and P. Malik, “Cassandra: a decentralized structured storage system,” ACM SIGOPS Oper. Syst. Rev., vol. 44, no. 2, pp. 35–40, 2010.

[155] G. P. Copeland and S. N. Khoshafian, “A decomposition storage model,” Acm Sigmod Rec., vol. 14, no. 4, pp. 268–279, 1985.

[156] D. Abadi, P. Boncz, S. H. Amiato, S. Idreos, and S. Madden, The design and implementation of modern column-oriented database systems. Now Hanover, Mass., 2013.

[157] A. Nayak, A. Poriya, and D. Poojary, “Type of NOSQL databases and its comparison with relational databases,” Int. J. Appl. Inf. Syst., vol. 5, no. 4, pp. 16–19, 2013.

[158] S. Mukherjee, “The battle between NoSQL Databases and RDBMS,” Available SSRN 3393986, 2019.

[159] Mehul Nalin Vora, “Hadoop-HBase for large-scale data,” in Proceedings of 2011 International Conference on Computer Science and Network Technology, Dec. 2011, vol. 1, pp. 601–605. doi: 10.1109/ICCSNT.2011.6182030.

[160] K. Bakshi, “Considerations for big data: Architecture and approach,” in 2012 IEEE Aerospace Conference, Mar. 2012, pp. 1–7. doi: 10.1109/AERO.2012.6187357.

[161] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, “ZooKeeper: Wait-free Coordination for Internet-scale Systems.,” in USENIX annual technical conference, 2010, vol. 8, no. 9.

[162] A. Khurana, “Introduction to HBase schema design,” White Pap. Cloudera, 2012.

[163] A. Hendawi et al., “Benchmarking large-scale data management for Internet of Things,” J. Supercomput., vol. 75, no. 12, pp. 8207–8230, 2019.

[164] E. Conte and C. M. Cuza, “Column-based Databases and HBase,” 2018.

[165] R. Choudhry, HBase High Performance Cookbook. Packt Publishing Ltd, 2017.

[166] M. S. Kumar, “Comparison of NoSQL Database and Traditional Database-An emphatic analysis,” JOIV Int. J. Inform. Vis., vol. 2, no. 2, pp. 51–55, 2018.

[167] L. Perkins, E. Redmond, and J. Wilson, Seven databases in seven weeks: a guide to modern databases and the NoSQL movement. Pragmatic Bookshelf, 2018.

[168] H. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker, “Map-reduce-merge: simplified relational data processing on large clusters,” in Proceedings of the 2007 ACM SIGMOD international conference on Management of data, 2007, pp. 1029–1040.

[169] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, “Pig latin: a not-so-foreign language for data processing,” in Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008, pp. 1099–1110.

[170] A. Thusoo et al., “Hive: a warehousing solution over a map-reduce framework,” Proc. VLDB Endow., vol. 2, no. 2, pp. 1626–1629, 2009.

[171] M. Kornacker et al., “Impala: A Modern, Open-Source SQL Engine for Hadoop.,” in Cidr, 2015, vol. 1, p. 9.

[172] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian, “A comparison of join algorithms for log processing in mapreduce,” in Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, 2010, pp. 975–986.

[173] J. Dittrich, J.-A. Quiané-Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad, “Hadoop++: making a yellow elephant run like a cheetah (without it even noticing),” Proc. VLDB Endow., vol. 3, no. 1–2, pp. 515–529, 2010.

[174] A. F. Al-Badarneh and S. A. Rababa, “An analysis of two-way equi-join algorithms under MapReduce,” J. King Saud Univ. - Comput. Inf. Sci.. May 2020. doi: 10.1016/j.jksuci.2020.05.004.

[175] Z. Bousalem. I. El Guabassi. and I. Cherti. “Toward Adaptive and Reusable Learning Content Using XML Dynamic Labeling Schemes and Relational Databases.” in International Conference on Advanced Intelligent Systems for Sustainable Development. 2018. pp. 787–799.

[176] J. Darmont. “Data-Centric Benchmarking.” in Encyclopedia of Information Science and Technology. Fourth Edition. IGI Global. 2018. pp. 1772–1782.

[177] R. Han. X. Lu. and J. Xu. “On big data benchmarking.” in Workshop on Big Data Benchmarks. Performance Optimization. and Emerging Hardware. 2014. pp. 3–18.

[178] R. Han. L. K. John. and J. Zhan. “Benchmarking big data systems: A review.” IEEE Trans. Serv. Comput.. vol. 11. no. 3. 2018.

[179] A. Ghazal et al.. “BigBench: towards an industry standard benchmark for big data analytics.” in Proceedings of the 2013 ACM SIGMOD international conference on Management of data. 2013. pp. 1197–1208.

[180] L. Wang et al.. “Bigdatabench: A big data benchmark suite from internet services.” in High Performance Computer Architecture (HPCA). 2014 IEEE 20th International Symposium on. 2014. pp. 488–499.

[181] B. F. Cooper. A. Silberstein. E. Tam. R. Ramakrishnan. and R. Sears. “Benchmarking cloud serving systems with YCSB.” in Proceedings of the 1st ACM symposium on Cloud computing. 2010. pp. 143–154.

[182] D. Bermbach. E. Wittern. and S. Tai. “Getting Started in Cloud Service Benchmarking.” in Cloud Service Benchmarking. Springer. 2017. pp. 151–153.

[183] H. Matallah. G. Belalem. and K. Bouamrane. “Evaluation of NoSQL Databases: MongoDB. Cassandra. HBase. Redis. Couchbase. OrientDB.” Int. J. Softw. Sci. Comput. Intell. IJSSCI. vol. 12. no. 4. pp. 71–91. 2020.

[184] G. Haughian. R. Osman. and W. J. Knottenbelt. “Benchmarking replication in cassandra and mongodb nosql datastores.” in International Conference on Database and Expert Systems Applications. 2016. pp. 152–166.

[185] F. Bajaber. S. Sakr. O. Batarfi. A. Altalhi. and A. Barnawi. “Benchmarking big data systems: A survey.” Comput. Commun.. vol. 149. pp. 241–251. 2020.

[186] P. Martins. M. Abbasi. and F. Sá. “A study over NoSQL performance.” in World Conference on Information Systems and Technologies. 2019. pp. 603–611.

[187] V. Abramova. J. Bernardino. and P. Furtado. “Which nosql database? a performance overview.” Open J. Databases OJDB. vol. 1. no. 2. pp. 17–24. 2014.

[188] A. W. Yassien and A. F. Desouky. “RDBMS. NoSQL. Hadoop: A Performance-Based Empirical Analysis.” in Proceedings of the 2nd Africa and Middle East Conference on Software Engineering. 2016. pp. 52–59.

[189] H. Matallah. G. Belalem. and K. Bouamrane. “Experimental comparative study of NoSQL databases: HBASE versus MongoDB by YCSB.” Comput Syst Sci Eng. vol. 32. no. 4. pp. 307–317. 2017.

[190] X. Tian. R. Han. L. Wang. G. Lu. and J. Zhan. “Latency critical big data computing in finance.” J. Finance Data Sci.. vol. 1. no. 1. pp. 33–41. 2015.

[191] V. Abramova. J. Bernardino. and P. Furtado. “Experimental evaluation of NoSQL databases.” Int. J. Database Manag. Syst.. vol. 6. no. 3. p. 1. 2014.

[192] “International Data Corporation. 2018. Data Age 2025. report.” [Online]. Available: https://www.import.io/wp-content/uploads/2017/04/Seagate-WP-DataAge2025-March-2017.pdf

[193] H. R. Vyawahare. P. P. Karde. and V. M. Thakare. “A hybrid database approach using graph and relational database.” in 2018 International Conference on Research in Intelligent and Computing in Engineering (RICE). 2018. pp. 1–4.

[194] K. Sahatqija. J. Ajdari. X. Zenuni. B. Raufi. and F. Ismaili. “Comparison between relational and NOSQL databases.” in 2018 41st international convention on information and communication technology. electronics and microelectronics (MIPRO). 2018. pp. 0216–0221.

[195] C.-H. Lee and Y.-L. Zheng. “Automatic SQL-to-NoSQL schema transformation over the MySQL and HBase databases.” in 2015 IEEE International Conference on Consumer Electronics-Taiwan. 2015. pp. 426–427.

[196] G. Zhao. Q. Lin. L. Li. and Z. Li. “Schema conversion model of SQL database to NoSQL.” in 2014 Ninth International Conference on P2P. Parallel. Grid. Cloud and Internet Computing. 2014. pp. 355–362.

[197] D. Serrano. D. Han. and E. Stroulia. “From Relations to Multi-dimensional Maps: Towards an SQL-to-HBase Transformation Methodology.” in 2015 IEEE 8th International Conference on Cloud Computing. Jun. 2015. pp. 81–89. doi: 10.1109/CLOUD.2015.21.

[198] H.-J. Kim. E.-J. Ko. Y.-H. Jeon. and K.-H. Lee. “Migration from RDBMS to Column-Oriented NoSQL: Lessons Learned and Open Problems.” in Proceedings of the 7th International Conference on Emerging Databases. Singapore. 2018. pp. 25–33. doi: 10.1007/978-981-10-6520-0_3.

[199] J. Yoo. K.-H. Lee. and Y.-H. Jeon. “Migration from RDBMS to NoSQL Using Column-Level Denormalization and Atomic Aggregates..” J. Inf. Sci. Eng.. vol. 34. no. 1. 2018.

[200] R. Ouanouki. “Extracting rules to help in the conversion of a relational database to NoSQL (column-oriented) database technology.” PhD Thesis. École de technologie supérieure. 2020.

[201] R. Ouanouki. A. April. A. Abran. A. Gomez. and J.-M. Desharnais. “Toward building RDB to HBase conversion rules.” J. Big Data. vol. 4. no. 1. pp. 1–21. 2017.

[202] G. Zhao. L. Li. Z. Li. and Q. Lin. “Multiple nested schema of HBase for migration from SQL.” in 2014 Ninth International Conference on P2P. Parallel. Grid. Cloud and Internet Computing. 2014. pp. 338–343.

[203] N. Dimiduk. A. Khurana. M. H. Ryan. and M. Stack. HBase in action. Manning Shelter Island. 2013.

[204] D. Salmen. T. Malyuta. R. Fettersand. and A. Norbert. Cloud data structure diagramming techniques and design patterns. 2009.

[205] K. S. Aggour. Intelligent and Scalable Algorithms for Canonical Polyadic Decomposition. Rensselaer Polytechnic Institute. 2019.

[206] K. Gandhi and A. Kumar. “Comparative Analysis of NoSQL Databases - New Generation of Data Storage Model.” Int. J. Sci. Res. Dev.. vol. 5. no. 4. pp. 1109–1114. Jul. 2017.

Annexes

Annexe 1

Commandes YCSB

Nous présentons ci-dessous quelques exemples des commandes YCSB que nous avons développé pour implémenter les tests que nous avons proposé dans les perspectives du chapitre V

1. Test1

- **Mode monoposte**
  - **HBase**

//---------- Table 4M ----------

//---------- 5k Operations ----------

//******** run Workload A ********

```
bin/ycsb run hbase12 -P workloads/workloada -p columnfamily=cf4 -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=5000 -p recordcount=4000000 > E1_1Node_run_workloadA_Op-5k_Rcr-4M_HBase.txt
```

//---------- 10k Operations ----------

//******** run Workload A ********

```
bin/ycsb run hbase12 -P workloads/workloada -p columnfamily=cf4 -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=10000 -p recordcount=4000000 > E1_1Node_run_workloadA_Op-10k_Rcr-4M_HBase.txt
```

//---------- 100k Operations ----------

//******** run Workload A ********

```
bin/ycsb run hbase12 -P workloads/workloada -p columnfamily=cf4 -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=100000 -p recordcount=4000000 > E1_1Node_run_workloadA_Op-100k_Rcr-4M_HBase.txt
```

//------------------- 1M Operation -------------------

//******** run Workload A ********

```
bin/ycsb run hbase12 -P workloads/workloada -p columnfamily=cf4 -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=1000000 -p recordcount=4000000 > E1_1Node_run_workloadA_Op-1M_Rcr-4M_HBase.txt
```

- **MySQL**

//------------------- Table 4M -----------------

//------------------- 5k Operations -------------------

//******** run Workload A

```
bin/ycsb run jdbc -P workloads/workloada -P ~/YCSB/jdbc/src/main/conf/db.properties -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=5000 -p
```

//------------------- 10k Operations -------------------

//******** run Workload A

```
bin/ycsb run jdbc -P workloads/workloada -P ~/YCSB/jdbc/src/main/conf/db.properties -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=10000 -p recordcount=4000000 > E1_1Node_run_workloadA_Op-10k_Rcr-4M_MySQL.txt
```

//------------------- 100k Operations -------------------

//******** run Workload A

```
bin/ycsb run jdbc -P workloads/workloada -P ~/YCSB/jdbc/src/main/conf/db.properties -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=100000 -p recordcount=4000000 > E1_1Node_run_workloadA_Op-100k_Rcr-4M_MySQL.txt
```

//---------- 1M Operations ----------

//******** run Workload A

```
bin/ycsb run jdbc -P workloads/workloada -P ~/YCSB/jdbc/src/main/conf/db.properties -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=1000000 -p recordcount=4000000 > E1_1Node_run_workloadA_Op-1M_Rcr-4M_MySQL.txt
```

- **Mode distribué**
  - **HBase**
    - **Nœud 1**

//---------- 5k Operations ----------

//******** run Workload A

```
bin/ycsb run hbase12 -P workloads/workloada -p columnfamily=cf4 -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=1250 -p recordcount=1000000 > E1_4Nodes_N1_run_workloadA_Op-5k_Rcr-4M_HBase.txt
```

//---------- 10k Operations ----------

//******** run Workload A

bin/ycsb run hbase12 -P workloads/workloada -p columnfamily=cf4 -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=2500 -p recordcount=1000000 > E1_4Nodes_N1_run_workloadA_Op-10k_Rcr-4M_HBase.txt

- **Nœud 2**

//------------------- 5k Operations -------------------

//******** run Workload A

```
bin/ycsb run hbase12 -P workloads/workloada -p columnfamily=cf4 -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=1250 -p recordcount=1000000 > E1_4Nodes_N2_run_workloadA_Op-5k_Rcr-4M_HBase.txt
```

//------------------- 10k Operations -------------------

//******** run Workload A

```
bin/ycsb run hbase12 -P workloads/workloada -p columnfamily=cf4 -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=2500 -p recordcount=1000000 > E1_4Nodes_N2_run_workloadA_Op-10k_Rcr-4M_HBase.txt
```

- **Nœud 3**

//------------------- 5k Operations -------------------

//******** run Workload A

```
bin/ycsb run hbase12 -P workloads/workloada -p columnfamily=cf4 -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=1250 -p recordcount=1000000 > E1_4Nodes_N3_run_workloadA_Op-5k_Rcr-4M_HBase.txt
```

//------------------- 10k Operations -------------------

//******** run Workload A

```
bin/ycsb run hbase12 -P workloads/workloada -p columnfamily=cf4 -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=2500 -p recordcount=1000000 > E1_4Nodes_N3_run_workloadA_Op-10k_Rcr-4M_HBase.txt
```

- **Nœud 4**

//------------------- 5k Operations -------------------

//******** run Workload A

```
bin/ycsb run hbase12 -P workloads/workloada -p columnfamily=cf4 -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=1250 -p recordcount=1000000 > E1_4Nodes_N4_run_workloadA_Op-5k_Rcr-4M_HBase.txt
```

//---------- 10k Operations ----------

//******** run Workload A

```
bin/ycsb run hbase12 -P workloads/workloada -p columnfamily=cf4 -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=2500 -p recordcount=1000000 > E1_4Nodes_N4_run_workloadA_Op-10k_Rcr-4M_HBase.txt
```

- **MySQL**
  - **Nœud 1**

//---------- 5k Operations ----------

//******** run Workload A

```
bin/ycsb run jdbc -P workloads/workloada -P ~/YCSB/jdbc/src/main/conf/db.properties -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=1250 -p recordcount=1000000 > E1_4Nodes_N1_run_workloadA_Op-5k_Rcr-4M_MySQL.txt
```

//---------- 10k Operations ----------

//******** run Workload A

```
bin/ycsb run jdbc -P workloads/workloada -P ~/YCSB/jdbc/src/main/conf/db.properties -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=2500 -p recordcount=1000000 > E1_4Nodes_N1_run_workloadA_Op-10k_Rcr-4M_MySQL.txt
```

  - **Nœud 2**

//---------- 5k Operations ----------

//******** run Workload A

```
bin/ycsb run jdbc -P workloads/workloada -P ~/YCSB/jdbc/src/main/conf/db.properties -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=1250 -p recordcount=1000000 > E1_4Nodes_N2_run_workloadA_Op-5k_Rcr-4M_MySQL.txt
```

//---------- 10k Operations ----------

//******** run Workload A

```
bin/ycsb run jdbc -P workloads/workloada -P ~/YCSB/jdbc/src/main/conf/db.properties -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=2500 -p recordcount=1000000 > E1_4Nodes_N2_run_workloadA_Op-10k_Rcr-4M_MySQL.txt
```

- **Nœud 3**

//---------- 5k Operations ----------

//******** run Workload A

```
bin/ycsb run jdbc -P workloads/workloada -P ~/YCSB/jdbc/src/main/conf/db.properties -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=1250 -p recordcount=1000000 > E1_4Nodes_N3_E1_4Nodes_N3_run_workloadA_Op-5k_Rcr-4M_MySQL.txt
```

//---------- 10k Operations ----------

//******** run Workload A

```
bin/ycsb run jdbc -P workloads/workloada -P ~/YCSB/jdbc/src/main/conf/db.properties -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=2500 -p recordcount=1000000 > E1_4Nodes_N3_E1_4Nodes_N3_run_workloadA_Op-10k_Rcr-4M_MySQL.txt
```

- **Nœud 4**

//---------- 5k Operations ----------

//******** run Workload A

```
bin/ycsb run jdbc -P workloads/workloada -P ~/YCSB/jdbc/src/main/conf/db.properties -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=1250 -p recordcount=1000000 > E1_4Nodes_N4_run_workloadA_Op-5k_Rcr-4M_MySQL.txt
```

//---------- 10k Operations ----------

//******** run Workload A

```
bin/ycsb run jdbc -P workloads/workloada -P ~/YCSB/jdbc/src/main/conf/db.properties -p table=usertable4M -p zeropadding=8 -p insertorder=ordered -p operationcount=2500 -p recordcount=1000000 > E1_4Nodes_N4_run_workloadA_Op-10k_Rcr-4M_MySQL.txt
```