UNIVERSITÉ DU QUÉBEC À MONTRÉAL

COMPRESSION DANS LES ENTREPÔTS DE DONNÉES POUR L'AMÉLIORATION DES PERFORMANCES

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE DE GESTION

PAR

DJAMEL GARAR

JANVIER 2013

REMERCIEMENTS

Je tiens à remercier mon Directeur de recherche M. Daniel Lemire, professeur et chercheur du centre de recherche TELUQ, pour sa disponibilité constante et ses conseils pratiques qui m'ont aidé à réaliser mon mémoire. Aussi, son expérience dans la recherche m'a permis d'apprendre plus dans le domaine d'entrepôt de données.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

Tableau Page

LISTE DES ABRÉVIATIONS

CPU	Central Processing Unit
DBMS	Data base management system
ED	Entrepôt de données (Data Warehouse)
ETC	Extraction transformation chargement
OLAP	On line Analytical Processing (traitement analytique en ligne)
OLTP	On line Transaction Processing (traitement transactionnel en ligne)
RAM	Random Access Memory
SAP	Systems, Applications and Products for data processing
SGBDR	Système de gestion de bases de données relationnelles
SQL	Structured Query Language
SSB	Star Schema Benchmark
TI	Technologies information
XML	Extensible Markup Language

RÉSUMÉ

Les entrepôts de données jouent un rôle important dans la collecte et l'archivage d'une grande masse d'informations. Ces dernières sont utilisées dans la gestion et la prise des décisions pour des affaires stratégiques de l'entreprise. Cependant, l'exécution des requêtes complexes dans une grande masse d'information dégrade les performances du système d'entrepôt de données, dont la vitesse d'exécution des requêtes.

Une des techniques les plus répandues pour remédier au problème précédent est de mettre en place un algorithme de compression de données. En effet, la compression de données permet d'une part de réduire le volume de données d'une table et d'autre part de charger et de traiter beaucoup des données en mémoire centrale et évite l'accès fréquent au disque de l'ordinateur.

Aujourd'hui, il existe plusieurs systèmes de gestion de base de données qui intègrent différents algorithmes de compression de données. La plupart de ces algorithmes convergent vers une technique commune basée sur l'utilisation d'un dictionnaire de données. Ce dernier permet d'enregistrer une valeur unique correspondante aux données répétitives trouvées dans la table de données.

Notre recherche dans ce mémoire vise, premièrement, à exploiter les algorithmes de compressions en particulier l'algorithme de compression de base de données Oracle; deuxièmement, à proposer un nouveau prototype de compression de données inspiré de l'approche Oracle. Ce prototype introduit un nouveau concept d'un dictionnaire hiérarchique. Ce dernier est défini par une structure hiérarchique contenant un super dictionnaire de données relié à plusieurs dictionnaires de données. Le super dictionnaire a pour rôle d'enregistrer toutes les valeurs communes entre les dictionnaires. La mise en œuvre de ce nouveau prototype a pour but de développer les techniques de compression de données et d'améliorer les performances de l'entrepôt de données.

Mots-clés
Entrepôt de données (ACM 98, H.2.7), Compression (ACM 98, E.4), Dictionnaires (ACM 98, H.3.1), Performances (ACM 98, K.6.2).

INTRODUCTION

1 CONTEXTE GÉNÉRAL DE LA RECHERCHE

Les entrepôts de données (ED) sont des bases de données volumineuses comportant différentes informations utiles pour la prise de décision des affaires de l'entreprise. L'entrepôt de données récupère périodiquement ces données à partir de différents systèmes de production de l'entreprise. L'intégration des données du système de production vers le système d'entrepôt de données est assurée par l'outil ETC (Extraction, Transformation et Chargement). Cet outil permet de nettoyer, de filtrer et de générer un format des données qui peuvent être facilement stockées dans l'entrepôt de données. Ensuite, ces données de l'entrepôt sont utilisées pour l'analyse et la prise de décision.

Généralement, les entrepôts de données sont exploités dans les industries comme la télécommunication, la finance, l'assurance, ou le transport. La Figure 0.1 montre un exemple d'évolution d'un ED d'une compagnie LGR Télécommunication, tel que la prévision d'évolution de la taille de l'entrepôt de données atteindra 1050 téraoctets (To) en 2008 et se rapproche à 3 pétaoctets (Po) en 2012 (Winter 2008).

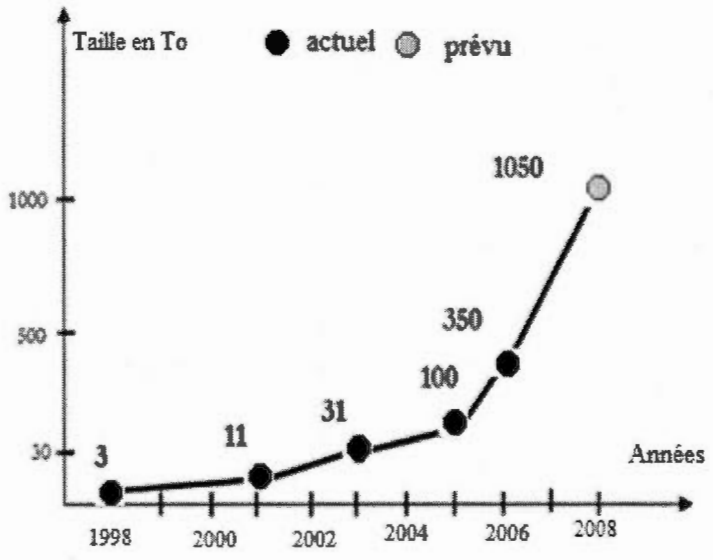


Figure 0.1 Croissance d'ED de la compagnie LGR. Figure adaptée de l'article (Winter 2008).

La croissance et l'évolution rapide du stockage dans l'entrepôt de données s'expliquent par différents phénomènes, dont l'intégration des systèmes d'informations hétérogènes (SGBDR, XML, fichier texte,...), l'évolution de l'Internet (Web 2.0,...), le développement des nouvelles applications informatique et les exigences de l'intelligence d'affaires (concurrence, augmentation le nombre d'entreprises, suivi du marché d'affaires,...). Cette croissance de la taille des entrepôts de données cause souvent un temps de réponse élevé dans l'exécution des requêtes, ce qui retarde l'analyse et la prise de décision des utilisateurs.

Face à ces problèmes qui surviennent dans l'entrepôt de données, différents sujets de recherche furent étudiés comme l'optimisation des requêtes complexes et agrégées (Bellatreche 2003), les index bitmap dans les entrepôts de données (Aouiche, Darmont et al. 2005), le stockage et les performances de lecture/écriture dans l'entrepôt de données (Nicola et Rizvi 2003), la compression dans l'entrepôt de données (Poess et Potapov 2003), l'entrepôt de données en grille (Wehrle 2009), etc.

La compression de données est une technique de codage numérique qui a été étudiée et utilisée depuis longtemps. Elle fait partie du domaine de la théorie de l'information. La théorie de l'information est une branche des mathématiques qui a ses origines dans les travaux de Claude Shanon au Bell Labs dans les années 1940. Il s'est posé diverses questions sur l'information, y compris sur la façon de stocker et de communiquer des messages.

Par le temps, plusieurs algorithmes de compression de données ont été développés dans divers domaines de la technologie d'informations. Aujourd'hui, nous trouvons deux catégories de la compression de données: la compression avec perte et la compression sans perte. La première catégorie (avec perte) est utilisée généralement dans les applications multimédias contenant des images (types jpeg), des sons (types mp3) et des vidéos (types mpeg). L'algorithme de compression avec perte permet de restaurer partiellement le fichier original lors de la décompression. Certains algorithmes utilisent les propriétés de l'oreille et de l'œil humain pour supprimer des informations inutiles lors de la compression des données. La deuxième catégorie (sans perte) est utilisée dans les applications informatiques comme les bases de données, les fichiers du système d'exploitation, etc. L'algorithme de compression sans perte permet de restaurer complètement le fichier original lors de la décompression comme les algorithmes Huffman, RLE, Zip, etc.

Aussi, la compression de données a été développée dans les entrepôts de données. Elle consiste essentiellement la compression les tables volumineuses comme les tables de faits et les tables de dimension de l'entrepôt de données. Celle-ci permet d'améliorer les performances du coût de stockage des données et le temps d'exécution des requêtes. Présentement, nous trouvons ce dernier type de compression dans les systèmes de base de données comme Oracle 11g, Microsoft SQL Server 2008, IBM Database2 version 9.7, etc.

Dans notre projet de recherche, nous étudierons les différentes méthodes et techniques de compression de données, ensuite, nous proposerons un nouveau prototype de compression. Celui-ci sera mis en œuvre et comparé avec les autres systèmes de bases de données.

2 PROBLÉMATIQUE ET QUESTION DE RECHERCHE

Dans les systèmes de traitement transactionnel en ligne (OLTP), les données sont régulièrement mises à jour par les applications de l'entreprise ou les outils d'administration de base de données comme le langage de requête structuré (SQL). Par conséquent, on ne peut bénéficier de la compression de données sur des tables souvent mises à jour, car l'exécution de ces dernières sur des tables compressées peut augmenter le temps d'utilisation du processeur de l'ordinateur (CPU).

Par ailleurs, la compression de données est plus répandue dans les entrepôts de données. En effet, la plupart des tables d'entrepôt de données sont volumineuses et en lectures seules. Ces propriétés des tables ont des avantages dans la compression de l'entrepôt de données afin de réduire le volume de données et améliorer les performances.

Plusieurs algorithmes de compression de l'entrepôt de données ont été développés. Oracle et Microsoft compressent la table de la base de données en plusieurs blocs de dictionnaire ou plusieurs pages de données (Poess et Potapov 2003; Microsoft 2011). IBM et Terardata compressent la table de la base de données en un seul bloc de dictionnaire (IBM 1994; Morris 2002). L'approche de la compression par un seul dictionnaire permet d'améliorer efficacement la compression de données, mais d'un autre côté augmente un peu le coût de traitement de processeur (CPU), car, la compression d'une table entière dans un dictionnaire peut saturer la mémoire cache (Poess et Potapov 2003). Une autre approche a été utilisée pour remédier à l'approche précédente en faisant la compression de la table en plusieurs blocs de données, ce qui peut alléger le stockage des données compressées dans la mémoire cache et améliore la

compression des données. Cependant, la compression de la table en plusieurs blocs individuels ou plusieurs pages individuelles peut ignorer plusieurs valeurs de données communes entre les blocs de dictionnaire, ce qui peut réduire le facteur de la compression de données. De plus, la compression d'une table en plusieurs blocs ou en plusieurs pages peut avoir plusieurs accès de lecture ou d'écriture du disque.

Une nouvelle approche de compression dans l'entrepôt de données a été développée par Abadi et al. (2006). Cette approche étudie et évalue certains algorithmes de compression de données pour les intégrer dans les systèmes de base de données orientée colonnes, en particulier, C-Store. C-Store a été créé par un groupe des développeurs de base de données afin d'optimiser la lecture d'un grand volume de données (Stonebraker, Abadi et al. 2005). Dans la technologie de l'intelligence d'affaires, SAP a utilisé les techniques de compression de données dans la base de données SAP NetWeaver TREX en prenant en compte l'organisation des données, la mémoire centrale et l'architecture des serveurs de bases de données (Lemke, Sattler et al. 2010). D'autres travaux de recherche ont étudié les performances des algorithmes d'agrégats pour le traitement des opérations analytiques en ligne OLAP (Online Analytical Processing). Ces algorithmes utilisent des requêtes d'agrégats exécutées sur un grand volume de données compressées (Li et Srivastava 2002).

Les différentes méthodes et techniques de compression de données citées précédemment ont des avantages et des inconvénients. La problématique est définie par la recherche et l'étude d'un prototype de compression qui permet de développer les algorithmes de compression existants et améliorer les performances de l'entrepôt de données. C'est-à-dire, nous essayons de répondre à la question principale de recherche suivante :

Comment compresser l'entrepôt de données pour améliorer les performances?

À partir de cette question principale de recherche, nous allons étudier les différents algorithmes de compression de données qui nous permettent

d'améliorer et de développer une ou plusieurs techniques de compression de données. Cet énoncé de la problématique est décrit par deux sous-questions de recherche suivantes :

1. *Quelles sont les méthodes de compressions de l'entrepôt de données utilisées dans la littérature ou dans les systèmes des bases de données?*
2. *Comment peut-on améliorer les algorithmes de compression de l'entrepôt de données pour avoir de bonnes performances?*

La deuxième question doit tenir compte de plusieurs paramètres de performance comme le taux de compression, la vitesse d'exécution des requêtes, etc. Le rendement des paramètres de performance dépendent essentiellement du contenu du fichier de données à compresser et le type des requêtes à exécuter (Poess et Potapov 2003; Abadi, Madden et al. 2006; Lemke, Sattler et al. 2010).

3 OBJECTIFS DE RECHERCHE

La compression dans les grandes bases de données permet non seulement de minimiser la taille des données, mais aussi améliorer les performances des requêtes (Ray, Haritsa et al. 1995). À partir de ce principe, les objectifs de recherche permettant de répondre aux questions de recherches précédentes consistent, premièrement, étudier et caractériser les méthodes de compression de l'entrepôt de données utilisées dans la littérature ou dans les systèmes des bases de données; deuxièmement, concevoir et développer un prototype de compression de l'entrepôt de données; troisièmement, exécuter conjointement, notre prototype de compression avec le prototype de compression d'Oracle dans un jeu de données de référence SSB (Start Schema Benchmark). Cette exécution de ces deux prototypes permet de mesurer les performances du taux de compression de données et le temps d'exécution des requêtes; quatrièmement, analyser et comparer conjointement les résultats de la performance de notre prototype de compression avec le prototype de compression d'Oracle.

L'approche de développement de notre prototype de compression a été inspirée de l'approche de compression d'Oracle, c'est-à-dire proposer un prototype de compression nommé par la suite le prototype 3 ayant la même structure de stockage que Oracle. La différence est que le stockage du prototype 3 est structuré par un arbre de dictionnaire hiérarchique. Cette nouvelle approche est conçue dans l'objectif de développer et tester un nouveau prototype de compression d'entrepôt de données. Par conséquent, le prototype 3 a plusieurs avantages suivants :

- Amélioration de la compression des données.
- Chargement plus de données en mémoire centrale.
- Organisation physique des données plus structurées.
- Générer plusieurs variantes de compression entre le super dictionnaire et le dictionnaire de données.

4 STRUCTURE DU MÉMOIRE

Le mémoire est structuré en trois chapitres. Le premier chapitre présente la revue de la littérature qui contient les concepts et l'architecture d'un entrepôt de données, les différents modes de stockage de l'entrepôt de données, les méthodes standards de compression de données comme la méthode de compression de Huffman, la méthode de compression Lempel Ziv Welch ainsi que les techniques de compression présentées dans les différents systèmes de bases de données relationnels. Ensuite, nous décrivons les différents paramètres de performance qui caractérisent les méthodes de la compression des tables dans l'entrepôt de données. Dans le deuxième chapitre, nous présentons les différentes approches de la compression de données par dictionnaire, ensuite nous détaillons les étapes de développement de notre prototype de compression de l'entrepôt de données. Dans le dernier chapitre, nous présentons les résultats de la compression de données basées sur le schéma de référence en étoile (Star Schéma Benchmark) et les résultats de temps de réponse des requêtes exécutées dans l'entrepôt de données.

L'analyse et la comparaison des résultats sont effectuées entre deux prototypes, notre prototype de compression avec le prototype de compression d'Oracle. Cette comparaison permet d'évaluer et d'améliorer notre prototype de compression. Nous donnons à la fin du chapitre, notre point de vue dans cette recherche en montrant nos contributions dans ce mémoire, les limites autour de cette recherche, et en proposant les différentes pistes de recherche qui peuvent être étudiées et développées dans le futur. Nous terminons ce mémoire par une conclusion.

CHAPITRE I

REVUE DE LA LITTÉRATURE

1.1 CONCEPTS ET ARCHITECTURE D'UN ENTREPÔT DE DONNÉES

1.1.1 Définition d'un entrepôt de données

Un entrepôt de données (ED) est une base de données consolidée, provenant toutes ses informations à partir des bases de données de production. Généralement, un ED est utilisé par les applications décisionnelles afin d'analyser la situation de l'entreprise à court terme ou à long terme et de faire un suivi de toutes les activités stratégiques de l'entreprise. Celles-ci sont visualisées graphiquement dans un tableau de bord (dashboard) permettant aux gestionnaires de prendre les décisions en temps opportuns.

Selon la définition d'Inmon (2002), un ED est caractérisé par quatre éléments :

Le premier élément définit des données orientées sujets, c'est-à-dire les données de l'entrepôt sont organisées par des thèmes conçus en fonction des besoins de l'entreprise. Ces thèmes représentent des magasins de données (data mart), par exemple le magasin de données vente.

Le deuxième élément définit des données intégrées, c'est-à-dire les données proviennent de différentes sources hétérogènes et elles sont stockées dans l'entrepôt qu'après la phase ETC. Cette intégration des données dans l'entrepôt permet d'avoir une seule version globale, unique et cohérente pour tous les utilisateurs.

Le troisième élément définit des données historiées, c'est-à-dire les nouvelles données stockées dans l'ED sont conservées sans supprimer les anciennes données. Un paramètre de temps est associé à chaque donnée stockée dans l'ED pour faire la distinction entre les différentes valeurs d'une même information. Par ailleurs, un système de production ne dispose pas cette caractéristique (historié) puisque ces données sont mises à jour régulièrement.

Le quatrième élément définit des données non volatiles, c'est-à-dire les données de l'entrepôt sont principalement utilisées en mode consultation, et elles sont moins fréquemment modifiées ou supprimées par les utilisateurs.

1.1.2 Architecture d'un entrepôt de données

Généralement, un entrepôt de données est composé de trois phases : la phase de préparation (acquisition des données), la phase de stockage des données et la phase de présentation (accès aux données). La Figure 1.1 montre les composantes et les relations entre ces trois phases. Notre projet de recherche se focalise sur la phase de stockage des données.

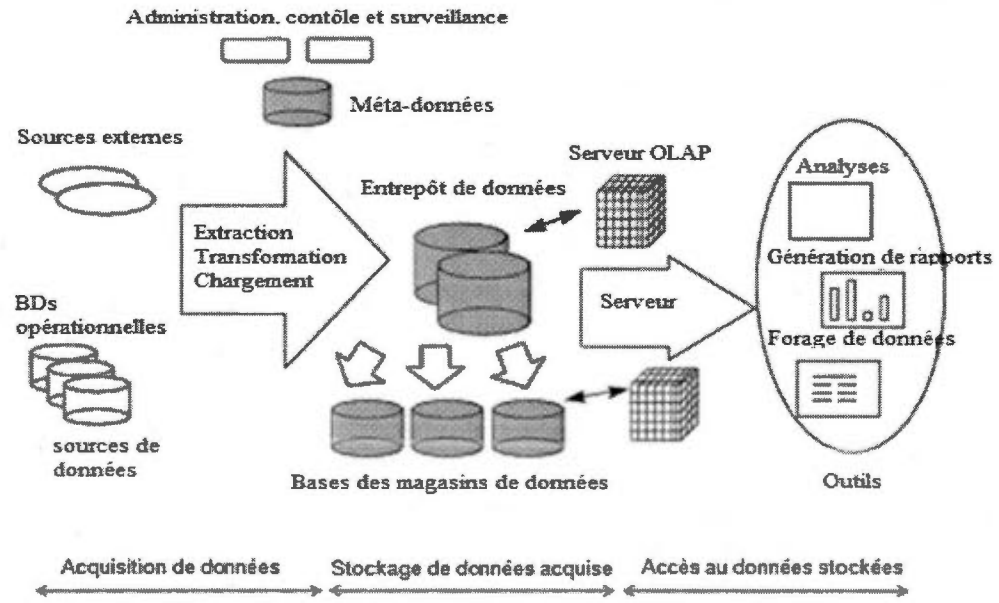


Figure 1.1 Architecture ED. Figure adaptée de l'article (Chaudhuri et Dayal 1997, p. 2).

Phase de préparation: représente la phase la plus importante dans la mise en œuvre d'un ED. Elle est réalisée par l'outil ETC qui permet d'extraire les données à partir des systèmes opérationnels (source de données), de faire des transformations (nettoyage, filtrage, formatage, agrégation, ….) et enfin le chargement les données traitées dans l'ED. Le temps de traitement des données par ETC dépend de la complexité des types de données, les tailles de données et le nombre des sources de données hétérogènes.

Phase de stockage : définit l'espace de stockage des données de l'entrepôt de façon détaillée et agrégée. Elle est représentée par une base de données multidimensionnelle composée des tables de faits et des tables de dimensions.
Le stockage des données est défini au préalable par une architecture de données. Cette architecture peut être conçue de différents modèles. On trouve par exemple, un modèle d'architecture en bus de magasins de données, un modèle d'architecture centralisé, un modèle d'architecture concentrateur et rayons (Hub-and-spoke) et un modèle d'architecture fédéré (Watson et Ariyachandra 2010). Le choix d'une ou plusieurs d'architectures dépend des exigences et les besoins de chaque entreprise.

Généralement, la conception d'une architecture d'ED est basée de deux approches. La première approche est l'approche descendante (top-down) préconisée par Inmon, signifie que les magasins de données sont générés à partir d'un ED centralisé (voir la fig. 1.1). La deuxième approche est l'approche ascendante (bottom-up) préconisée par Kimball, signifie que l'ensemble des magasins de données de l'entreprise constitue l'ED, c'est-à-dire les magasins de données sont conçus de manière progressive pour aboutir à un entrepôt de données structuré par l'un des trois modèles, un modèle en étoile, un modèle en flocon ou un modèle en constellation (voir les fig. 1.2, 1.3, 1.4)

Phase de présentation : cette phase présente les informations de l'entrepôt de données sous forme d'un tableau de bord de manière conviviale facilitant aux gestionnaires la consultation et l'analyse des données pour la prise des décisions. Elle utilise plusieurs logiciels qui font l'édition des états, la fouille de données, la statistique de données, l'exportation de données, etc.

1.2 STOCKAGE DANS L'ENTREPÔT DE DONNÉES

1.2.1 Stockage multidimensionnel

Un ED est organisé sous forme d'un modèle multidimensionnel de données représenté par des axes d'analyses nommés tables de dimensions et des mesures d'analyses nommées tables de faits. La Figure 1.2 montre un exemple d'un ED de ventes d'une chaîne de magasin.

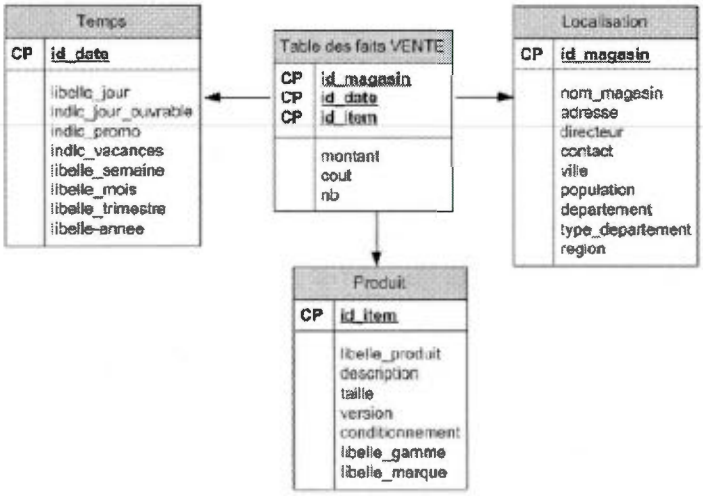


Figure 1.2 Schéma en étoile d'un ED ventes.

Un ED peut être représenté par différents modèles conceptuels OLAP suivants. Ces modèles ont été créés et développés par Kimball et al. (2008) :

Modèle en étoile (star schéma): ce modèle est composé d'une seule table de faits relie par des tables de dimensions (voir la fig. 1.2).

Modèle en flocon de neige (snowflake schema) : ce modèle est une extension du modèle en étoile, c'est-à-dire on peut avoir des relations entre les tables de dimensions (voir la fig. 1.3).

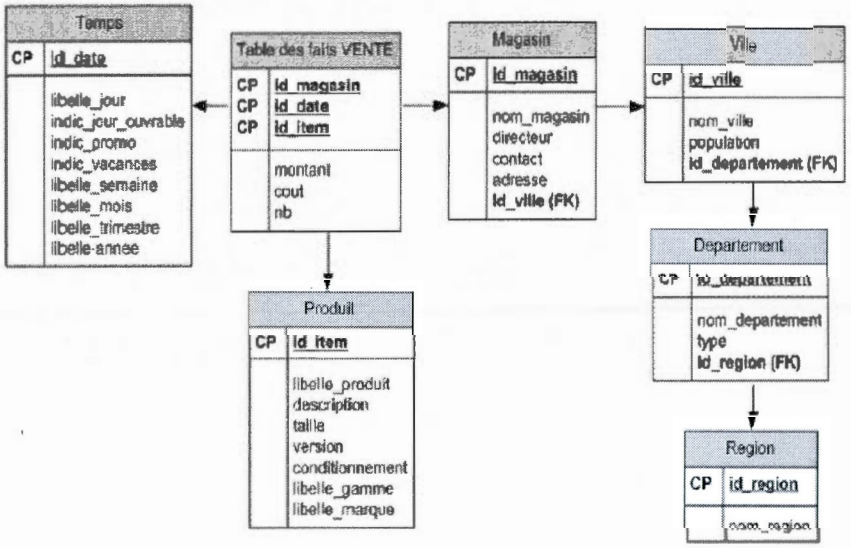


Figure 1.3 Schéma en flocon de l'ED ventes.

Modèle en constellation : ce modèle permet de regrouper plusieurs tables de faits partageant toutes ou une partie de leurs tables de dimensions (voir la fig. 1.4).

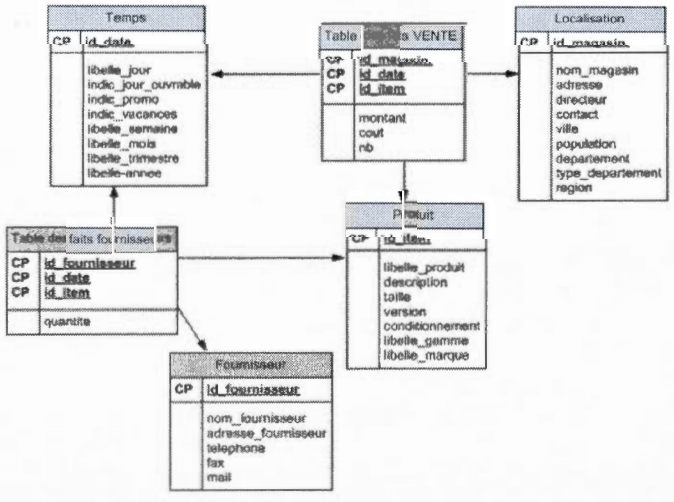


Figure 1.4 Schéma en constellation de l'ED Ventes.

Les trois modèles précédents créent une vue multidimensionnelle afin de faciliter aux gestionnaires la consultation et l'analyse des données de façon transversale. Ils sont utilisés dans les applications de traitement analytique en ligne (OLAP). Par contre, les modèles OLTP créent une vue relationnelle pour l'enregistrement et le suivi des activités de l'entreprise. Le Tableau 1.1 décrit une comparaison entre ces deux modèles.

Tableau 1.1 Comparaison entre OLTP et OLAP

Items	OLTP	OLAP
Nature	Transactionnel	Décisionnel
But	Enregistrement et suivi	Analyse et décision
Utilisations	Quotidienne	Aléatoire
Modèle de données	Relationnel (Entités / Relations)	Multidimensionnel (Faits / Dimensions)
Données	Détaillées	Détaillées, agrégées
Requêtes	Simples interrogeant un petit volume de données.	Complexes interrogeant un grand volume de données.

1.2.2 Mode de stockage des données

Le stockage des données dans l'entrepôt peut se faire par l'un des deux modes, le mode de stockage orienté ligne ou le mode de stockage orienté colonne. La définition et la comparaison de ces deux modes de stockage seront présentées dans les sections suivantes.

1.2.2.1 Stockage orienté ligne

Le stockage orienté ligne est le stockage standard, utilisé par la plupart des SGBDR contenant les bases de données de production. Ce type de stockage permet d'enregistrer les valeurs de la table ligne par ligne.

D'après la table des employés montrée par le Tableau 1.2, le stockage en ligne présente les données de la table sous forme d'un vecteur (1, Durant, Jacques, 40000; 2, Dupont, Marie, 50000; 3, Martin, Jeanne, 44000). Nous remarquons que les données du vecteur sont stockées consécutivement avec des types de données différents.

Tableau 1.2 Table des employés

Emp id	Nom	Prénom	Salaire
1	Durant	Jacques	40000
2	Dupont	Marie	50000
3	Martin	Jeanne	44000

1.2.2.2 Stockage orienté colonne

Le stockage orienté colonne ou le fichier de données transposé a été introduit dans les premiers jours du développement de SGBD au début des années 1970. Dans ce mode de stockage, l'institut de statistique du canada a mis en œuvre dans les années 1976 un système RAPID. Ce système permet de traiter et de récupérer des données de recensement des logements et la population canadienne (Turner, Hammond et al. 1979).

Dans le domaine commercial, Sybase IQ a été le seul le produit disponible pendant plusieurs années dans la catégorie des SGBD orientés colonnes. Cependant, cela a été changé au cours des dernières années avec l'apparition de l'open source et de nombreux produits commerciaux. On trouve par exemple les SGBD orientés colonnes de types commerciaux (Infobright, Vertica, Sysbase IQ, ...) et de types open source (Monet DB,...).

Le stockage orienté colonne permet de stocker les données de la table colonne par colonne. Par exemple, d'après la table des employés montrée par le Tableau 1.2, le stockage en colonne présente les données de la table sous forme d'un vecteur (1, 2, 3; Durant, Dupont, Martin; Jacques, Marie, Jeanne; 40000, 50000, 44000).

Nous remarquons que les données du vecteur sont stockées consécutivement de façons compatibles (mêmes types de données) colonne par colonne. Cette organisation en colonne facilite la compression de données de l'entrepôt et l'exécution des requêtes d'agrégats, puisque les données d'une table sont stockées colonne par colonne dans le disque.

D'après les résultats du Tableau 1.3, Stonebraker et al. (2005) ont conclu que les DBMS orientées colonnes en particulier C-Store sont plus rapides dans le temps d'exécution des différentes requêtes agrégées par rapport aux DBMS orientées lignes, et ont caractérisé le DBMS orienté colonne par les points suivants :

1. La représentation en colonnes évite la lecture des attributs inutilisés.
2. Le stockage sous forme de projection plutôt qu'une table entière, permet un classement approprié de stockage de multiples colonnes.
3. La représentation en colonnes permet une meilleure compression des données.
4. Les opérateurs des requêtes fonctionnent sur une représentation compressée.

Tableau 1.3 Vitesse des requêtes de 3 DBMS (Stonebraker, Abadi et al. 2005, p. 11)

Requêtes	C-store	Orientée ligne	Orientée colonne
Q1	0.03	6.80	2.24
Q2	0.36	1.09	0.83
Q3	4.90	93.26	29.54
Q4	2.09	722.90	22.23
Q5	0.31	116.56	0.93
Q6	8.50	652.90	32.83
Q7	2.54	265.80	33.24

1.2.2.3 Comparaison entre ces deux modes de stockages

D'après les deux sections précédentes sur le stockage orienté ligne et le stockage orienté colonne, nous pouvons résumer certains éléments qui caractérisent ces deux modes de stockage :

- Les systèmes orientés colonnes sont plus efficaces dans les requêtes d'agrégats contenant un certain nombre de colonnes de la table.
- Les systèmes orientés colonnes sont plus efficaces lorsqu'il y a des mises à jour d'un grand volume de données similaires appliquées dans la même colonne.
- Les systèmes orientés colonnes sont plus efficace dans l'optimisation de l'espace de disque.
- Les systèmes orientés lignes sont plus efficace dans la lecture ou l'écriture d'une ligne. En effet, cette ligne peut être traitée avec un seul accès de disque.

De plus, les systèmes de base de données orientés lignes sont plus avantageux pour les applications OLTP car ces applications exécutent plusieurs requêtes simples de mises à jour interactives. Par opposition, les systèmes de base de données orientés colonnes sont plus avantageux pour les applications d'entrepôts de données, l'intelligence d'affaires et les supports d'aide à la décision, car ces applications exécutent des requêtes complexes en lecture seule sur un volume de données important (Abadi, Madden et al. 2008). Cependant, certains systèmes orientés lignes de type OLAP comme le système Teradata a fait ses preuves dans la manipulation d'un grand volume de données avec une taille qui peut atteindre un pétaoctets (Po).

1.3 CONCEPTS ET MÉTHODES DE LA COMPRESSION DES DONNÉES

1.3.1 Définition de la compression des données

La compression de données permet de réduire la taille du fichier de données ou la taille d'une table de base de données en appliquant un algorithme de compression. Ces données peuvent être différents types d'informations comme de type texte, numérique, date, image, vidéo son, etc. Par conséquent, cette compression de données a pour avantage de stocker les données compressées dans un espace mémoire utilisant moins de bits. La Figure 1.5 montre un diagramme simplifié de compression et de décompression des données.

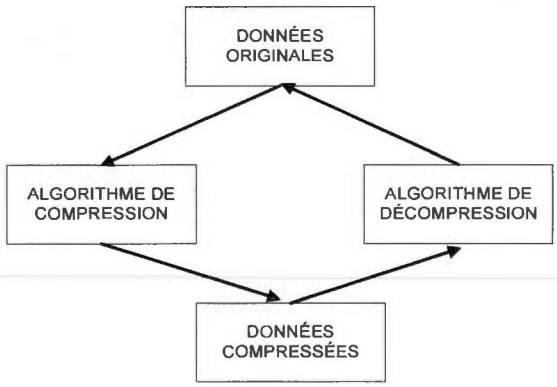


Figure 1.5 Diagramme de fonctionnement de la compression.

Dans notre contexte, nous traitons la compression de données sur des tables d'une base de données. Par conséquent, la technique la plus utilisée dans la compression d'une table de base de données est l'emploi d'un dictionnaire. Cette technique permet d'éliminer les données répétitives trouvées dans la table en remplaçant ces données de longueur variable avec des codes entiers de longueur fixe. On obtient par la suite un schéma de compression plus léger (Binnig, Hildenbrand et al. 2009).

Le rôle de la compression de données permet de :

1. Éliminer la répétition des données de la table, ce qui résulte à la réduction de l'espace de stockage de disque.
2. Charger plus de données en mémoire centrale, ce qui résulte à la réduction d'accès de lecture et l'écriture de disque.
3. Exécuter rapidement des requêtes d'agrégats, ce qui résulte à la rapidité dans le temps de réponse des requêtes des utilisateurs.

1.3.2 Méthodes générales de la compression des données

Il existe plusieurs méthodes de compression des données. Les performances de ces méthodes sont distinguées par le taux de compression et la vitesse de compression/décompression des données. On trouve les méthodes de compression les plus simples aux plus complexes. Les plus simples sont ceux qui compressent la répétition des données textuelles comme l'algorithme de compression par plages (Run Length Encoding). Les plus complexes sont les méthodes de codage statistique comme les algorithmes de Huffman et de Shanon/Fano. Ces derniers algorithmes permettent de coder les caractères avec un minimum de bits en fonction de leurs fréquences d'apparition dans les sources données.

Les recherches dans le domaine de compression ont permis de développer d'autres algorithmes, comme l'algorithme de Lempel Ziv Welch. Ce dernier algorithme part du principe que des chaînes de caractères apparaissant régulièrement dans les données sources peuvent être remplacées par des indices représentant leurs adresses dans un dictionnaire de référence construit progressivement. Aussi, on trouve des algorithmes prédictifs qui cherchent à prévoir les caractères futurs en fonction des caractères déjà lus. Les sections suivantes expliquent en détail certains algorithmes.

1.3.2.1 Algorithme Run-Length Encoding (RLE)

L'algorithme RLE représente l'algorithme le plus connu pour les formats de fichier BMP, JPEG (images fixes), MPEG (images vidéo). Le principe de cet algorithme est de détecter la répétition des données dans une séquence d'éléments (par exemple une chaîne de caractères), puis remplacer cette répétition des données par deux informations : un chiffre indiquant le nombre de répétitions et l'information répétée. Par exemple, considérons la chaîne de caractères : aaabbbbcc. La compression de cette chaîne par l'algorithme RLE est : 3a4b2c.

Cependant, cet algorithme dépend fortement de la chaine à compresser. Ce qui résulte parfois à une augmentation de la taille du fichier compressé. Par exemple, considérons la chaîne de caractères : Renault. La compression de cette chaîne par l'algorithme RLE est : 1R1e1n1a1u1l1t.

1.3.2.2 Algorithme de Huffman

L'algorithme de Huffman fait partie un des algorithmes de compression de données sans pertes. Il permet de construire un arbre binaire de façon ascendante en commençant par les feuilles de l'arbre et en remontant progressivement vers la racine. L'arbre binaire sert de générer des codes binaires 0 et 1 pour chaque élément du texte, ensuite cet arbre sera enregistré pour l'utiliser lors de la décompression des données. Cette technique de compression a un avantage de diminuer le nombre de bits utilisés pour coder un message, une chaine de caractères ou un fichier de texte.

De plus, l'algorithme de Huffman est utile dans le fichier contenant des données répétitives, c'est-à-dire, plus la répétition des données est très fréquente, moins on utilisera des bits pour coder les données répétitives.

L'exemple suivant illustre le principe et les étapes de construction d'un arbre binaire de Huffman appliqué sur la chaîne de caractère « toronto ».

La première étape, nous lisons complètement la chaine de caractères toronto, ensuite nous construisons une table de fréquence contenant le nombre de

fréquences de chaque caractère de la chaîne. Ensuite, cette table de fréquence est triée en ordre décroissant de la fréquence comme le montre le Tableau 1.4.

Tableau 1.4 Table de fréquence

Lettre	Fréquence	Probabilité
t	2	28.57%
o	3	42.85%
r	1	14.28%
n	1	14.28%
Total	7	

Tri la table

Lettre	Fréquence	Probabilité
o	3	42.85%
t	2	28.57%
r	1	14.28%
n	1	14.28%

La deuxième étape, nous identifions deux caractères ou deux nœuds qui ont une fréquence plus faible de la table. D'après le Tableau 1.4, nous remarquons que les nœuds n et r ont une fréquence plus faible égale à 1. Ensuite, nous créons un nœud parent N1 de ces deux nœuds avec par exemple le nœud n à droite, et le nœud r à gauche comme le montre la Figure 1.6.

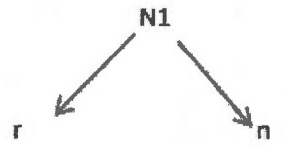


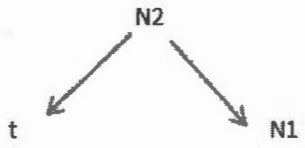
Figure 1.6 Deuxième étape de l'exemple.

La fréquence du nœud N1=2 qui est la somme de deux fréquences de nœuds (n=1, r =1). Ensuite, le nœud N1 est ajouté dans la table de fréquence dans l'ordre de la colonne de fréquence avec la suppression les deux nœuds (n, r). La nouvelle table de fréquence ordonnée est montrée par le Tableau 1.5.

Tableau 1.5 Table de fréquence réordonnée

Lettre	Fréquence	Probabilité
o	3	42.85%
t	2	28.57%
N1	2	28.56%

Cette deuxième étape sera réitérée plusieurs fois jusqu'il n'y a plus de nœuds dans la table de fréquence. Dans ce cas, nous appliquons encore une fois la deuxième étape et en identifions les deux nœuds les plus faibles de la table (N1, t), ensuite nous créons un nœud parent N2 de ces deux nœuds avec par exemple le nœud N1 à droite, et le nœud t à gauche. Le nouveau nœud N2=4 qui est la somme de deux fréquences de nœuds (N1=2, t=2). Ensuite, le nœud N2 est ajouté dans la table de fréquence dans l'ordre de la colonne de fréquence avec la suppression les deux nœuds (N1, t). La nouvelle table de fréquence ordonnée est montrée par la Figure 1.7.



Lettre	Fréquence	Probabilité
N2	4	57.13%
o	3	42.85%

Figure 1.7 Table des fréquences après la 2eme itération.

En dernière étape, un nœud N3 parent des deux nœuds (N2, o) représente la racine de l'arbre binaire de Huffman.

Pour obtenir la représentation binaire de la chaine de caractères toronto, on codifie les branches à droite de l'arbre par un 0 et les branches à gauche de l'arbre par un 1 comme le montre la Figure 1.8.

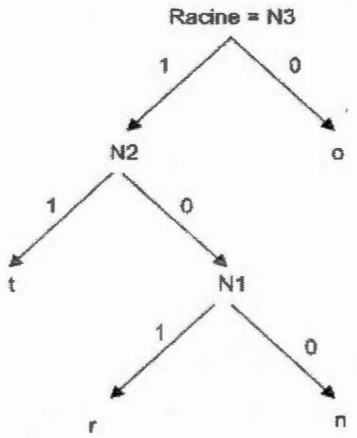


Figure 1.8 Arbre binaire de Huffman.

Pour chaque lettre du Tableau 1.6, on parcourt l'arbre de la Figure 1.8 de la racine jusqu'à la feuille. On obtient la représentation binaire de la chaine de caractères toronto = 11 0 101 0 100 11 0 montrée par le Tableau 1.6.

Tableau 1.6 Codage binaire de la chaîne de caractère toronto

Lettre	Fréquence	Binaire	Fréquence * Binaire
o	3	0	3*1=3
t	2	11	2*2=4
r	1	101	1*3=3
n	1	100	1*3=3
		Total	13

D'après le Tableau 1.6 nous remarquons que le caractère o qui a la plus haute fréquence, a obtenu le moins de bits, et les caractères n ou r qui ont moins de fréquences, ont obtenu plus de bits. Par conséquent, le stockage compressé de la chaine de caractères toronto occupe 13 bits (la dernière ligne du tabl. 1.6) alors que le stockage non compressé de la chaine de caractères toronto occupe 56 bits (7 octets = somme de la colonne fréquence du tabl. 1.6).

Enfin, pour compresser les données par l'algorithme de Huffman, nous lisons le fichier original caractère par caractère et nous écrivons le code Huffman correspondant dans le fichier de destination. Ce dernier fichier contient deux parties : l'entête du fichier et les données compressées. L'entête du fichier contient le nom original du fichier, la taille originale et un arbre binaire (voir la fig. 1.8) qui permet de reconstituer le fichier original. La décompression des données commence par lire l'entête du fichier, puis pour chaque bit lu, nous descendons dans l'arbre jusqu'à atteindre la feuille, nous écrivons le caractère résultant dans le fichier de restitution.

1.3.2.3 Algorithme de Lempel-Ziv-Welch (LZW)

L'algorithme LZW est un algorithme universel de compression sans perte des données crée par Lempel, Ziv, et Welch. La dernière version améliorée a été publiée par Welch en 1984 à partir de l'algorithme LZ78 publié par Lempel et Ziv en 1978. Le principe de cet algorithme exploite la fréquence d'apparition d'une suite de caractères, c'est-à-dire, chaque fois qu'une séquence de caractère apparaît dans les données sources, cette séquence peut être remplacée par un indice d'une valeur de départ 258 représentant leur adresse dans un dictionnaire qui se construira progressivement. Ce dernier est composé initialement par 256 caractères correspondant à la table ASCII (0...255), ainsi que deux autres codes 256, 257 servant de codes de contrôle.

L'intérêt de cette méthode LZW permet de construire un dictionnaire de façon dynamique lors de la compression et de la décompression des données, c'est-à-dire, ce dictionnaire n'est pas stocké dans le fichier compressé.

Le Tableau 1.7 montre les étapes de l'exécution de l'algorithme de compression LZW sur une chaine de caractères « compression-décompression ».

Tableau 1.7 Algorithme de compression LZW

Étape	Code(s) latent	Code lu	Buffer	Dictionnaire	Nombre émis
1	initialisation	c			
2	c	o	co	258=co	c=99
3	o	m	om	259=om	o=111
4	m	p	mp	260=mp	m=109
5	p	r	pr	261=pr	p=112
6	r	e	re	262=re	r=114
7	e	s	es	263=es	e=101
8	s	s	ss	264=ss	s=115
9	s	i	si	265=si	s=115
10	i	o	io	266=io	i=105
11	o	n	on	267=on	o=111
12	n	-	n-	268= n-	n=110
13	-	d	-d	269=-d	- = 45
14	d	é	dé	270=dé	d=100
15	é	c	éc	271= éc	é=130
16	c	o	co	co trouvé à 258	
17	co	m	com	272=com	258=co
18	m	p	mp	mp trouvé à 260	
19	mp	r	mpr	273=mpr	260=mp
20	r	e	re	re trouvé à 262	
21	re	s	res	274=res	262=re
22	s	s	ss	ss trouvé à 264	
23	ss	i	ssi	275=ssi	264=ss
24	i	o	io	io trouvé à 266	

25	io	n	ion	276=ions	266=io
26	n	rien	n		n=110

La description du Tableau 1.7 est décrite par les étapes suivantes :

Étape 1 : le caractère 'c' est lu. Dans cette étape, il n'y a aucun traitement à faire puisque le dictionnaire est vide. Le caractère 'c' sera donc le code latent.

Étape 2 : le caractère 'o' est lu. On met dans la chaîne buffer le code latent + le code lu. La chaine buffer est une variable qui sert de stocker la combinaison de code latent et de code lu. Le buffer vaut donc 'co'. On cherche si la valeur du buffer se trouve déjà dans le dictionnaire. Comme ce n'est pas le cas, on ajoute la chaine 'co' au dictionnaire à l'emplacement 258. Ensuite, on écrit la valeur 99, dont son code latent 'c' dans le fichier compressé, et on place le caractère lu "o" dans le code latent.

Étape 3 : le caractère 'm' est lu. Le buffer contient donc 'om'. Ce buffer n'est pas dans le dictionnaire, on ajoute la chaîne 'om' au dictionnaire à l'emplacement 259. Ensuite, on écrit la valeur 111 dont son code latent 'o' dans le fichier compressé, et le code latent devient 'm'.

Étape 4 : le caractère 'p' est lu. Le buffer contient donc 'mp'. Ce buffer n'est pas dans le dictionnaire, on ajoute la chaîne 'mp' au dictionnaire à l'emplacement 260. Ensuite, on écrit la valeur 109 dont son code latent 'm' dans le fichier compressé, et le code latent devient 'p'. En suivant la même méthode jusqu'à l'étape 15. Dans cette étape 15, nous n'avons pas réduire la taille du fichier, car on n'a pas rencontré une chaine existante dans le dictionnaire.

Étape 16 : le caractère 'o' est lu. Le buffer contient donc 'co'. La chaine 'co' existe déjà dans le dictionnaire à l'emplacement 258 (étape 2). Dans cette étape 16, rien n'est écrit, ni dans le dictionnaire, ni dans le fichier compressé, puis le code latent devient donc 'co'.

Étape 17 : le caractère 'm' est lu. Le buffer contient donc 'com'. Ce buffer n'est pas dans le dictionnaire, on ajoute 'com' au dictionnaire à l'emplacement 272. Ensuite,

on écrit la valeur 258 dont son code latent 'co' dans le fichier compressé, et le code latent devient 'm'. Dans les étapes qui suivent, nous appliquons le même principe de l'étape 16 et l'étape 17 jusqu'à l'étape 26 qui marque la fin de la chaine à compresser. Dans cette dernière étape, le code latent est 'n' qui existe déjà dans le dictionnaire, on écrit sa valeur 110 dans le fichier compressé.

Nous remarquons que la méthode LZW passe par une phase d'apprentissage et de reconnaissance des caractères pour qu'elle devient par la suite efficace, c'est-à-dire, plus la taille du fichier est important, plus la compression est meilleure.

Par comparaison entre la chaine source et la chaine compressée, nous trouvons que la taille de la chaine source = 25 octets dont chaque caractère occupe 1 octet, alors que la taille de la chaine compressée = 20 nombres émis * 9 bits (codage de 512) = 180 bits qui est presque de 23 octets. Donc un gain de compression de 2 octets.

1.3.3 Méthodes de la compression des bases de données

Dans le domaine des bases de données, il existe plusieurs SGBDR qui gèrent le système transactionnel OLTP et le système décisionnel OLAP. Dans le cadre de notre projet, nous choisissons trois systèmes compétitifs : Oracle, DB2, SQL Server que nous détaillerons dans la section suivante.

1.3.3.1 Méthode de la compression d'Oracle

Oracle Corporation est une entreprise américaine spécialisée dans la technologie de l'information (TI), fondée en 1977 par Ellison, Miner, et Oates. La première création technologique d'Oracle a été un SGBDR présenté comme un produit commercial. Les offres d'Oracle en matière des services et des produits ont dépassé le SGBDR en incluant d'autres produits comme les middlewares et les applications. Oracle a grandi rapidement pour élargir son marché d'affaire par

acquisition des nouvelles entreprises comme Peoplesoft, Siebel, Hyperion, BEA et dernièrement Sun Microsystems.

L'algorithme de compression d'Oracle est plus approprié pour les entrepôts de données qui contiennent un grand volume de données historiées avec des requêtes plus complexes. En effet, la compression Oracle permet, premièrement, de réduire le coût d'archivage et de récupération (Backup/Recovery) des données de l'entrepôt; deuxièmement, de stocker des données compressées de manière plus flexible en mémoire centrale; troisièmement, de maintenir ou d'améliorer les performances du système comme le traitement de CPU (Poess et Potapov 2003).

La compression introduite par Oracle dans les versions 9i, 10g et la dernière version 11g, consiste à éliminer les doublons trouvés dans chaque bloc de la table en utilisant une table de symbole. Un bloc de données représente une petite unité de stockage correspondant à un nombre spécifique d'octets de l'espace de la table. La taille du bloc optimale correspond à 8ko dans la plupart des systèmes, cependant, les systèmes d'aide à la décision DSS (Decision Support System) utilisent un bloc plus grand que les systèmes OLTP (Oracle 2011).

Lorsqu'une table est définie compressée par le mot-clé « COMPRESS », Oracle réserve de l'espace dans chaque bloc pour stocker une copie unique des données qui apparaissent dans différents emplacements du bloc. Cet espace réservé est appelé « table de symbole ». Les données ciblées par la compression et qui apparaissent dans les lignes du bloc sont remplacées par des pointeurs qui pointent vers les données de la table symbole. La Figure 1.9 montre un exemple de compression d'un bloc de données Oracle appliqué sur une table de faits facture client.

Dans l'exemple montré par la Figure 1.9, initialement la table facture client est stockée complètement dans un bloc en mode non compressé. Après l'exécution de l'algorithme de compression Oracle, une table de symbole a été créée après l'entête du bloc, ensuite l'algorithme recherche ligne par ligne toutes les données redondantes. D'après la Figure 1.9, nous remarquons que les valeurs "Henry", "Tom", "McGee", "Todd", "1.99" sont des données répétitives. Ces

dernières sont ajoutées dans la table de symbole et elles sont remplacées par des références dans la table client. Aussi, l'algorithme recherche les occurrences des séquences de caractères comme la séquence Tom Smith 1.99 qui est répétée deux fois dans la table. Cette dernière est ajoutée dans la table de symbole et elle est remplacée par une seule référence.

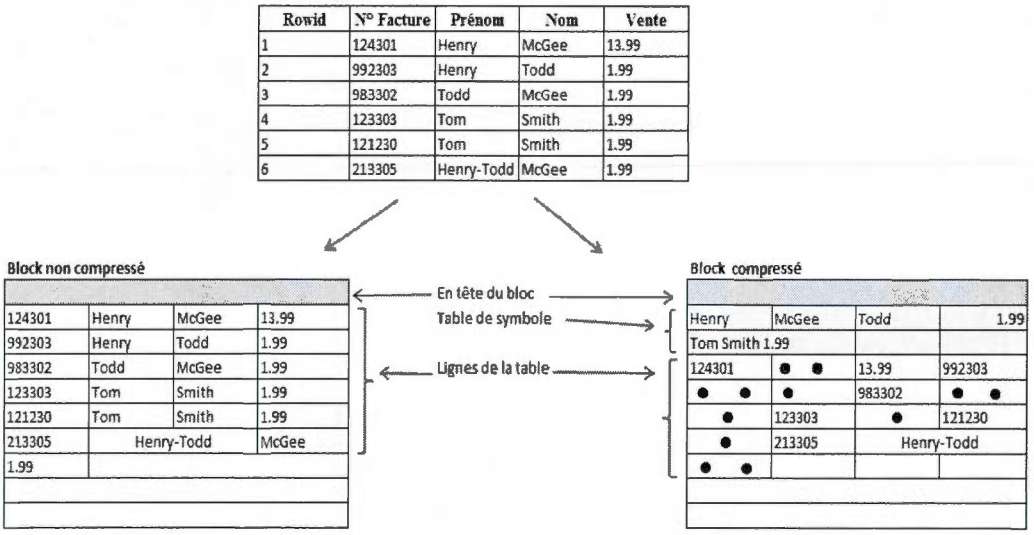


Figure 1.9 Compression d'un bloc de données Oracle. Figure adaptée de l'article (Poess et Potapov 2003, p. 2).

La Figure 1.10 montre les résultats de la comparaison entre les données non compressées et les données compressés de trois tables volumineuses Sales, Weekly_Sales, Weekly_Aggr. La table Sales est une table de faits de 2.5 millions de lignes et 19 colonnes d'un schéma de données en étoile composé de 5 dimensions : Time, Customer, Region, Item, Promotion. Les deux autres tables additionnelles Weekly_Sales, Weekly_Aggr sont des tables d'agrégats dont la première table Weekly_Sales regroupe les lignes de la table Sales pour chaque Item et Customer par nombre de semaines alors que la deuxième table Weekly_Aggr est basé sur la table Weekly_Sales avec agrégation en plus sur le code postal.

Nous remarquons dans la Figure 1.10 que les données de la table Sales sont compressées et passées d'une taille de 27 Go à une taille de 8.6 Go. Cela donne un facteur de compression[1] de 3.1 et une économie d'espace[2] de 68%. De même pour les tables Weekly_Sales et Weekly_Aggr donnent des facteurs de compression respectivement de 2.9 et de 4.0 et une économie d'espace respectivement de 65% et de 75%.

Nous concluons que le pourcentage d'économie de la base de données est de 67%, ce qui résulte à un bénéfice de la compression de 2/3 du total de la base de données.

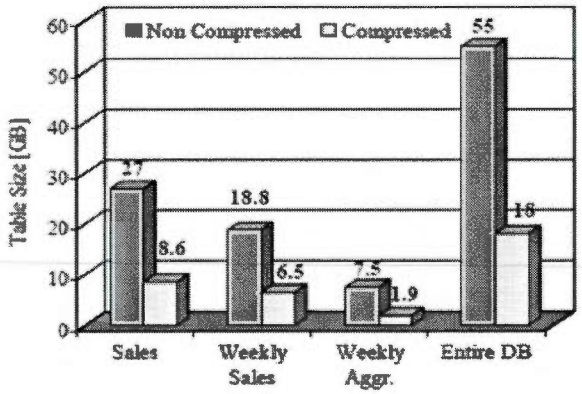


Figure 1.10 Compression tables ED (Poess et Potapov 2003, p. 6).

1.3.3.2 Méthode de la compression SQL Server

Microsoft SQL est un SGBDR développé et commercialisé en 1989 par Microsoft. Ces principaux langages de requête sont T-SQL (Transact SQL) et ANSI SQL. La dernière version la plus stable est SQL Server 2008 Release 2. Cette version inclut plusieurs fonctionnalités comme support XML, les objets Framework .NET, Microsoft Visual Studio, Microsoft Office System.

---

[1] Facteur de compression = #blocs non compressés / #blocs compressés

[2] Économie espace= (#blocs non compressés - #blocs compressés)/ #blocs non compressés * 100

Dans SQL Server, la compression d'une table est appliquée sur plusieurs pages de données. Le terme de page de Microsoft est similaire à celle du bloc d'Oracle, c'est-à-dire, une page est un espace de stockage dont sa taille peut être de 4 ko, de 8 ko, de 16 ko ou de 32 ko.

La compression d'une table de données par page se compose de trois opérations de compressions : la compression en ligne, la compression en préfixe et la compression en dictionnaire (Microsoft 2011).

Compression en ligne : permet de modifier uniquement le format de stockage physique des données qui sont associées à un type de données. Ce type de compression apporte des améliorations sur les enregistrements de la table. Cette amélioration s'applique, premièrement, dans la réduction de la structure de métadonnées associées aux enregistrements de table. Ces métadonnées portent différentes informations sur les caractéristiques des attributs de la table; deuxièmement, dans l'utilisation d'un format de stockage de longueur variable au lieu d'un format de stockage fixe pour les types de données numériques et les chaines de caractères.

Compression en préfixe : permet de stocker dans une zone réservée qui suit immédiatement l'en-tête de page, une copie unique des valeurs répétitives qui apparaissent dans les préfixes d'une colonne. Les valeurs de préfixe répétées dans la colonne sont remplacées par des codes entiers ou des références qui pointent vers la valeur de préfixe correspondant. Toutefois, si la valeur d'une ligne ne correspond pas exactement à la valeur de préfixe sélectionnée, une correspondance partielle peut être indiquée.

Compression en dictionnaire : permet d'appliquer le même principe que la compression en préfixe sauf que la compression en dictionnaire s'effectue dans la totalité de page (plusieurs colonnes). Ce type de compression est un complémentaire de la compression en préfixe.

La Figure 1.11 montre un exemple de la compression en préfixe et en dictionnaire d'une page de données contenant des chaines de caractères.

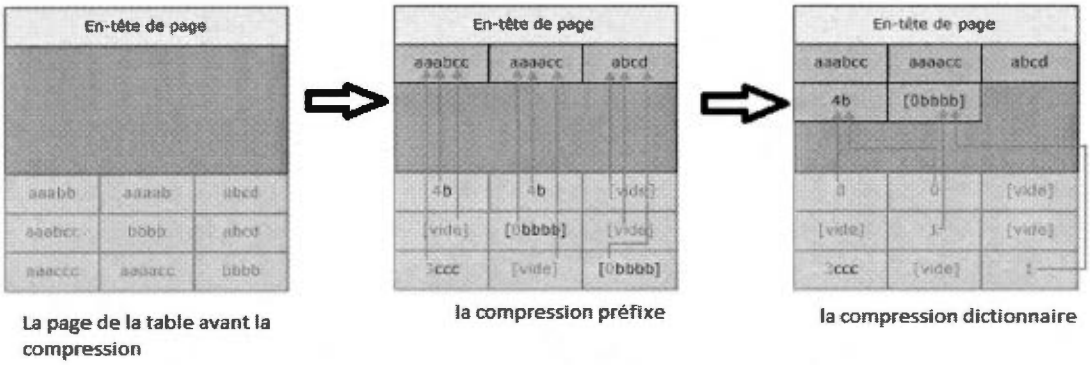


Figure 1.11 Compression d'une page de données (Microsoft 2011).

Aussi, la compression de données en page apporte des améliorations dans l'exécution des requêtes. D'après la Figure 1.12, les huit requêtes exécutées dans un environnement de test des performances des applications, montrent que les temps d'exécution des données compressées en pages sont plus rapides que les temps d'exécution des données non compressées.

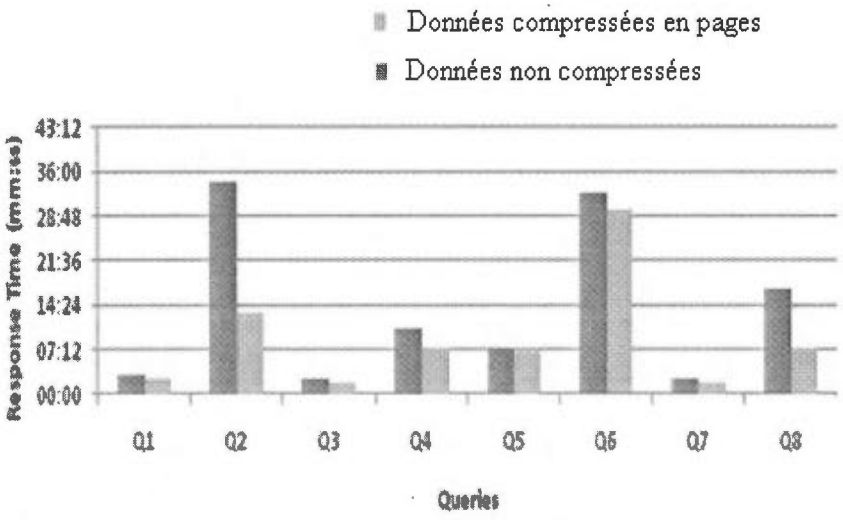


Figure 1.12 Mesure des performances des pages compressées (Mishra 2009).

1.3.3.3 Méthode de la compression IBM DB2

DB2 est un SGBDR développé et commercialisé en 1983 par IBM. Le serveur de base de données universel DB2 est disponible dans une variété des matérielles (mainframes) et des systèmes d'exploitations (Linux, IBM Aix, Solaris, HP-UX, Windows). La version 9.7 est la dernière version du produit DB2 depuis le mai 2009.

La technique de compression utilisée par DB2 est basée sur la compression de la table entière, c'est-à-dire, DB2 examine toutes les lignes de la table pour trouver des données répétitives ou identifier des séquences de caractères similaires. Chaque fois que les données répétitives ou les séquences de caractères répétitives sont trouvées, celles-ci sont remplacées par un symbole et les correspondances sont stockées dans le dictionnaire.

La Figure 1.13 montre un exemple de la compression d'une table employé stockée dans la base de données DB2. Avant la compression, nous remarquons que les données de la table sont enregistrées ligne par ligne dans un vecteur. Après l'application de l'algorithme de compression, nous remarquons que le système a trouvé un suffixe d'une valeur répétitive (opoulos) dans la colonne nom et une séquence de caractère (Whitby, ONT, L4N5R4) répétitive dans la première et la deuxième ligne. Ces deux informations sont enregistrées dans le dictionnaire avec une correspondance par un symbole qui est marqué dans le vecteur de la table des employés.

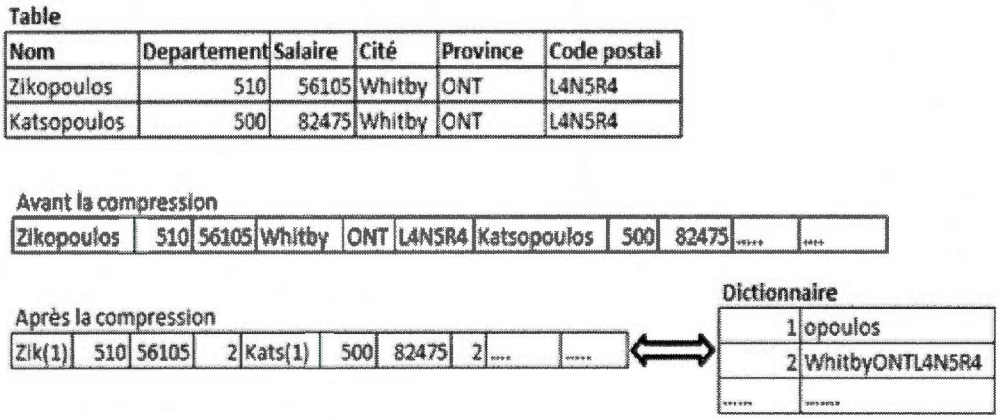


Figure 1.13 Compression d'une table en DB2 (Vincent 2010, p. 3).

D'après la Figure 1.14, les résultats de la compression de 7 tables (A, B, ...G) montrent que toutes les tables ont des taux de compressions positives, presque la moitié des données des tables ont été réduites. Par exemple la table A économise un taux de compression de 63%, la table B de 60%, la table C de 55%, la table D de 56%, la table E de 68%, la table F de 67% et la table G de72%.

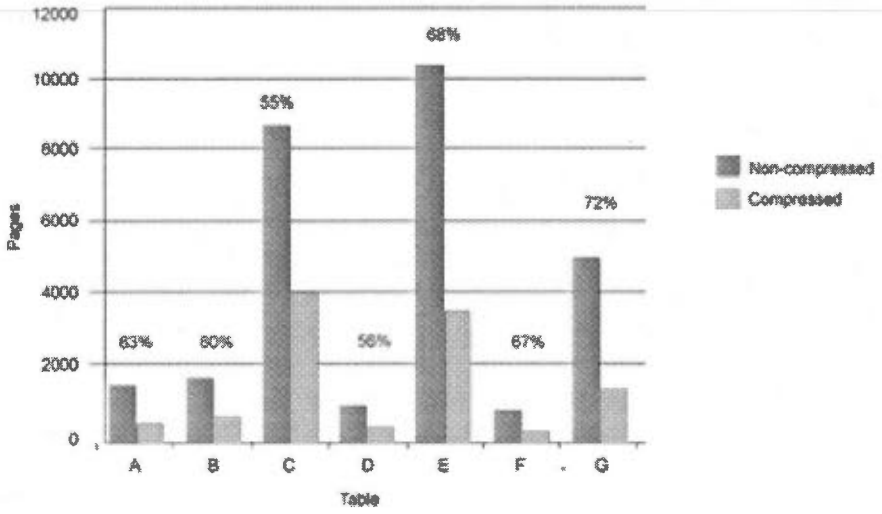


Figure 1.14 Comparaison de la compression des tables DB2 (Ahuja 2006).

1.3.4 Synthèse des méthodes de la compression des données

Les méthodes de compression de données sont différentes entre les systèmes de gestion des bases de données. Cela dépend des techniques de la conception et du développement des algorithmes de compression dans les moteurs des bases de données.

Le Tableau 1.8 montre le taux de compression des tables d'une base de données TPC-H entre les deux systèmes DB2 et Oracle. Ces résultats indiquent clairement que le taux de compression de données des tables pour le système DB2 est supérieur du taux de compression des tables pour le système Oracle (plus de 60% pour DB2 comparé à plus de 40% pour Oracle). En effet, le système DB2 procède à l'élimination des répétitions des données au niveau de la table alors que le système Oracle procède l'élimination des répétitions des données au niveau du bloc, c'est-à-dire Oracle ignore plusieurs valeurs communes entre les blocs de données.

Tableau 1.8 Taux de compression (Eaton 2006; O'Mahony 2009)

	IBM DB2	ORACLE
Table lineitem	58%	38%
Table orders	60%	18%
Base de données entière	59%	29%

On peut déduire des paragraphes précédents que les techniques de compression par bloc ou par page comme Oracle et Microsoft apportent plusieurs avantages principaux: premièrement, le chargement des données compressées dans la mémoire centrale est plus flexible, c'est-à-dire, sans encombrer la mémoire; deuxièmement, le traitement des données compressées dans le CPU est moins coûteux; troisièmement, la décompression des données par bloc ou par page est plus rapide. Le Tableau 1.9 résume une comparaison des méthodes vues précédemment sur les différentes les caractéristiques de compression.

Tableau 1.9 Comparaison des méthodes de compression

Caractéristiques	ORACLE	IBM	MICROSOFT
Compression de données répétitives	Oui	Oui	Oui
Compression des séquences de caractères (motifs)	Oui	Oui	Non
Compression des préfixes	Non	Oui	Oui
Compression des lignes	Oui	Oui	Oui
Compression par table entière	Non	Oui	Non
Compression en bloc ou page	Oui	Non	Oui
Mises à jour des données compressées	Oui	Oui	Oui

Nous concluons d'un autre côté que les résultats de la compression de données dépendent essentiellement du contenu de la table, c'est-à-dire, la compression de données est déterminée par les types d'attributs et la distribution des données au niveau de la table. Nous pouvons énumérer certaines caractéristiques importantes dans la compression des données :

- La table contient plusieurs données répétitives. Ces données peuvent être des valeurs, des préfixes ou des séquences de plusieurs valeurs.
- La table contient une ou plusieurs colonnes de cardinalité faible, c'est-à-dire que la table contient moins des valeurs différentes dans une colonne comme par exemple la colonne sexe (cardinalité 2), la colonne pays (cardinalité < 160), la colonne ville (cardinalité < 1 million), etc.
- La table contient plusieurs valeurs des colonnes de longueur fixe. Ceci permet de créer et de gérer régulièrement l'espace mémoire pour le stockage des données compressées.

1.4 PERFORMANCES DANS L'ENTREPÔT DE DONNÉES

D'après la revue de la littérature sur les algorithmes de compression des données, nous pouvons recenser plusieurs paramètres de performance de l'entrepôt de données. L'amélioration de ces paramètres de performance représente les objectifs de développement de l'algorithme de compression de données. Parmi les paramètres de performance, nous trouvons, premièrement, la réduction de la taille des données, et par conséquent, l'amélioration de l'espace mémoire. Celle-ci nous permet d'archiver dans le disque ou de charger dans la mémoire centrale un grand volume de données; deuxièmement, la réduction du trafic réseau, et par conséquent, l'amélioration de la bande passante du réseau (bandwidth). Celle-ci nous permet de transférer rapidement des données dans plusieurs serveurs de bases de données; troisièmement, la réduction de lecture et de l'écriture (input / output) des données du disque (throughtput). Celle-ci nous permet de minimiser l'accès fréquent au disque; quatrièmement, la rapidité de la vitesse d'exécution des requêtes des utilisateurs. En effet, la sélection des données compressées élimine le parcours de plusieurs valeurs répétitives. Cependant, cette quatrième performance dépend des types des requêtes (requêtes agrégats, requêtes sélectives, ...) à exécuter dans l'entrepôt de données.

Par la suite, nous détaillerons les deux premiers paramètres de performance de l'entrepôt de données à savoir la performance d'espace mémoire et la performance d'exécution des requêtes.

1.4.1 Performance d'espace mémoire

La compression de la table peut réduire significativement l'espace de stockage des données de disque, ainsi que les exigences de la mémoire cache (buffer cache) lors de l'interrogation des tables de la base de données (Poess et Potapov 2003). En effet, la table compressée utilise moins des blocs de données dans le disque, ce qui implique moins des blocs chargés en mémoire centrale.

L'écart entre l'espace de données non compressé et l'espace de données compressé peut se calculer de différente façon. Par exemple, dans le système

Oracle, le calcul s'effectue au niveau des blocs par l'utilisation de la formule (Poess et Potapov 2003) :

*ss = ((#nbc-#bc)/nbc)*100)* tel que ss représente l'économie d'espace (space saving ss) entre les blocs non compressés (nbc) et les blocs compressés (bc). Aussi, dans les méthodes de compression standard, le calcul du taux de compression s'effectue directement par le quotient entre les données compressées et les données non compressées (Pascal 1993) :

Taux de compression = #données compressées / données non compressées.

Dans notre contexte de recherche, nous utilisons une formule simple en faisant la différence entre la taille du fichier non compressé et la taille du fichier compressé.

1.4.2 Performance d'exécution des requêtes

Les requêtes permettent d'interroger un ensemble des données compressées ou non compressées de l'entrepôt. Parfois, l'exécution des données compressées par les requêtes nécessite une opération de décompression des données. En effet, cette dernière opération est nécessaire pour l'affichage des données originales. Cependant, les performances des requêtes reposent essentiellement dans la réduction des coûts de la décompression des données, c'est-à-dire, la compression de données peut être avantageuse si le coût de la décompression de données est minime (Westmann, Kossmann et al. 2000; Lemke, Sattler et al. 2010).

Un modèle de lecture des données compressées durant l'exécution de plusieurs requêtes a été élaboré par Poess et Potapov (2003). Ce modèle est illustré par la Figure 1.15.

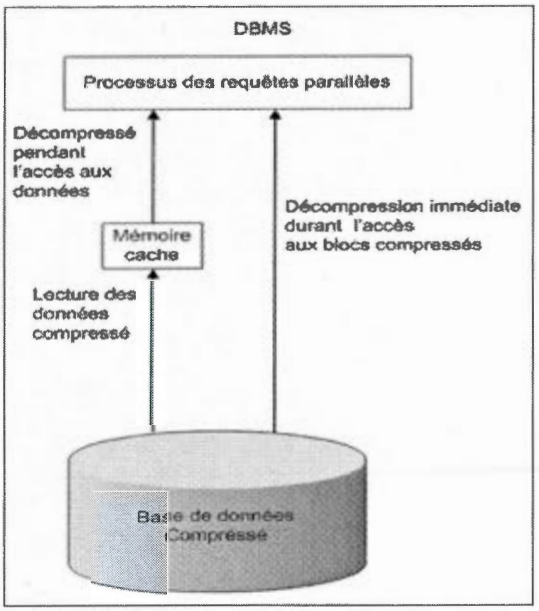


Figure 1.15 Modèle d'accès aux données compressées (Poess et Potapov 2003, p. 5).

Selon la Figure 1.15, il y a trois paramètres de performance d'entrepôt de données compressées: la vitesse d'exécution des requêtes, le temps d'utilisation de la CPU et le temps d'accès aux données. L'appendice B fournit différentes formules pour calculer la valeur de ces trois paramètres de performance. Ces formules ont été extraites de Poess et Potapov (2003).

1.5 CONCLUSION

Dans ce chapitre, nous avons présenté et expliqué les concepts d'ED. Ce dernier est composé de trois phases essentielles : l'acquisition des données, le stockage des données et la présentation des données. Par la suite, nous avons détaillé la deuxième phase de stockage des données, et on a défini le concept multidimensionnel OLAP et ses modèles conceptuels en étoile, en flocon de neige et en constellation. Dans le stockage d'ED, nous avons cité une nouvelle approche de stockage des données basée sur l'approche orientée colonne. Cette dernière approche permet d'optimiser le stockage des données dans les modèles

multidimensionnels OLAP et de minimiser la durée d'exécution des requêtes analytiques de type agrégat. En ce qui concerne notre sujet de recherche, nous avons étudié et caractérisé les différentes méthodes de compression des données. À partir de ces méthodes de compression de données, nous avons synthétisé différents paramètres des performances qui peuvent être examinés après l'exécution de l'algorithme de compression. Ces paramètres de performance représentent les enjeux des méthodes de compression de données. Cependant, pour répondre et atteindre nos objectifs du projet de recherche, nous avons testé notre prototype de compression en choisissant deux paramètres de performance : le taux de compression et la vitesse d'exécution des requêtes. En effet, le taux de compression est le paramètre principal dans l'évaluation de l'algorithme de compression de donnée, alors que la vitesse d'exécution des requêtes est le paramètre le plus utilisé dans les requêtes interrogeant l'entrepôt de données.

L'étude de la revue de la littérature nous a permis de développer un nouveau prototype de compression d'un entrepôt de données. Les différentes étapes de développement et de la mise en œuvre de ce nouveau prototype seront présentées dans le chapitre II.

CHAPITRE II

MÉTHODOLOGIE DE LA RECHERCHE

2.1 INTRODUCTION

Dans ce chapitre, nous présenterons deux sections principales, une section qui décrit les différentes approches de la compression de données et une section qui décrit la démarche de développement de notre prototype de compression. La conception et le développement de notre prototype ont été basés sur les études précédentes effectuées sur les méthodes et les techniques de compression des systèmes de bases de données relationnels Oracle, Microsoft SQL Server et IBM DB2. Les étapes de développement de notre prototype de compression seront présentées dans la section 2.3.

2.2 APPROCHES DE LA COMPRESSION DANS L'ENTREPÔT DE DONNÉES

Les méthodes de la compression des bases de données citées dans la revue de la littérature emploient la structure d'un dictionnaire. Dans ces méthodes de compression, il existe plusieurs approches ou techniques de compression des données. Dans cette section, nous allons détailler trois approches de la compression de données : l'approche par un dictionnaire global nommée prototype 1, l'approche par plusieurs dictionnaires nommée prototype 2 et l'approche par dictionnaire hiérarchique nommée prototype 3.

2.2.1 Approche de la compression par un dictionnaire

Cette première approche de la compression par un dictionnaire (prototype 1) est inspirée de SGBDR IBM DB2 (IBM 1994). Elle permet de compresser les données d'une table entière en utilisant un seul dictionnaire. La technique de compression s'effectue par l'élimination toutes les données répétitives trouvées dans les lignes de la table. Ces données répétitives de types chaînes de caractères, de types date, ou d'autres types sont remplacées par des nombres entiers et leurs valeurs sont stockées une seule fois dans le dictionnaire global. Ce dernier permet de retrouver l'information originale durant la décompression de la table. Cette phase de décompression est nécessaire lorsqu'il y a des requêtes interrogeant la table de l'entrepôt de données. La Figure 2.1 montre un exemple de la compression d'une table voiture en utilisant un dictionnaire global.

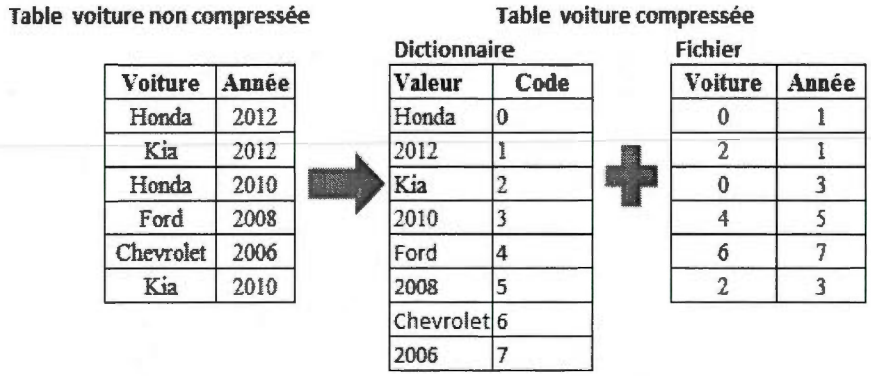


Figure 2.1 Approche de la compression par dictionnaire global.

2.2.2 Approche de la compression par plusieurs dictionnaires

Cette deuxième approche de la compression par plusieurs dictionnaires (prototype 2) est inspirée de l'approche Oracle (Poess et Potapov 2003). Elle reprend le même principe de la 1ère approche précédente, sauf, qu'elle compresse les données de la table en plusieurs blocs dont chaque bloc est composé d'un dictionnaire de données associé à un fichier de données. L'ensemble des dictionnaires de données et les fichiers de données constituent la table entière.

Le résultat de cette approche de compression par bloc que nous avons mise en œuvre dans le chapitre III, fournit deux fichiers compressés dépendants. Le premier fichier contient plusieurs dictionnaires de données structurés en blocs. Chaque bloc de données contient une partie des données de la table. Le deuxième fichier contient des codes entiers correspondants aux données stockées dans les blocs de dictionnaire. Ces codes entiers sont enregistrés en format binaire d'une taille de 8 bits (type byte), de 16 bits (type short) ou de 32 bits (type integer). Ces différents codes sont créés en fonction du volume et les types d'attributs de la table de données. La Figure 2.2 montre un exemple de compression d'une table voiture en plusieurs dictionnaires structurés en deux blocs.

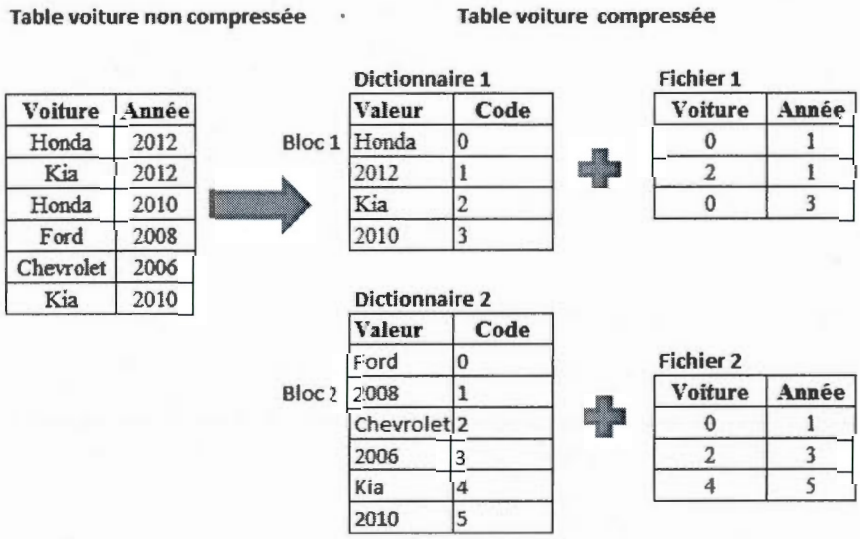


Figure 2.2 Approche de la compression par plusieurs dictionnaires.

2.2.3 Approche de la compression par un dictionnaire hiérarchique

Cette troisième approche de compression par un dictionnaire hiérarchique (prototype 3) représente une nouvelle approche que nous avons proposée dans ce projet de recherche. Cette approche reprend le même principe de la deuxième approche, en ajoutant un niveau supplémentaire de bloc nommé super dictionnaire. Celle-ci implique la création d'un arbre hiérarchique sous forme père-fils entre le super dictionnaire et les dictionnaires de données. Le super

dictionnaire sert de stocker toutes les données communes entre les blocs de dictionnaire. La Figure 2.3 montre un exemple de la compression par un dictionnaire hiérarchique sur une table voiture. La compression de cette table a créé un super dictionnaire (super dict) relié à deux dictionnaires (dict1, dict2) qui sont associés à deux fichiers de données.

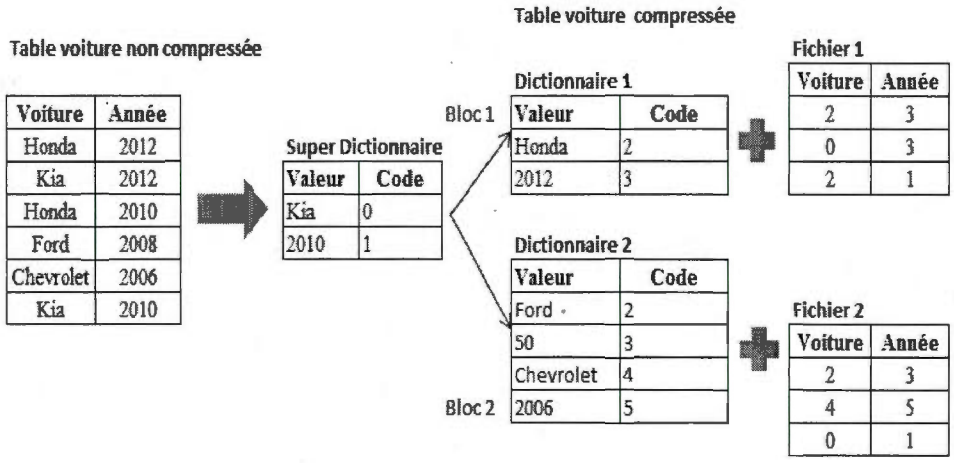


Figure 2.3 Approche de la compression par un dictionnaire hiérarchique.

Dans la démarche de la mise en œuvre du prototype 3, il existe plusieurs variantes de compression de données appliquées sur les blocs de dictionnaire pour créer un bloc de super dictionnaire. On peut citer quelques variantes de compression suivantes :

La première variante définit par : Quelle est la taille ou la longueur des données répétitives trouvées dans les blocs de dictionnaire?

La deuxième variante définit par: Quel est le nombre d'occurrences des données répétitives trouvées dans les blocs de dictionnaire?

La troisième variante définit par : Quel est le nombre des blocs de dictionnaire reliant le super dictionnaire?

La quatrième variante définit par : Quelle est la façon de parcourir et de regrouper les blocs de dictionnaire?

Ces variantes de compression sont définies lors de développement du prototype de compression. La section 2.3 suivante détaille les étapes de développement et de la mise en œuvre de ces approches de compression en particulier le prototype 2 et le prototype 3.

2.3 DÉMARCHE DE DÉVELOPPEMENT D'UN PROTOTYPE DE COMPRESSION

Cette section montre les étapes de notre méthodologie de recherche dans la mise en œuvre du prototype de compression de l'entrepôt de données. Le programme d'exécution des deux prototypes de compression (prototype 2, prototype 3) est représenté par un moteur de base de données nommée EngineDB. Ce dernier permet de calculer et d'afficher le taux de compression de données et la vitesse d'exécution des requêtes des deux prototypes de compression. Les étapes de la méthodologie de développement du prototype de compression commencent à partir de la conception du prototype de compression jusqu'à l'analyse et la comparaison des résultats, comme l'illustre la Figure 2.4. Ces étapes seront détaillées dans les sections suivantes.

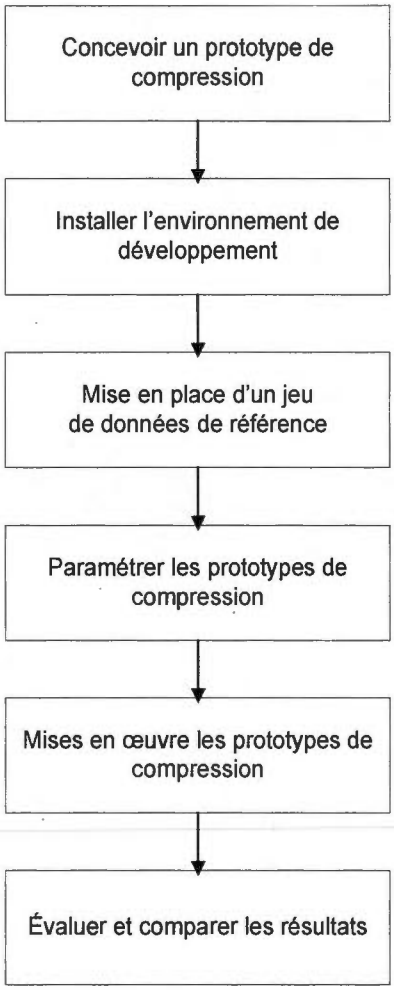


Figure 2.4 Démarche de développement d'un prototype de compression.

2.3.1 Concevoir un prototype de compression

Cette étape est un élément principal de nos objectifs de recherche. Elle définit la démarche de la conception et de développement de notre prototype de compression prototype 3 vu dans la section 2.2.3. Ce prototype a été conçu à partir des études sur les méthodes de compression des systèmes de base de données citées dans la revue de littérature.

2.3.2 Installer l'environnement de développement

Cette étape représente la préparation de la plate forme matérielle et logicielle pour le développement et l'exécution du prototype de compression. La partie logicielle constitue d'un système d'exploitation Windows 7 contenant un environnement de développement (EDI) java NetBeans version 7. L'installation de java est effectuée en deux phases. La première phase, installé le kit de développement java (JDK) requise pour compiler les programmes de java. La deuxième phase, installer le logiciel NetBeans requis pour développer notre moteur de base de données EngineDB.java (Anne 2009). La partie matérielle constitue d'une station de travail équipée d'un processeur double cœurs à une vitesse d'exécution 2.10 GHz et une mémoire centrale RAM de 4 Go.

2.3.3 Mise en place d'un jeu de données de référence

Les prototypes de compression ont été testés sur un entrepôt de données nommé SSB (Star Schéma Benchmark) composé d'une table de faits et des tables de dimensions. Ce schéma SSB est une dérivée du TPC-H Benchmark. Une définition et une comparaison de ces deux bases de données TPC-H Benchmark et SSB seront détaillées dans la section suivante.

2.3.3.1 TPC-H Benchmark

TPC (Transaction Processing Performance Council) est un organisme à but non lucratif qui regroupe plusieurs membres informatique (IBM, Oracle,...). Aujourd'hui, TPC représente une référence universelle dans le but d'élaborer des tests de performance sur des systèmes d'information transactionnels OLTP sous l'appellation TPC-C ou des systèmes d'information décisionnels OLAP sous l'appellation TPC-H. Cette dernière référence TPC-H évalue les performances de différents systèmes d'aides à la décision en exécutant un ensemble des requêtes sur un grand volume de données afin de répondre aux questions critiques du métier d'affaires (TPC-H 2011). Les éditeurs des logiciels et des matériels informatiques comme IBM ou Oracle publient régulièrement les records sur

l'évolution des tests des performances. La Figure 2.5 montre un schéma de données TPC-H Benchmark composé de deux tables de faits LINEITEM, ORDERS et de six tables de dimensions PART, PARTSUPP, SUPPLIER, CUSTOMER, NATION, REGION. Les flèches du schéma font l'association entre les tables sources et les tables destinations via des clés de référence.

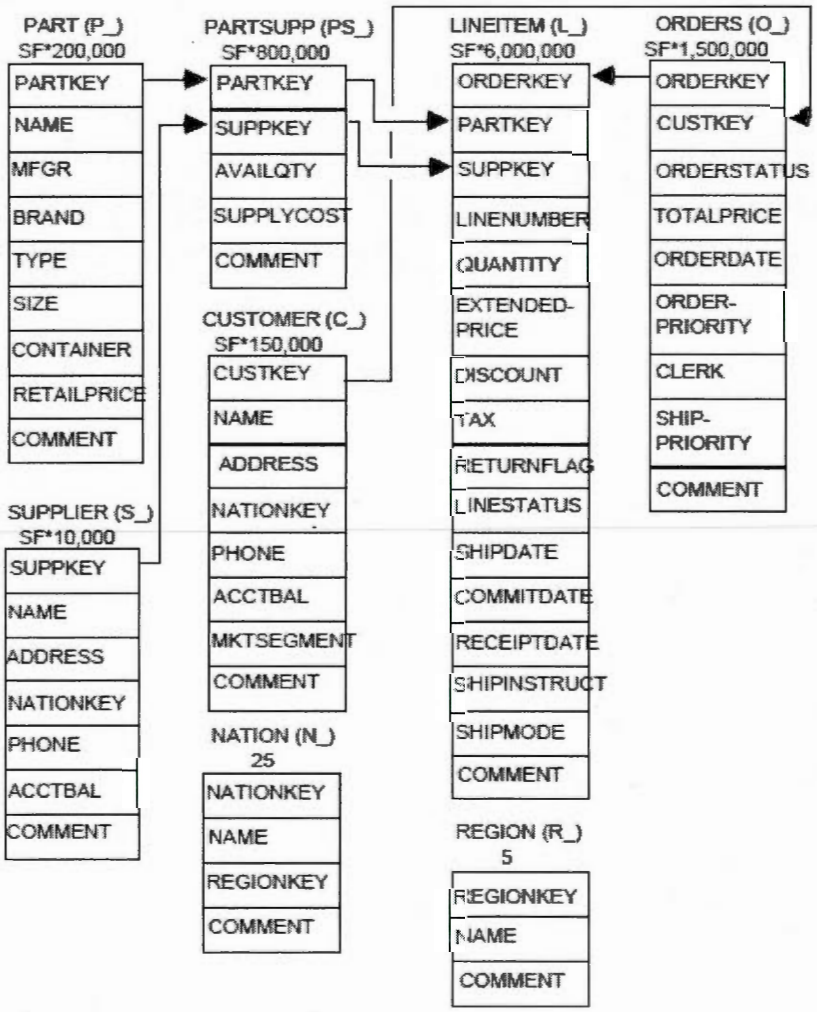


Figure 2.5 Schéma TPC-H (O'Neil, O'Neil et al. 2009, p. 1).

2.3.3.2 SSB

Le SSB est un schéma simplifié du schéma TPC-H benchmark (O'Neil, O'Neil et al. 2009). Plusieurs modifications ont été apportées sur le schéma TPC-H afin d'obtenir le schéma SSB comme l'illustre la Figure 2.6. Ces modifications touchent principalement la suppression des attributs inutiles comme l'attribut COMMENT puisque ces attributs occupent plus d'espace de stockage et ne sont pas agrégés dans les requêtes d'agrégats, la fusion les deux tables de faits LINEITEM et ORDER en une seule table de faits LINEORDER pour avoir un schéma en étoile, etc.

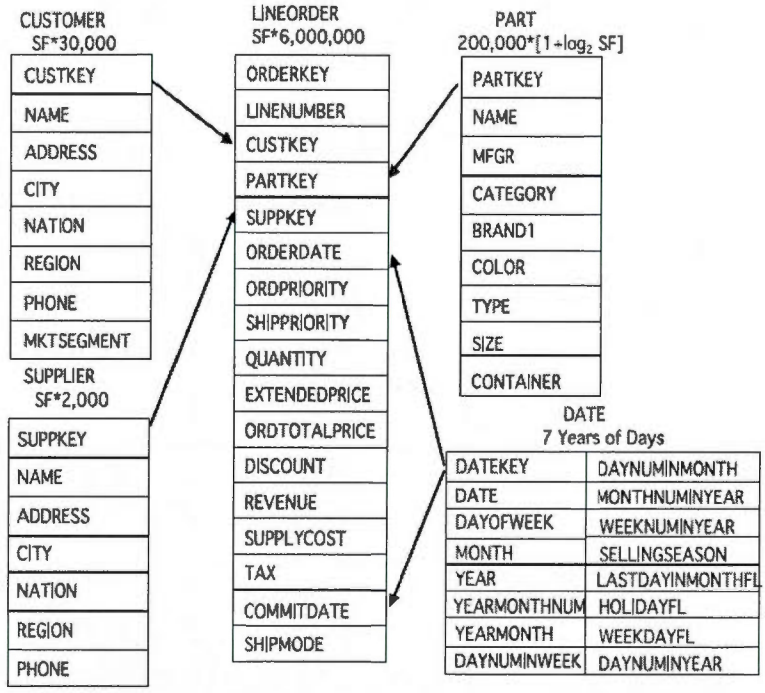


Figure 2.6 Schéma SSB (O'Neil, O'Neil et al. 2009, p. 1).

L'installation de SSB génère deux outils DBGEN et QGEN représentant deux programmes exécutables écrits en langage ANSI-C. Ces deux outils permettent respectivement de peupler une base de données et de générer des requêtes afin de tester les performances d'un système ou une application métier.

L'exécution du premier DBGEN (Database Generator) fournit un entrepôt de données sous forme en étoile (star schéma) représenté par la Figure 2.6, dont la table centrale nommée la table de faits LINEORDER reliée par des axes d'analyses nommés les tables de dimensions (CUSTOMER, SUPPLIER, DATE, PART). La création de différents volumes d'une table de l'entrepôt de données dépend du paramètre d'entrée SF (Scale Factor) de la commande DBGEN. Ce dernier paramètre est un nombre entier qui définit le facteur d'évolution d'une table.

Par exemple, la commande suivante *(dbgen)* génère une table de faits LINEORDEDR par le paramètre (-T L) avec une taille de 1 Go par le paramètre (-s 1) :

root@centos dbgen]# ./dbgen -v -T L -s 1

L'exécution du deuxième programme QGEN (Query Generator) fournit un ensemble des requêtes de référence caractérisées par deux aspects essentiels : l'aspect fonctionnel et l'aspect sélectif. Les requêtes de l'aspect fonctionnel permettent de couvrir toutes les fonctionnalités du système en exécutant un ensemble des requêtes dans le schéma SSB. Les résultats de ces requêtes permettent aux utilisateurs d'évaluer les performances du système fonctionnel à mettre en œuvre. Les requêtes de l'aspect sélectif (Filter Factor FF) permettent de sélectionner et de récupérer un volume de données restreint selon le degré de sélectivité appliqué dans les clauses des requêtes (O'Neil 1991).

Par conséquent, la mise en œuvre des prototypes de compression 2 et 3 est basée par deux éléments : le 1^{er} élément est défini par une table de faits LINEORDER interrogée par des requêtes de sélection. En effet, cette table est très volumineuse et très sollicitée dans la consultation de l'entrepôt de données; le 2^{eme} élément est défini par des requêtes de sélection. Dans ce cas, nous utilisons deux types de requêtes, des requêtes à balayage complet de la table (full scan) qui permettent de parcourir toutes les lignes de la table et des requêtes sélectives qui permettent de récupérer un ensemble restreint des données de la table.

2.3.4 Paramétrer les prototypes de compression

Les prototypes de compression 2 et 3 ne sont mis en œuvre qu'après l'initialisation des paramètres de compression. Par exemple, le paramètre de bloc de données est initialisé à 8 ko, car cette valeur est plus recommandée dans le stockage des données de l'entrepôt. De plus, le prototype de compression 3 a été défini, premièrement, chaque super dictionnaire est relié à deux dictionnaires de données; deuxièmement, deux dictionnaires de données sont associés s'il y a une valeur commune entre eux quelques soit la taille de la valeur trouvée; troisièmement, la recherche des dictionnaires de données contenant des valeurs communes est effectuée de manière séquentielle selon leurs stockages dans le fichier de données.

2.3.5 Mises œuvre des prototypes de compression

Cette étape consiste à développer en langage JAVA un moteur de base de données nommée EngineDB.java. Ce moteur permet d'exécuter l'algorithme de compression et de décompression du prototype 2 et 3, d'exécuter les requêtes d'entrées, de calculer le taux de compression et la vitesse d'exécution des requêtes. La structure de données et les fonctions du projet EngineDB.java sont présentées dans l'appendice A.

La Figure 2.7 illustre les étapes du processus de compression du prototype 2 et 3.

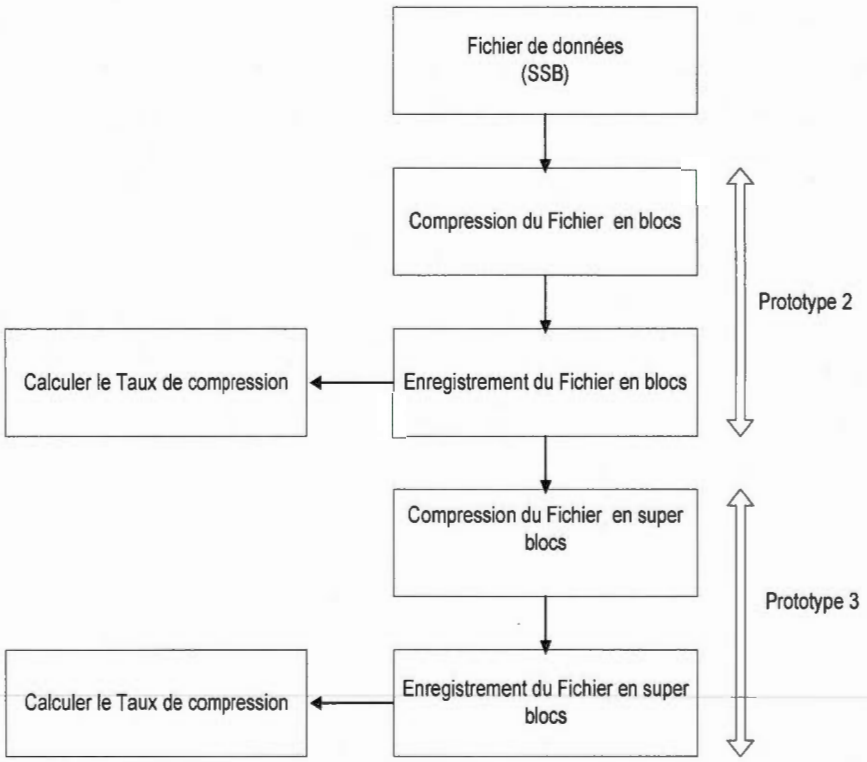


Figure 2.7 Phase de la compression des données.

La Figure 2.8 illustre l'exécution et le calcul du temps de réponse des requêtes.

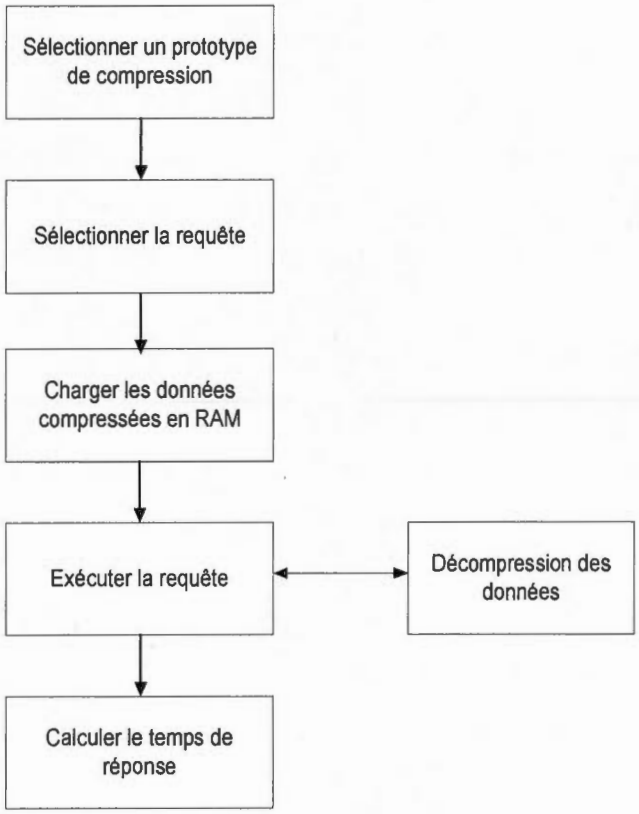


Figure 2.8 Phase d'exécution des requêtes.

2.3.6 Évaluer et comparer les résultats

Les performances du taux de compression et le temps de réponse des requêtes sont évaluées sur différents volumes de données de la table de faits LINEORDER. Ces volumes de données sont aussi triés pour calculer encore une fois les performances de l'entrepôt de données. Les résultats obtenus sont comparés et analysés entre les deux prototypes 2 et 3. Le chapitre III présente les résultats d'exécution du prototype 2 et 3.

2.4 CONCLUSION

Dans ce chapitre, nous avons proposé les trois approches de compression des tables de l'entrepôt de données. Toutes ces approches utilisent une technique de compression de dictionnaire de données sans perte (lossless). Cependant, la première approche (prototype 1) compresse les données de façon plus rentable par rapport aux techniques de compression par bloc (prototype 2 et 3). Aussi, nous avons abordé la méthodologie de recherche de la compression de l'entrepôt de données pour améliorer les performances en mettant l'accent sur toutes les étapes de développement de notre moteur de base de données EngineDB. Ce moteur exécute les programmes des prototypes de compression 2 et 3 appliqués sur un entrepôt de données SSB. Les résultats des tests obtenus par le moteur EngineDB sont basés par deux paramètres de performance : le taux de compression et la vitesse d'exécution des requêtes. Ces résultats seront détaillés dans le chapitre III.

CHAPITRE III

PRÉSENTATION DES RÉSULTATS ET ANALYSE

3.1 INTRODUCTION

Dans les chapitres précédents, nous avons défini les enjeux de l'entrepôt de données en deux axes principaux, le volume des données important et le délai d'exécution des requêtes complexe. Ensuite, nous avons exploré la revue de la littérature sur les méthodes et les techniques de compression de données en particulier la technique d'utilisation de dictionnaire. Les études des algorithmes de compression des systèmes de bases de données ont permis de proposer une méthodologie de recherche d'un prototype de compression qui débute de la définition et la conception jusqu'à la mise en œuvre. Dans ce dernier chapitre, nous présenterons, et nous comparerons les résultats d'exécution des deux prototypes 2 et 3. Ces derniers sont exécutés en deux étapes successives, la compression de données et l'exécution des requêtes. Dans la première étape, la compression de données a été appliquée sur une table faits volumineuse d'un entrepôt de données. Dans la deuxième étape, un ensemble de requêtes ont été exécutées dans notre moteur de base de données EngineDB. Ensuite, nous présenterons une synthèse des résultats d'exécution de ces deux étapes précédentes en décrivant les points forts et les points faibles de chacun de ces deux prototypes de compression. Enfin, nous proposerons les perspectives de recherche, en mettant en évidence, notre contribution dans ce projet, les limites du

projet, et les différentes pistes de recherche future du projet, ensuite nous terminerons ce mémoire par une conclusion.

3.2 DÉTERMINATION DE L'ENTREPÔT DE DONNÉES

La compression de l'entrepôt de données a été appliquée sur un schéma SSB. Après l'évaluation des données du schéma SSB, nous constatons que la table de faits LINEORDER est plus avantageuse dans les tests des méthodes de compression et les performances d'exécution des requêtes. En effet, cette table de faits inclut tous les clés de référence des dimensions avec différents types de données comme le montre le Tableau 3.1. Cela nous permet d'exécuter différents types de requêtes dans l'entrepôt de données.

Tableau 3.1 Structure de LINEORDER (O'Neil, O'Neil et al. 2009, p. 3)

Colonnes	Types de données	Définition
Lo_orderkey	Numérique	Lo_orderkey et lo_linenumber sont des clés composées dont le premier est la clé primaire de la table orders et le deuxième est la clé primaire de la table lineitem.
Lo_linenumber	Numérique	
Lo_custkey	Numérique	Clé étrangère de la table customer.
Lo_partkey	Numérique	Clé primaire de la table part.
Lo_suppkey	Numérique	Clé primaire de la table supplier.
Lo_orderdate	Date	Date de la commande orders.
Lo_orderpriority	Texte	Définit la priorité de la commande, contient 5 valeurs distinctes (urgent, high, medium, low, not speci)
Lo_shippriority	Texte	Définit la priorité de l'expédition. Ce champ n'est pas renseigné et est contient une valeur 0 partout.
Lo_quantity	Numérique	La quantité du produit commandée
Lo_extendedprice	Numérique	Le prix du produit
Lo_ordtotalprice	Numérique	Le montant total de la commande
Lo_discount	Numérique	Le pourcentage de 0 à 10 de remise pour le produit.
Lo_revenue	Numérique	Le revenue du produit calculé par la formule : (lo_extendedprice*(100-lo_discount))/100)
Lo_supplycost	Numérique	Le coût du produit
Lo_tax	Numérique	Les taxes de 0 à 8 sur le produit
Lo_commitdate	Date	Date de la validation de la commande
Lo_shipmode	Texte	Définit le mode d'expédition, contient 7 valeurs (air, rail, mail, truck,….)

De plus, cette table de faits est la plus volumineuse par rapport aux tables de dimensions, car elle résulte de la combinaison de deux tables importantes LINEITEM et ORDER du schéma TPC-H Benchmark. Le Tableau 3.2 montre les caractéristiques des tables TPC-H créent par un facteur d'évolution SF=1.

Tableau 3.2 Estimation de la taille de la base de données (TPC-H 2011, p. 87)

Tables	Cardinalité	Longueur d'une ligne (Octet)	Taille (Mo)
Supplier	10000	159	2
Part	200000	155	30
Partsupp	800000	144	110
Cutomer	150000	179	26
Orders	1500000	104	149
Lineitem	6001215	112	641
Nation	25	128	< 1
Region	5	124	< 1
Total	8661245		956

3.3 RÉSULTATS DE LA COMPRESSION DES DONNÉES

Le Tableau 3.3 montre les différents volumes de la table de faits LINEORDER (LO). Celle-ci nous permet d'évaluer et de tester les performances de la compression de données et l'exécution des requêtes durant l'étape de la mise en œuvre de notre moteur de base de données EngineDB.

Tableau 3.3 Volumes de LINEORDER (LO)

Tables	LO2	LO3	LO4	LO5	LO6	LO7	LO8
Volumes	500	1000	10000	50000	100000	500000	600000

Pour chaque volume de données généré (LO), nous faisons un tri de la table. Ce tri peut être un tri ayant une cardinalité faible ou un tri ayant une cardinalité forte. Dans ce cas, nous obtenons trois modes de stockage de la table : un mode normal, un mode en tri faible (cardinalité faible) et un mode en tri fort (cardinalité forte).

- Un mode normal est défini par la génération initiale des données la table.
- Un mode en tri faible est défini par le tri des données de la table en commençant par les champs de cardinalités faibles jusqu'aux les champs de cardinalités fortes. Par exemple, la cardinalité d'un champ sexe = 2 puisque ce champ contient deux valeurs distinctes masculin et féminin, ce champ est considéré comme un champ de cardinalité faible. Le Tableau 3.4 montre les cardinalités des champs pour chaque version d'une table LINEORDER (LO). Cette cardinalité peut être différente d'un volume de données de la table à un autre. Cela tout dépend le nombre des variations des valeurs du champ.
- Un mode en tri fort est défini par le mode inverse du mode de tri faible.

Tableau 3.4 Cardinalités des champs LINEORDER (LO)

Colonnes / tables	LO2	LO3	LO4	LO5	LO6	LO7	LO8
Lo_shippriority	1	1	1	1	1	1	1
Lo_orderpriority	5	5	5	5	5	5	5
Lo_shipmode	7	7	7	7	7	7	7
Lo_linenumber	7	7	7	7	7	7	7
Lo_tax	9	9	9	9	9	9	9
Lo_discount	11	11	11	11	11	11	11
Lo_quantity	51	51	51	51	51	51	51
Lo_orderdate	118	237	1597	2395	2406	2406	2406
Lo_suppkey	429	796	1987	2000	2000	2000	2000
Lo_ordtotalprice	100	210	1989	9917	19911	99915	119747

Lo_custkey	123	253	2347	9055	13675	19836	19926
Lo_commitdate	439	800	2398	2460	2461	2466	2466
Lo_orderkey	125	256	2509	12459	24914	124929	149891
Lo_supplycost	494	959	6724	12160	12725	12928	12934
Lo_extendedprice	389	807	7835	38772	76091	320062	368485
Lo_revenue	389	807	7871	39512	79059	381483	453202
Lo_partkey	499	997	42968	44157	78625	183445	189892

Les résultats de la compression de données des deux prototypes 2 et 3 sont montrés par les Tableaux 3.5, 3.6 et 3.7. Ces résultats ont été testés sur les différents volumes de la table de faits LINEORDER (LO) en appliquant les trois modes de tri. Chaque valeur du tableau est mesurée en kilooctet (ko) qui indique la taille du fichier compressé. La structure du prototype 3 est définie par un super bloc regroupant deux blocs ayant une ou plusieurs valeurs communes. Les blocs de dictionnaire ont été choisis selon leurs stockages séquentiels dans le disque.

Tableau 3.5 Compression LINEORDER en mode normal

Volumes	Prototype 2	Prototype 3
500	30	30
1000	59	59
10000	573	566
50000	2865	2834
100000	5735	5672
500000	28718	28405
600000	34457	34082

Tableau 3.6 Compression LINEORDER en mode de cardinalité faible

Volumes	Prototype 2	Prototype 3
500	30	30
1000	61	60
10000	642	634
50000	3321	3293
100000	6684	6633
500000	33749	33530
600000	40525	40265

Tableau 3.7 Compression LINEORDER en mode de cardinalité forte

Volumes	Prototype 2	Prototype 3
500	32	31
1000	65	64
10000	658	650
50000	3291	3257
100000	6543	6478
500000	32596	32298
600000	39066	38715

3.4 RÉSULTATS DE L'EXÉCUTION DES REQUÊTES

La majorité des accès à un entrepôt de données sont en lecture seule. Cela a été un objectif de la conception des tables compressées (Poess et Potapov 2003). Dans notre projet, nous visons principalement les requêtes en lecture seule de type SELECT interrogeant une table de faits LINEORDER. Il existe plusieurs types de requêtes d'accès aux lignes d'une table. On trouve par exemple les requêtes de

lecture entière de la table (full scan), les requêtes conditionnelles (filters), les requêtes d'accès par ROWID (adresse row), requêtes d'agrégats (group by) et les requêtes de jointures.

Afin d'évaluer la nouvelle technique de compression par dictionnaire hiérarchique, nous avons exécuté et testé les prototypes de compression 2 et 3 sur deux types de requêtes : les requêtes de lecture entière de la table et les requêtes conditionnelles.

Les premiers types des requêtes de lecture entière de la table permettent de parcourir toutes les lignes de la table. Par exemple : select * from lineorder.

Les deuxièmes types des requêtes conditionnelles permettent de sélectionner et de récupérer une partie des lignes de la table en appliquant des contraintes sur les attributs de la table. Par exemple : select * from lineorder where LO_orderpriority='URGENT'.

Dans chaque exécution d'une requête, nous calculons le temps de réponse de la requête. Ce temps est calculé par l'écart entre l'application de la fonction java System.currentTimeMillis() dans le début et dans la fin de la requête.

Les Tableaux 3.8, 3.9 et 3.10 montrent les résultats d'exécution mesurés en milli seconde (ms) de la requête Q1 : select * from lineorder.

Tableau 3.8 Temps de réponse Q1 en mode normal

Volumes	Prototype 2	Prototype 3
500	2	2
1000	3	4
10000	37	44
50000	194	219
100000	383	439
500000	1926	2203
600000	2286	2672

Tableau 3.9 Temps de réponse Q1 en mode de cardinalité faible

Volumes	Prototype 2	Prototype 3
500	2	2
1000	3	4
10000	37	44
50000	189	220
100000	381	440
500000	1936	2237
600000	2339	2691

Tableau 3.10 Temps de réponse Q1 en mode de cardinalité forte

Volumes	Prototype 2	Prototype 3
500	2	2
1000	4	4
10000	37	45
50000	193	223
100000	385	447
500000	1955	2267
600000	2381	2724

Les Tableaux 3.11, 3.12 et 3.13 montrent les résultats d'exécution de la requête Q2: select * from lineorder where LO_orderpriority= 'URGENT'.

Tableau 3.11 Temps de réponse Q2 en mode normal

Volumes	Prototype 2	Prototype 3
500	0	0
1000	1	1
10000	9	11
50000	51	55
100000	100	108
500000	500	545
600000	590	679

Tableau 3.12 Temps de réponse Q2 en mode de cardinalité faible

Volumes	Prototype 2	Prototype 3
500	0	0
1000	0	1
10000	8	10
50000	40	47
100000	83	95
500000	442	508
600000	536	615

Tableau 3.13 Temps de réponse Q2 en mode de cardinalité forte

Volumes	Prototype 2	Prototype 3
500	0	0
1000	1	1
10000	10	12
50000	52	58
100000	101	115
500000	536	602
600000	650	719

3.5 SYNTHÈSE DES RÉSULTATS

Dans la partie des résultats de la compression de différents volumes de données (100000, 500000, 600000) de la table de faits LINEORDER, nous avons constaté que la compression de données du prototype 3 est meilleure que la compression de données du prototype 2. Par ailleurs, les résultats de la compression de données en mode normal donnent une meilleure compression des données par rapport au mode de cardinalité faible et le mode de cardinalité forte. Les Figures 3.1, 3.2 et 3.3 résument les résultats de la compression entre le prototype 2 et le prototype 3. Ces résultats de compression ont été exécutés sur trois modes de compression (normal, tri faible et tri fort).

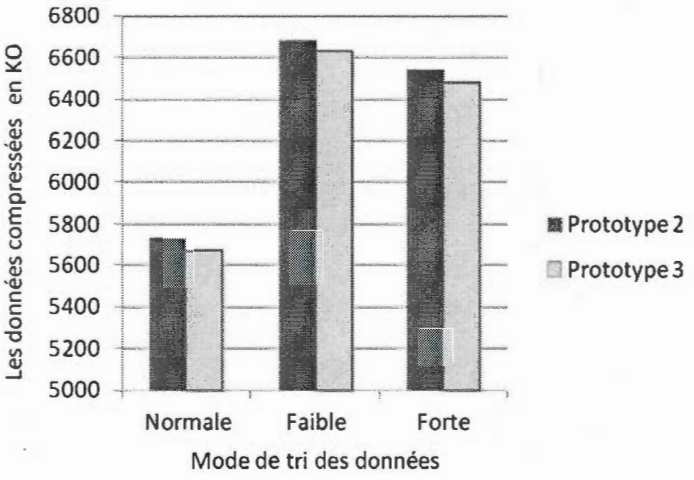


Figure 3.1 Compression d'une table contenant 100000 lignes.

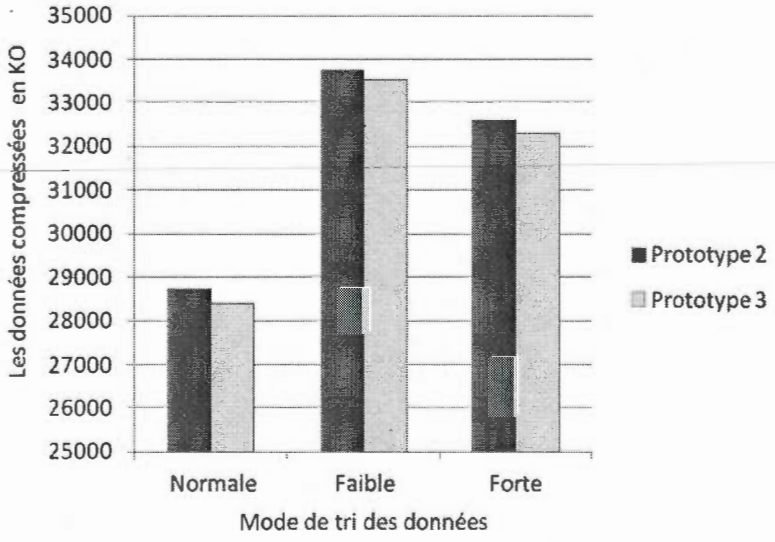


Figure 3.2 Compression d'une table contenant 500000 lignes.

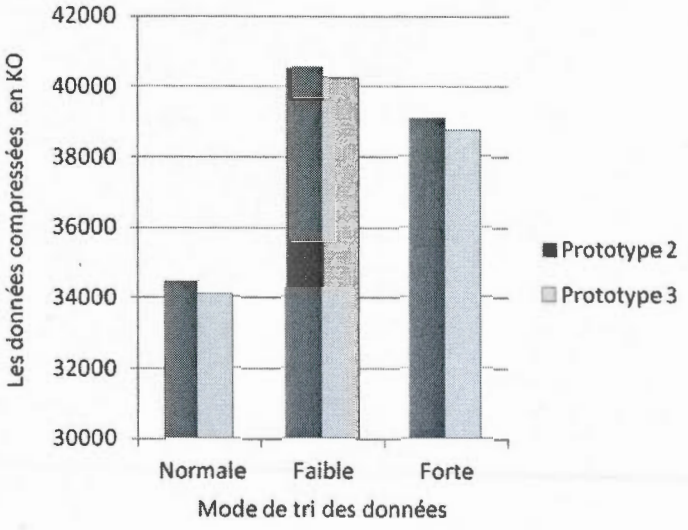


Figure 3.3 Compression d'une table contenant 600000 lignes.

Dans la partie des résultats d'exécution des requêtes appliquées aux trois modes de compression (normal, cardinalité faible, cardinalité forte), nous avons constaté que le temps de réponse des requêtes Q1 et Q2 du prototype 3 sont élevés par rapport au temps de réponse des requêtes Q1 et Q2 du prototype 2.

L'analyse de ces résultats peut être justifiée par plusieurs facteurs. Le 1er facteur : le prototype 3 utilise plus de tests conditionnels (si et sinon) par rapport le prototype 2 durant la décompression des données. En effet, ces tests conditionnels sont utilisés par le prototype 3 pour parcourir l'arbre hiérarchique afin de retrouver les données originales stockées dans le dictionnaire et le super dictionnaire. Ceci provoque un retard dans le temps de réponse des résultats des requêtes. De plus, Poess et Potapov (2003) ont montré que l'accès aux champs compressés dans certains types de requêtes complexes augmente le coût d'exécution du CPU, puisque pour chaque champ compressé, un pointeur est ajouté entre le dictionnaire et le fichier de données afin de décompresser et de retrouver l'information originale. Le 2eme facteur : les données utilisées dans les tests des prototypes de compression sont des données de type SSB générées

automatiquement par le programme DBGEN. Par conséquent, ces données ne reflètent pas réellement un entrepôt de données d'une entreprise. De plus, Poess et Potapov (2003) ont analysé plusieurs entrepôts de données de différents clients, et ont montré que les données d'un entrepôt sont généralement groupées sur certains attributs des colonnes comme l'attribut date.

Les Figures 3.4, 3.5 et 3.6 résument les résultats des tests de réponse de la requête Q1 entre le prototype 2 et le prototype 3 appliquées sur différents volumes des données (100000, 500000 et 600000) de la table de faits LINEORDER.

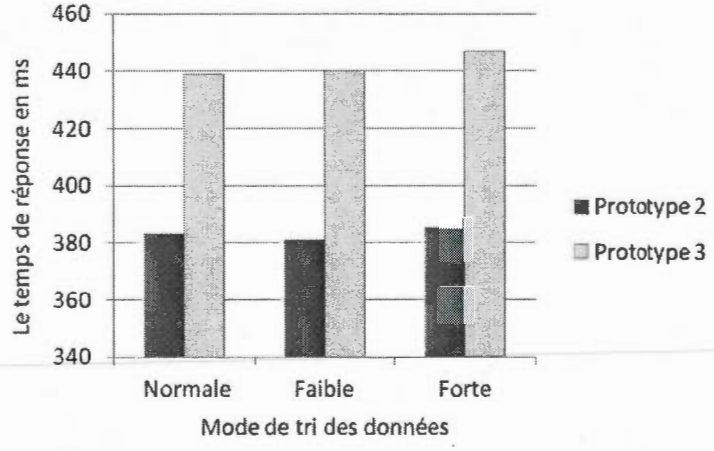


Figure 3.4 Réponse Q1 d'une table contenant 100000 lignes.

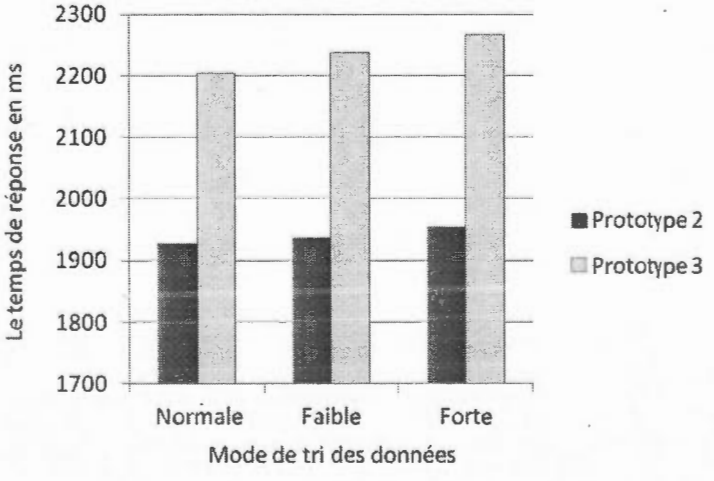


Figure 3.5 Réponse Q1 d'une table contenant 500000 lignes.

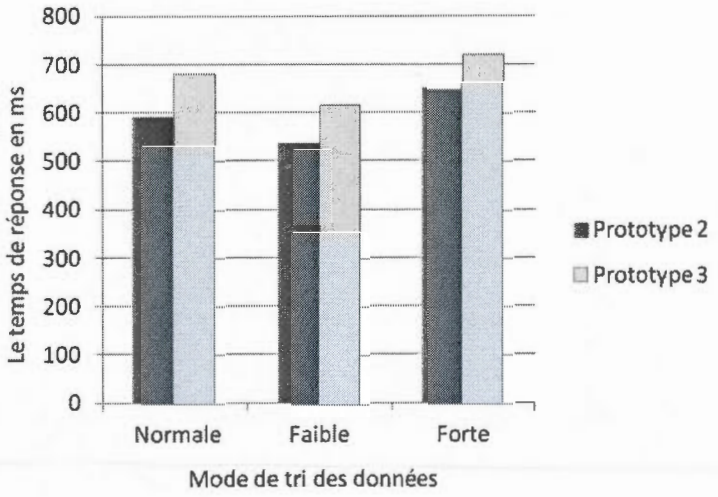


Figure 3.6 Réponse Q1 d'une table contenant 600000 lignes.

3.6 PERSPECTIVES DE RECHERCHE

3.6.1 Contribution de recherche

Plusieurs algorithmes de compression des tables de la base de données relationnelles ont été développés. Cependant, ces algorithmes traitent la compression de données en utilisant un seul dictionnaire global ou plusieurs dictionnaires structurés en blocs.

Notre contribution dans ce projet vise, premièrement, de tirer profit de ces algorithmes de compression; deuxièmement, de proposer un nouveau prototype de compression sous forme d'une structure hiérarchique composée des blocs de dictionnaire de données. Ce nouveau prototype représente une extension du prototype d'Oracle, car il permet d'ajouter un niveau supplémentaire de bloc de dictionnaire de données nommée le super dictionnaire. Chaque bloc de super dictionnaire est relié à deux blocs de dictionnaire. Par conséquent, cette structure hiérarchique permet d'éliminer plusieurs valeurs communes entre chaque groupe de deux blocs de dictionnaire. De plus, ce nouveau prototype permet de faire la

compression et la décompression des tables d'une base de données relationnelle ou des tables de l'entrepôt de données.

3.6.2 Limites de recherche

Notre projet de recherche a étudié une nouvelle technique de compression basée sur un dictionnaire hiérarchique. Ce dernier a été conçu et développé dans un prototype de compression nommé prototype 3. Cependant, le projet de recherche a été limité et testé sur une plate forme bien spécifique. Cette plate forme est constituée d'un système d'exploitation Windows 7 en mode 64 bits et une station de travail dotée d'un processeur à double cœurs, d'une mémoire RAM de 4 Go et d'un disque dur de 500 Go.

De plus, les tests de compression de données et d'exécution des requêtes ont été effectués sur un jeu de données spécifique de type SSB et un ensemble des requêtes de type SELECT incluant les requêtes de lecture entière de la table et les requêtes conditionnelles. Ces requêtes interrogent différents volumes de données d'un entrepôt représenté par la table de faits LINEORDER.

Enfin, nous avons tenu en compte deux paramètres principaux dans le calcul des performances de l'entrepôt de données : le paramètre du taux de compression et le paramètre d'exécution des requêtes.

3.6.3 Avenues de recherche future

Notre algorithme de compression a été développé dans une approche de stockage orientée ligne. Dans les travaux futurs, il serait intéressant de reprendre notre sujet de recherche dans une nouvelle approche qui est l'approche de stockage orientée colonne. Dans cette dernière approche, plusieurs travaux de recherche ont été menés (Stonebraker, Abadi et al. 2005), et plusieurs bases de données orientées colonne ont été développées et commercialisées comme Monet DB, Vertica, Infobright.

3.7 CONCLUSION

Dans ce chapitre, nous avons mis en œuvre deux prototypes (prototype 2 et prototype 3) dans un environnement de développement java sous un entrepôt de données SSB. Le programme de fonctionnement de ces deux prototypes a été nommé le moteur de base de données EngineDB. Dans ce moteur, nous avons calculé dans premier temps, le taux de compression de chaque prototype. La compression de données a été effectuée sur différents volumes de données du schéma SSB. Dans un deuxième temps, nous avons calculé la vitesse d'exécution des requêtes. Dans les résultats de comparaison entre les deux prototypes, une première évaluation basée sur le taux de compression indique que la compression de données du prototype 3 est meilleure que la compression de données du prototype 2 puisque le prototype 3 utilise un super dictionnaire pour stocker toutes les valeurs répétitives trouvées dans les deux blocs de dictionnaire. Une deuxième évaluation basée sur la vitesse d'exécution des requêtes indique que le temps de réponse des requêtes du prototype 2 est plus rapide que le temps de réponse des requêtes du prototype 3 puisque la décompression du prototype 3 prend un certain temps dans le parcours de l'arbre hiérarchique afin de retrouver l'information originale entre le super dictionnaire, le dictionnaire et le fichier de données. À partir de ces évaluations précédentes, nous pouvons conclure que l'amélioration des performances de l'entrepôt de données est un compromis (trade-off) entre le taux de compression de données et la vitesse d'exécution des requêtes. C'est à dire une compression de données plus élevée peut influencer sur le fonctionnement de CPU et par conséquent, cela peut causer un ralentissement de la vitesse d'exécution des requêtes.

CONCLUSION

Dans ce document, nous avons défini notre contexte de recherche dans la compression de l'entrepôt de données pour améliorer les performances, c'est-à-dire proposer des améliorations aux algorithmes de la compression afin d'augmenter la performance de l'entrepôt de données. Celle-ci peut être définie par la question suivante : comment compresser l'entrepôt de données pour améliorer les performances?

À partir de ce contexte de recherche, nous avons entamé l'étude de la revue de la littérature sur les méthodes et les techniques de compression de données. Celle-ci nous a permis de préciser les questions de recherche, ainsi que les objectifs de recherche à réaliser.

Dans notre mémoire, nous avons étudié et décrit trois systèmes de gestion de bases de données : le système Oracle, le système Microsoft (SQL Server), et le système IBM (DB2). Ces systèmes utilisent une technique commune de compression des tables de données. Cette technique est basée sur l'emploi d'un dictionnaire. Cependant, chaque système a un algorithme différent dans la compression des tables. Pour cela, deux approches utiles de la compression ont été détaillées. La première approche de compression est l'emploi d'un dictionnaire global (prototype 1). La deuxième approche de compression est l'emploi de plusieurs dictionnaires (prototype 2). Dans les deux approches précédentes, nous avons identifié des avantages et des inconvénients. Cela nous a amené à proposer une nouvelle approche de compression nommée prototype 3. Ce dernier est défini par une structure d'arbre hiérarchique composée d'un super dictionnaire relie aux dictionnaires de données. Cette nouvelle structuration de stockage de données est conçue dans le but d'améliorer les algorithmes de compression de données.

Plusieurs tests et comparaisons ont été effectués entre le prototype 2 et le prototype 3. Les résultats des tests indiquent que le prototype 3 a obtenu un meilleur taux de compression au détriment de la vitesse d'exécution des requêtes. C'est-à-dire, une compression plus élevée peut avoir une conséquence négative sur la vitesse d'exécution des requêtes. Cela implique que l'amélioration des performances de l'entrepôt de données est un compromis entre la compression et la vitesse des requêtes.

Dans les travaux futurs, nous souhaiterons un redéveloppement de cet algorithme sur d'autres variantes ou techniques de compression comme la compression de données en colonne, le tester et le comparer sur différentes plates formes avec différents entrepôts de données.

APPENDICE A

LA STRUCTURE ET LES FONCTIONS DU MOTEUR ENGINEDB

Tableau A.1 Structure du moteur EngineDB

Nom	**Description**
Arraylist	Tableau dynamique qui permet de stocker le fichier de données en format binaire.
Bidimap	Tableau dynamique à deux valeurs <key, value> de type bidimap de l'api commun collection. Cette nouvelle structure hérite de la classe hashmap, elle permet de stocker les objets de dictionnaire de données.
Objectoutputstream objectinputstream	Permet de lire et écrire des objets de dictionnaire.
Dataoutputstream Datainputstream	Permet de lire et écrire des données de la base sous forme binaire (8 bits, 16 bits, 32 bits).
Csvwriter Csvreader	Permet de lire et écrire des données du fichier csv. Ces classes utilise un package spéciale nommé csv4j
Préfixe buffered	Ce buffer est utilisé dans toute la structure précédente. Elle permet d'allouer un tampon mémoire pour la rapidité du transfert des données en lecture ou en écriture du disque.

File fileofdata File fileofdic	Déclaration d'un fichier de données ou d'un dictionnaire de données
Hashmap<integer,util.dictionary> listofdictionary;	Listofdictionary est un tableau de type hashmap composé de la classe dictionary. Il contient les données (data) et les objets de dictionnaire de données (dic)

Tableau A.2 Fonctions du moteur EngineDB

Nom	Description
Buildfromcsv	Lire et charge dans des tableaux array les données du fichier csv avec l'application en même temps l'algorithme de compression en bloc.
Saveblock	Écrire tous les flux de données et d'objet stockés dans des tableaux array dans les fichiers de sorties selon la taille des données int, short ou byte.
Loadallblocks	L'opération inverse de la méthode saveblock. Elle permet de lire et charge tous les flux de données et d'objet dans des tableaux array.
Restorecsv	Est une méthode supplémentaire qui permet de vérifier la fiabilité du programme java et la restauration (ou la décompression) des données originale.
Buildsupdictionay()	Lire et charge dans la classe superdictionary les données du fichier csv avec l'application en même temps l'algorithme de compression en super bloc.
Savesupdictionary()	Écrire tous les flux de données et d'objet stockés dans la classe superdictionary dans les fichiers de sorties selon la taille des données int, short ou byte.
Loadsupdictionary()	L'opération inverse de la méthode savesupdictionary(). Elle permet de lire et charge tous les flux de données et d'objet dans la classe superdictionary.
Restorecsvfromsupdictionary()	Est une méthode supplémentaire qui permet de vérifier la fiabilité du programme java et la restauration (ou la décompression) des données originale.

APPENDICE B

LES FORMULES DE CALCUL DES PARAMÈTRES DE PERFORMANCE

- La vitesse d'exécution des requêtes est calculée par la formule suivante:
  Vitesse d'exécution de la requête = (TENC – TEC) / TENC
  TENC : Temps Écoulée Non Compressés.
  TEC : Temps Écoulée Compressées

- Le temps d'utilisation du CPU mesuré en seconde dans un intervalle de temps T est calculé par la formule suivante:

$$CPUC = \sum_{i=1}^{T} c_i(t_i - t_{i-1}) \text{ où } c_i \in \{c_1, c_2, \dots, c_T\} \text{ et } t_i \in \{t_0 = 0, t_1, \dots, t_T\}$$

  c_i : représente l'utilisation de chaque CPU mesuré dans un intervalle de temps t_i.

- L'accès aux données en lecture et en écriture (IO :Input /Output) mesuré en Mo dans un intervalle de temps T est calculé par la formule suivante:

$$IOC = \sum_{i=1}^{T} io_i(t_i - t_{i-1}) \text{ où } io_i \in \{io_1, io_2, \dots, io_T\} \text{ et } t_i \in \{t_0 = 0, t_1, \dots, t_T\}$$

  io_i : représente l'utilisation de chaque IO mesuré dans un intervalle de temps t_i.

BIBLIOGRAPHIE

Abadi, D. J., Madden S. R. et Ferreira M. C. 2006. «Integrating compression and execution in column-oriented database systems». In *International Conference on Management of Data* p. 671 - 682. Chicago, USA: ACM SIGMOD.

Abadi, D. J., Madden S. R. et Hachem N. 2008. «ColumnStores vs. RowStores: How Different Are They Really ?». In *International Conference on Management of Data* p. 967-980. Vancouver, Canada: ACM SIGMOD.

Ahuja, R. 2006. «Introducing DB2 9, Part 1: Data compression in DB2 9». IBM. En ligne. <http://www.ibm.com/developerworks/data/library/techarticle/dm-0605ahuja/index.html>. Consulté le 15 Novembre 2011.

Anne, T. 2009. «*Le livre de Java premier langage*», 6. Paris: Eyrolles, 517 p.

Aouiche, K., Darmont J., Boussaïd O. et Bentayeb F. 2005. «Automatic Selection of Bitmap Join Indexes in Data Warehouses». Springer-Verlag. En ligne. <http://www.springerlink.com.proxy.bibliotheques.uqam.ca:2048/content/5hux1hnnta5wr2vl/fulltext.pdf>. Consulté le 13 Décembre 2011.

Bellatreche, L. 2003. «Techniques d'optimisation des requêtes dans les data warehouses». LISI/ENSMA. En ligne. <http://www.lisi.ensma.fr/ftp/pub/documents/papers/2003/2003-SISPS-Bellatreche.pdf>. Consulté le 06 Janvier 2012.

Binnig, C., Hildenbrand S. et Färber F. 2009. «Dictionary-based order-preserving string compression for main memory column stores». In *Proceedings of the 35th SIGMOD International Conference on Management of Data* p. 283-296. New York, USA: ACM.

Chaudhuri, S. et Dayal U. 1997. «An overview of Data Warehousing and OLAP Technologie ». *ACM SIGMOD Record*, vol. 26, no 1, p. 65-74.

Eaton, C. 2006. «Compression comparison to Oracle and Microsoft». Blogs. En ligne. <http://it.toolbox.com/blogs/db2luw/compression-comparison-to-oracle-and-microsoft-8871>. Consulté le 21 Mars 2012.

IBM. 1994. «DB2 V3 Performance Topics». IBM Corporation. En ligne. <http://www.redbooks.ibm.com/redbooks/pdfs/gg244284.pdf?nocache>. Consulté le 17 Mars 2012.

Inmon, W. H. 2002. «*Building the Data Warehouse*», 3: John Wiley, 412 p.

Kimball, R., Ross M., Thornthwaite W., Mundy J. et Becker B. 2008. «*The Data Warehouse Lifecycle Toolkit*», 2: John Wiley 672 p.

Lemke, C., Sattler K.-U., Faerber F. et Zeier A. 2010. «Speeding Up Queries in Column Stores: A Case for Compression». In *Proceedings of the 12th International Conference on Data Warehousing and Knowledge Discovery* p. 117-129. Bilbao, Spain: Springer Berlin / Heidelberg.

Li, J. et Srivastava J. 2002. «Efficient Aggregation Algorithms for Compressed Data warehouses». *Knowledge and Data Engineering, IEEE Transactions on*, vol. 14, no 3, p. 515-529.

Microsoft. 2011. «Data Compression». Library Microsoft SQL Server En ligne. <http://msdn.microsoft.com/en-us/library/cc280449>. Consulté le 12 Février 2012.

Mishra, S. 2009. «Data Compression: Strategy, Capacity Planning and Best Practices». Microsoft. En ligne. <http://technet.microsoft.com/en-us/library/dd894051(SQL.100).aspx>. Consulté le 25 Mars 2012.

Morris, M. 2002. «Teradata Multi-Value Compression V2R5.0». Teradara Whitepaper.

Nicola, M. et Rizvi H. 2003. «Storage Layout and I/O Performance in Data Warehouses». CiteSeerx En ligne. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.3.9256&rep=rep1&type=pdf>. Consulté le 05 Avril 2012.

O'Mahony, C. 2009. «Data Compression in IBM DB2 and Oracle Database». Blog at database-diary.com En ligne. <http://database-diary.com/2009/08/10/data-compression-in-ibm-db2-and-oracle-database/>. Consulté le 10 Avril 2012.

O'Neil, P., O'Neil B. et Chen X. 2009. «Star Schema Benchmark». University of Massachusetts En ligne. <http://www.cs.umb.edu/~poneil/StarSchemaB.pdf>. Consulté le 10 Janvier 2012.

O'Neil, P. E. 1991. «The Set Query Benchmark». University of Massachusetts En ligne. <http://www.cs.umb.edu/~poneil/SetQBM.pdf>. Consulté le 06 Février 2012.

Oracle. 2011. «Choosing Data Block Size». Oracle Database Documentation Library 11g Release 2. En ligne. <http://docs.oracle.com/cd/E11882_01/server.112/e16638/iodesign.htm#PFGRF94404>. Consulté le 05 Avril 2012.

Pascal, P. 1993. «*Compression de donnees (méthodes, algorithmes, programmes détaillés)*». Paris: Eyrolles, 200 p.

Poess, M. et Potapov D. 2003. «Data compression in Oracle». In *Proceedings of the 29th International Conference on Very large Databases* p. 937-947. Berlin, Germany: VLDB Endowment.

Ray, G., Haritsa J. R. et Seshadri S. 1995. «Database Compression: A Performance Enhancement Tool». CiteSeer. En ligne. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.2346>. Consulté le 26 Avril 2012.

Stonebraker, M., Abadi D. J., Batkin A., Chen X., Cherniack M., Ferreira M., Lau E., Lin A., Madden S., O'Neil E., O'Neil P., Rasin A., Tran N. et Zdonik S. 2005. «C-store: a column-oriented DBMS». In *Proceedings of the 31st International Conference on Very large Databases* p. 553-564. Trondheim, Norway: VLDB Endowment.

TPC-H. 2011. «TPC BENCHMARK H». En ligne. <http://www.tpc.org/tpch/default.asp>. Consulté le 20 Février 2011.

Turner, M. J., Hammond R. et Cotton P. 1979. «A DBMS For Large Statistical Databases». In *Fifth International Conference on Very Large Data Bases* p. 319-327.

Vincent, T. 2010. «Resource Optimization Compression Space Reclamation Scan Sharing». IBM. En ligne. <http://www.monash.com/uploads/ibm-db2-compression-june-2010.pdf>. Consulté le 20 Janvier 2011.

Watson, H. J. et Ariyachandra T. 2010. «Key organizational factors in data warehouse architecture selection». *Decision Support Systems*, vol. 49, no 2, p. 200-212.

Wehrle, P. 2009. «Modèle multidimentionnel et OLAP sur architecture de grille ». Informatique, Lyon, École doctorale Informatique et Information pour la Société, Institut National des Sciences Appliquées de Lyon, 214 p.

Westmann, T., Kossmann D., Helmer S. et Moerkotte G. 2000. «The Implementation and Performance of Compressed Databases». *ACM SIGMOD*, vol. 29, no 3.

Winter, R. 2008. «Scaling The Data Warehouse». InformationWeek Software. En ligne. <http://www.informationweek.com/news/software/info_management/210900005 >. Consulté le 14 Mars 2011.