RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITÉ AMAR TÉLIDJI DE LAGHOUAT

FACULTÉ DES SCIENCES

Thèse de Doctorat en Sciences

Spécialité : Informatique

FOUILLE DE DONNÉES POUR LA FORMALISATION ET L'OPTIMISATION DE LA CONCEPTION PHYSIQUE DES ENTREPÔTS DE DONNÉES RELATIONNELS

Présentée et soutenue publiquement, le :

Par : Benameur ZIANI

Composition du jury

M.B. Yagoubi	Professeur	*Président*	Université de Laghouat
A. Moussaoui	Professeur	*Examinateur*	Université de Sétif
K. Boukhalfa	MCA	*Examinateur*	USTHB Alger
H. Cherroun	MCA	*Examinateur*	Université de Laghouat
Y. OUINTEN	MCA	*Directeur de thèse*	Université de Laghouat

REMERCIEMENTS

J'exprime d'abord ma profonde gratitude à Youcef OUINTEN, mon directeur de thèse. Je tiens à saluer son investissement et sa constante disponibilité. Ses remarques constructives m'ont beaucoup aidé dans la réalisation de ce travail.

J'adresse mes sincères remerciements à M.B. YAGOUBI, A. MOUSSAOUI, K. BOUKHALFA, et H. CHERROUN membres du jury, pour l'intérêt qu'ils ont porté à mon travail en acceptant d'être les examinateurs de ma thèse.

J'aimerais aussi chaleureusement remercier François Rioult et Bruno Crémilleux pour mon séjour fructueux passé au CREYC à l'Université de Caen Basse-Normandie.

Un grand merci aussi à N. Lagraa, Directeur du laboratoire LIM, qui a accompagné la rédaction de ma thèse. Merci pour les échanges passionnants et enrichissants dont j'ai pu bénéficier. Merci aussi pour nos nombreux thés et le plaisir partagé à discuter différents sujets de la vie.

Ma reconnaissance va à ceux qui ont plus particulièrement assuré un soutien affectif et sans faille : ma famille et en particulier mes sœurs.

Merci à tous les membres de l'équipe de football de la faculté des sciences, que je ne peux citer nommément avec qui j'ai eu le plaisir de partager des moments inoubliables sur les terrains de football. Merci pour les tournois gagnés dont je me souviendrai avec bonheur.

Mes meilleurs sentiments vont aux doctorants membres du laboratoire LIM (Laboratoire d'Informatique et de Mathématiques) que j'ai côtoyé.

Que ceux et celles que je n'ai pas nommé reçoivent aussi l'expression de toutes mes pensées. En témoignage de reconnaissance, je voudrais remercier tous ceux qui ont contribué à ma formation : mes instituteurs du primaire et du collège, mes professeurs du lycée, et mes enseignants à l'Université. Qu'ils trouvent ici l'expression de ma gratitude et de mon profond respect.

Merci à tous !

RÉSUMÉ

La conception physique des entrepôts de données consiste non seulement à la spécification détaillée des données et de leurs types, mais surtout à la sélection des techniques d'optimisation (index, fragments, etc.) appropriées et susceptibles d'améliorer les performances du système en minimisant les temps nécessaires à l'évaluation des requêtes.

Cependant, le choix d'une solution optimisée est une tâche très difficile qui nécessite non seulement du temps et de l'effort, mais aussi beaucoup d'expertise. Nous pouvons dire que la principale difficulté pour la conception physique réside dans l'énorme espace de recherche des solutions possibles à considérer. Ceci soulève une question importante à savoir comment choisir des structures physiques appropriées pour une charge de requêtes donnée ?.

Dans cette thèse, nous nous concentrons sur la recommandation automatique de deux types de structures physiques : les index de jointures binaires et les fragments verticaux dans le contexte des entrepôts de données relationnels modélisés par un schéma en étoile. Plus précisément, nous étudions l'applicabilité de solutions guidées par la fouille de données. Nous considérons que les problèmes étudiés peuvent être formalisés et résolus avec des techniques de la fouille de données.

Tout d'abord, nous considérons la sélection d'index comme un problème typique d'extraction des motifs fréquents. Les index sont construits avec des combinaisons d'attributs, vus en tant qu'items. Les requêtes de la charge de travail, vues comme des transactions, sont décrites par les attributs qu'elles référencent. Le fondement de notre approche est la notion de motifs fréquents maximaux. Cette technique permet de découvrir les éventuelles corrélations entre les attributs. En évitant la génération des index redondants, l'approche proposée conduit à une solution qui exprime l'ensemble des index pertinents de manière plus succincte. La minimisation du nombre d'index pertinents est une direction intéressante pour minimiser par la même voie l'espace de stockage requis.

D'autre part, les travaux existants ont souvent considéré le coût de la charge de travail comme le facteur clé pour recommander une configuration d'index, même si la configuration choisie pourrait être très coûteuse en terme d'espace de stockage. Contrairement à cette démarche de résolution, nous suggérons de considérer un ensemble de solutions optimisées en proposant une métrique d'évaluation permettant de guider la prise en considération de certaines configurations intéressantes que l'on peut négliger.

Dans la deuxième partie du travail, nous abordons le problème de la fragmentation verticale. Nous montrons qu'il est simple et efficace d'exploiter les propriétés intéressantes des représentations condensées des motifs fréquents afin de fragmenter une table. Nous démontrons que notre approche explore un espace de recherche très réduit pour proposer un schéma de fragmentation pertinent. Nous avons, ensuite, étudié le problème de la fragmentation verticale dans le contexte des entrepôts de données relationnels. Notre motivation est d'aborder un problème encore peu étudié dans la littérature.

Inspiré par le fait que les requêtes dans une charge de travail présente souvent de fortes dépendances, nous proposons une approche basée sur la classification automatique pour fragmenter la table des faits d'un entrepôt de données. La classification envisagée permet de mieux connaître les références réelles des attributs et offre ainsi un éclairage intéressant pouvant aider dans le processus de la fragmentation. Des modèles de coûts théoriques ont été également proposés pour estimer la pertinence des schémas de fragmentation recommandés.

Mots clefs : Entrepôts de données, conception physique, schéma en étoile, sélection d'index, fragmentation verticale, modèle de coût, motifs fréquents maximaux, classification.

ABSTRACT

Data warehouse systems exploit optimization techniques such as indexing and partitioning (fragmentation) to improve query performance, potentially by orders of magnitude. It is therefore important for a data warehouse administrator to choose the appropriate configuration of these physical structures.

Deciding on the physical design of a data warehouse is not an easy task, since it involves searching a vast space of possible alternatives. This raises the important question of how to choose the appropriate physical design for a given workload ?.

In this thesis, we focus on recommending two types of physical structures : Bitmap Join Indexes and vertical fragments for relational data warehouses given a query workload. More precisely, the applicability of data mining-driven approaches in the physical design context is investigated. This thesis states that the studied problems can be modeled as data mining problems and solved efficiently exploiting the algorithms that are commonly used in the data mining field.

First, we consider the index selection as a typical frequent itemsets mining problem. The indexes are built with combinations of attributes, viewed as items. The queries in the workload, viewed as transactions, are described by the attributes they involve. The foundation of our approach is the concept of maximal frequent itemsets. This data mining technique helps to discover strong correlations among attributes such that the presence of some attributes in a query will imply the presence of some other attributes. Moreover, by avoiding the generation of redundant indexes, the proposed approach leads to a solution that expresses the set of relevant indexes in a more succinct way. Consequently, it guarantees the reduction of the storage space requirements.

The existing literature, on the other hand, have often considered the workload cost as the key factor to recommend an index configuration even though the selected configuration might be costly in terms of storage space. Unlike previous approaches that focus on the configuration leading to the minimum workload cost, we suggest to consider a set of optimized solutions and we propose a metric for measuring the profit-effectiveness that helps to pick up the most promising one.

Second, we address the problem of vertical fragmentation. We show that it is simple and efficient to exploit the interesting properties of the condensed representations of frequent itemsets in order to vertically split a table. We demonstrate that our approach explores a very small search space to offer a relevant fragmentation scheme. Then, we studied the problem of vertical fragmentation in the context of

relational data warehouses. Inspired by the fact that the queries in a workload often presents strong dependencies, we propose a clustering based approach to fragment the fact table in a data warehouse. Clustering is one of the major data mining techniques that aims at grouping the data objects into meaningful classes (clusters) such that the similarity of objects within clusters is maximized, and the similarity of objects from different clusters is minimized. This can serve to group queries with similar references. The clustering results can drive and facilitate the fragmentation process. Theoretical cost models are also proposed to estimate the relevance of the recommended fragmentation schemes.

Keywords : Data warehouse, physical design, star scheme, index selection, vertical fragmentation, cost model, maximal frequent itemsets, clustering.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

1

PROBLÉMATIQUE ET OBJET DE LA THÈSE

Sommaire

1.1 contexte

Les systèmes de bases de données relationnelles ont considérablement évolué depuis leur apparition dans les années 70. La croissance sans cesse et la profusion des capacités de stockage ont contribué à l'explosion du volume de données recueillies dans les entreprises. On estime que le volume de données stockées dans le monde double tous les 20 mois [1]. Il s'agit notamment des données de ventes, de médecine, des finances et des domaines sociaux, représentant toutes les activités de notre société. Cependant, cette information produite par l'entreprise est souvent surabondante, non organisée et éparpillée dans de multiples systèmes opérationnels hétérogènes. En conséquence, les organisations possèdent des milliards d'octets de données et deviennent de plus en plus riche en données mais elles restent pauvres en connaissances [2][3][4]. Il est important de préciser la distinction entre les termes donnée, information et connaissance [5] :

- **Les données** sont les résultats de l'observation. Par exemple le nombre d'unités achetées pour un produit donné,
- **Les informations** sont les résultats de l'analyse des données. Par exemple, la répartition géographique de tous les produits achetés,
- **Les connaissances** définissent les information utiles pour la prise de décision. Par exemple, la découverte du mauvais emplacement de certains magasins de vente.

Les systèmes de gestion de bases de données (SGBD) basés sur le modèle OLTP (On Line Transaction Processing) permettent l'automatisation des tâches élémentaires (insertion, modification,...) d'une manière rapide et sûre. Cependant, ils sont particulièrement adaptés au traitement de petits volumes de données. Confrontés avec la collecte de données énormes, les entrepôts de données (data warehouses en

anglais), basés sur le modèle OLAP (On-Line Analytical Processing) ont apporté une solution adéquate et efficace au problème du stockage et de la gestion de grands volumes de données [6].

En plus de sa vocation de stockage, la modélisation d'un entrepôt est complètement dédiée à l'analyse de ses données. Un entrepôt de données permet de stocker, sur de longues périodes, des mesures sur les activités d'un organisme, ce qui permet de les étudier ultérieurement. Par exemple, l'historisation des ventes d'un produit dans une entreprise permet d'étudier son évolution dans le temps. L'entrepôt s'adresse ainsi à des utilisateurs particuliers, les *décideurs*, qui ont besoin de faire ressortir, à partir des données, des tendances (connaissances) leur permettront de prendre des décisions, voire de mettre en place des stratégies pour la gestion de l'entreprise. Le tableau 1 récapitule les principales différences entre les modèles OLTP et OLAP [7].

	OLTP	OLAP
Utilisation	SGBD	Entrepôt de données
Opération typique	Mise à jour	Analyse
Type d'accès	Lecture écriture	Lecture
Niveau d'analyse	Elémentaire	Global
Quantité d'informations échangées	Faible	Importante
Taille de la base	Faible (quelques Go)	Importante (plusieurs To)
Ancienneté des données	Récente	Historique

TABLE 1: Différences entres les modèles OLTP et OLAP [7]

Le concept d'entrepôt de données a été formalisé pour la première fois par W. Inmon [8]. Il s'agit de constituer *une base de données orientée sujet, intégrée et contenant des informations historiées, non volatiles et exclusivement destinées aux processus d'aide à la décision*. Cette définition met l'accent sur les caractéristiques suivantes [9] :

Orientées sujet : Les données des entrepôts sont organisées par sujet plutôt que par application. Par exemple, une chaîne de magasins d'alimentation organise les données de son entrepôt par rapport aux ventes qui ont été réalisées par produit et par magasin, au cours d'un certain temps.

Intégrées : Les données provenant des différentes sources doivent être intégrées, avant leur stockage dans l'entrepôt de données. L'intégration (mise en correspondance des formats, par exemple), permet d'avoir une cohérence de l'information.

Non volatiles : A la différence d'une base de données classique supportant des requêtes transactionnelles de type OLTP (On-Line Transaction Processing), un entrepôt de données est conçu pour supporter des requêtes de type OLAP (On-Line Analytical Processing). L'interrogation est l'opération la plus utilisée dans le contexte d'entrepôt de données où la mise à jour consiste seulement à alimenter l'entrepôt. Les données de l'entrepôt sont ainsi permanentes et ne peuvent pas être modifiées. Le rafraîchissement de l'entrepôt consiste à ajouter de nouvelles données, sans modifier ou perdre celles qui existent. On y intègre seulement de nouvelles données datées qui viennent s'ajouter aux précédentes.

Historiées : La prise en compte de l'évolution des données est essentielle pour la prise de décision. Par exemple, on utilise des techniques de prédiction en s'appuyant sur les évolutions passées pour prévoir les évolutions futures.

Dans l'environnement relationnel, les entrepôts de données sont très souvent modélisés par un schéma en étoile (Figure 1) [10][11]. C'est le modèle principal adopté et utilisé par la majorité des SGBD commerciaux [12]. Ce schéma est caractérisé par une table de faits de très grande taille liée à un ensemble de tables de dimension de tailles nettement plus petites. En effet, les tables de dimensions sont les données les plus statiques et grossissent beaucoup moins que la table de faits [13]. La table des faits contient les clés étrangères des tables de dimension ainsi qu'un ensemble de mesures décrivant l'activité de l'entreprise. Les tables de dimension contiennent des données qualitatives qui représentent des axes sur lesquels les mesures ont été collectées. Cette modélisation conceptuelle a pour objectif d'observer les faits à travers des mesures, en fonction des dimensions qui représentent les axes d'analyse [14][15].

Malgré la popularité et la prolifération des entrepôts de données, il reste un problème fondamental auquel ils sont confrontés : leurs performances. En effet, qui dit entrepôts de données dit manipulation de données colossales. De plus, les requêtes appliquées sur un entrepôt ont tendance à être complexes, ce qui nécessite souvent des opérations de calcul coûteuses tels que les jointures et les agrégations. D'autre part, ces requêtes doivent être exécutées sur des tables ayant potentiellement des millions, voir des milliards, d'enregistrements et leur évaluation nécessite ainsi beaucoup de temps [16].

Les requêtes définies sur un schéma en étoile sont appelées *requêtes de jointure en étoile*. Elles sont caractérisées par des opérations de sélection sur les tables de dimension, suivies par de multiple opérations de jointures avec la table des faits.

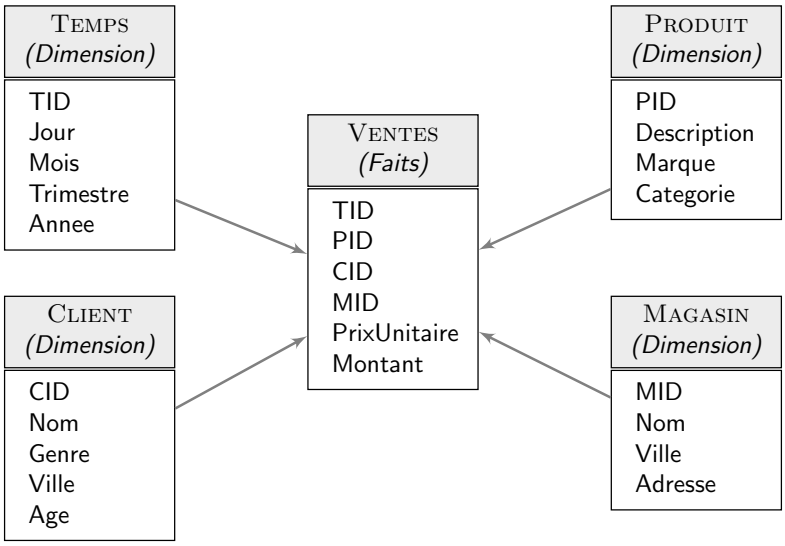


FIGURE 1: Exemple d'un schéma en étoile

Il n'y a aucune jointure directe entre les tables de dimensions [17]. La structure générale d'une requête de jointure en étoile, exprimée en SQL, est illustrée dans la figure 2. De telles requêtes peuvent nécessiter des heures, voir des jours de temps d'exécution [18]. Un tel temps de réponse ne peut être toléré dans l'environnement décisionnel des entrepôts de données. L'attente des utilisateurs d'un entrepôt est surtout une réponse très rapide ou du moins dans un délai acceptable.

Pour un schéma logique donné, les requêtes aboutissent toujours au même résultat quel que soit le schéma physique mis en place. Cependant, le coût de traitement (temps d'exécution) d'une requête peut varier de plusieurs ordres de grandeur entre différentes implémentations physiques du même schéma logique. Par conséquence, le choix de la conception physique à mettre en place a un impact important sur les performances de l'entrepôt. La conception physique de l'entrepôt consiste non seulement à la spécification détaillée des données et de leurs types, mais surtout à la sélection des techniques d'optimisation (index, fragments, vues matérialisées,...,etc.) appropriées susceptibles d'améliorer les performances du système en minimisant les temps nécessaires à l'évaluation des requêtes. L'évaluation des performances du système se base sur un ensemble de requêtes utilisateurs les plus fréquentes appelé *charge de travail*. Chaque requête est caractérisée par sa fréquence d'utilisation. Ceci soulève la question importante à savoir *comment choisir les structures physiques appropriées pour une charge de requêtes donnée ?*.

Afin d'optimiser la conception physique des entrepôts de données, plusieurs techniques ont été proposées dans la littérature et ont été classées en deux catégories [17] : techniques redondantes et techniques non redondantes. La différence principale entre les deux types de techniques réside dans le fait que les techniques redondantes

```
SELECT <Attributs>, <Mesures>
FROM F, D1, D2,... Dk WHERE
F.Cle1 = D1.Cle1 and
F.Cle2 = D2.Cle2 and
...
, ...
F.Clek = DK.Clek and
D1.A1 op V1 and
D2.A2 op V2 and
...
...
Dk.Ak op Vk
```

où :

- $D_i.Cle_i$ désigne la clé primaire de la table de dimension D_i,
- $D_i.A_i$ représente un attribut non clé de la table de dimension D_i,
- op est une opération de comparaison $(=, <, >, \leqslant, \dots)$
- V_i est une constante.

FIGURE 2: Structure générale d'une requête de jointure en étoile.

nécessitent un espace de stockage supplémentaire et un coût de maintenance. Toutefois, puisque la plupart des mises à jour des entrepôts de données sont effectués en vrac et en mode ajout, le coût de maintenance n'est pas significatif [19]. La figure 3 illustre les principales techniques d'optimisation et leur classification.

La conception physique des entrepôts de données est un problème d'optimisation qui nécessite des solutions à plusieurs problèmes inhérents, notamment l'indexation et la fragmentation des données. Ces problèmes sont connus pour être Np-complets [21] [22] [23]. Chaque problème peut être résolu avec différentes approches rendant ainsi la conception physique une tâche très difficile nécessitant non seulement du temps et de l'effort, mais aussi beaucoup d'expertise. L'administrateur a la tâche ardue de prendre des décisions relatives à la sélection des structures physiques appropriées. Les décisions retenues ont un impact majeur sur les performances du système. C'est ainsi que les coûts de l'emploi des administrateurs professionnels sont souvent beaucoup plus élevés que les coûts de licences de logiciels de base de données [24].

Exemple 1. *Afin d'expliciter la difficulté de la sélection des structures physique à mettre en place, considérons l'exemple suivant. Soit* n *le nombre d'attributs susceptibles d'être in-*

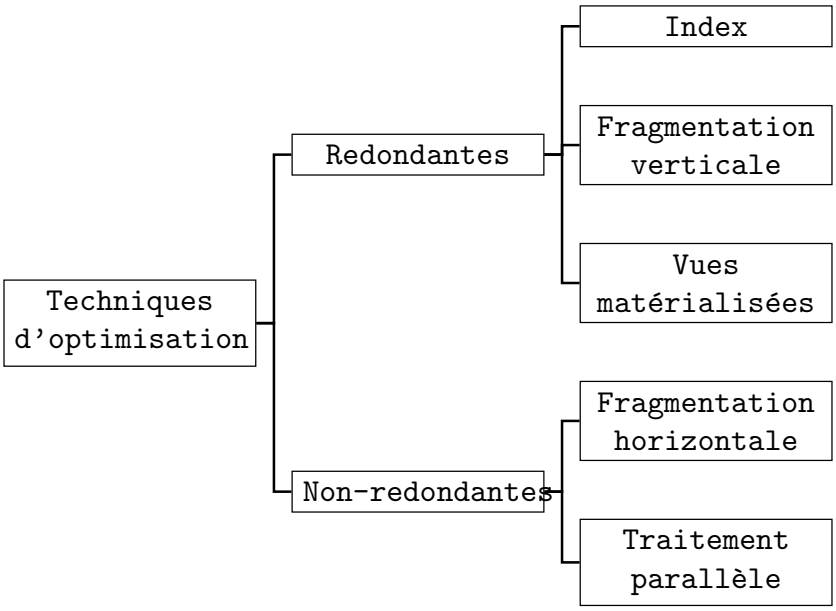


FIGURE 3: Classification des techniques d'optimisation[20]

dexés pour une charge de requêtes donnée. Étant donné qu'un index peut être créé sur un ou plusieurs attributs, alors l'espace de recherche de tous les index possibles contient exactement $2^n - 1$ *index différents. Générer une configuration d'index (une combinaison d'index) optimale consiste intuitivement à énumérer et évaluer l'ensemble des configurations d'une manière exhaustive, soit* $2^{2^n-1} - 1$ *configurations possibles !. Pour n=5, nous devons donc considérer environ* 2×10^9 *configurations différentes.*

Nous pouvons dire que la principale difficulté pour la conception physique réside dans l'énorme espace de recherche des solutions possibles à considérer. Ainsi, cette tâche dépasse de loin les capacités d'analyse manuelle de l'administrateur et nécessite l'utilisation de techniques d'analyse automatisées. C'est pour cette raison que l'automatisation de la procédure de sélection des structures physique optimisées à fait l'objet de plusieurs travaux de recherche dans une perspective d'aide à l'administrateur. L'objectif étant d'évoluer vers des suggestions automatiques en offrant des approches efficaces capables de transiger avec la complexité du problème.

Une approche efficace pour la sélection des structures physiques adéquates repose intuitivement sur l'exploitation des corrélations ou similarités entre les requêtes de la charge de travail. En effet, les requêtes de profils similaires (selon certains critères tels que les attributs référencés et la fréquence d'utilisation) sont optimisées par les mêmes structures physiques. Comprendre et caractériser les requêtes nous paraît alors essentiel pour identifier l'existence de fortes corrélations entre certains ensemble de requêtes et d'en déduire un modèle sur lequel il soit possible de s'appuyer pour identifier automatiquement les structures physiques adéquates pour une

charge de requêtes donnée. Ceci permettra de réduire tant que possible l'espace de recherche à considérer tout en gardant les solutions les plus intéressantes pour l'administrateur. Un examen minutieux des requêtes peut montrer que l'utilisation des attributs présente des régularités. L'identification de ces régularités permet de faire des recommandations plus pertinentes.

Par ailleurs, la fouille de données (data mining en anglais) se présente comme un ensemble de techniques d'analyse automatique constituant le cœur d'un processus appelé Extraction des Connaissances à partir des Données (ECD) visant *la découverte d'informations intéressantes, non triviales, implicites, préalablement inconnues et potentiellement utiles à partir des données* [25]. Ces informations pertinentes sont appelées connaissances ou motifs intéressants dans la littérature dédiée. La fouille de données constitue l'étape mathématique du processus d'extraction des connaissances. Á partir d'une formulation mathématique basée sur les objets, l'idée sous-jacente de la fouille de données est donc d'extraire les connaissances cachées à partir d'un ensemble de données conduisant à identifier des facteurs pertinents utiles pour la prise de décision et la recommandation d'actions. La motivation derrière l'extraction de telles connaissances vient historiquement d'un besoin d'analyse des données issues de transactions de supermarché (appelé traditionnellement le problème du panier de la ménagère) afin d'extraire des règles qui permettent de mettre en exergue des corrélations entre les différents achats des clients dans une même transaction.

Nous pouvons remarquer que le problème de sélection d'une structure physique semble étroitement adaptable aux objectifs d'un processus de fouille de données. Même si à première vue les deux domaines semblent indépendants, ils font appel aux mêmes notions et objectifs : extraire les connaissances cachées a partir d'un ensemble de données conduisant à identifier des facteurs pertinents utiles pour la prise de décision et la recommandation d'actions.

Une vision naturelle de la charge des requêtes stipule l'existence de fortes corrélations entre certains groupes de requêtes. Ces groupes correspondent intuitivement à des requêtes fortement corrélées entre elles qu'avec les autres requêtes de la charge. La détection de telles corrélations, qui peuvent décrire des spécificités communes aux requêtes considérées, peut être naturellement formalisée pour être traitée avec des techniques de fouille de données. L'analyse de la charge des requêtes par des techniques de fouille de données représente un moyen pour aider l'administrateur à trouver des connaissances, préalablement inconnues, qui sont susceptibles de l'intéresser. A titre d'exemple, une connaissance extraite est la suivante : *si une requête* q *référence l'attribut* a, *alors elle référence également les attributs* b *et* c *dans quatre-vingt-dix pour cent des cas.* Une telle connaissance contribue efficacement aux décisions prise

par l'administrateur quant à la sélection de différentes structures physique (index, fragments,. . .,etc.).

Nous souhaitons donc orienter notre travail vers une prise en considération de la régularité d'utilisation des attributs par les requêtes via une adaptation des techniques de la fouille de données. Notre objectif global est d'explorer quelques méthodes de fouille de données et de montrer, en particulier leur potentiel applicatif dans le cadre de notre étude. Cette orientation constitue le point clef de notre travail. Le thème de ce travail se situe donc à un carrefour entre le domaine de la fouille de données et l'automatisation de la conception physique des entrepôts de données. En dépit de sa forte adéquation avec la nature du problème abordé, très peu de travaux proposés sont fondées sur la fouille de données. D'autre part, ce qui rend aussi ce couplage un sujet attractif est le large champ d'applications des techniques de fouille de données qui ont été employées avec beaucoup de succès dans divers secteurs [26]. Nous les avons déjà appliqué à l'analyse des fichiers log des serveurs web [27].

1.2 OBJECTIFS ET CONTRIBUTIONS

Dans cette thèse, nous abordons le problème de l'optimisation de la conception physique des entrepôts de données relationnels modélisés par un schéma en étoile. Notre travail se focalise principalement sur la recommandation automatique de solutions optimisées pour deux techniques : l'indexation et la fragmentation verticale. Aspirant à la proposition de solutions performantes nous nous sommes focalisés sur l'optimisation des structures étudiées en usant de deux techniques de la fouille de données : l'extraction des motifs fréquents et la classification automatique. Ces techniques présentent un intérêt pratique majeur en permettant à partir des données du problème, l'extraction automatique de connaissances , inconnues à priori, utiles pour sa résolution. En exploitant les connaissances extraites, les approches proposées sont en mesure de suggérer des solutions répondant le mieux aux attentes de l'administrateur. Nos recherches se sont axées en premier lieu sur l'investigation des différentes approches proposées pour la résolution des problèmes étudiés. Sur la base d'une analyse de fond, nous avons identifié les limites de ces approches et nous proposons diverses améliorations que nous résumons comme suit :

1. **Problème de la sélection des index avec contrainte de stockage :** Le problème étudié concerne la sélection d'une configuration d'index minimisant le coût d'une charge de requêtes et respectant une limite d'espace de stockage fixée par l'administrateur. Nous avons bénéficié en particulier des propriétés intéressantes des motifs fréquents maximaux pour proposer une nouvelle approche de sélection. Nous nous sommes attaché principalement à établir une garantie

théorique de nos propositions. Notre démarche de résolution vient combler les limitations des travaux antérieurs similaires qui sont basés sur l'extraction des motifs fréquents fermés. Nos améliorations touchent trois points essentiels dans le processus de sélection.

- **Premièrement**, la solution proposée permet un élagage judicieux de l'espace de recherche en évitant complètement la génération dupliquée des index. Elle permet de réduire autant que possible le nombre des index maintenus sans sacrifier leur qualité. La minimisation du nombre d'index pertinents est une direction intéressante pour minimiser par la même voie l'espace de stockage requis. Nous avons montré formellement puis expérimentalement l'intérêt de notre approche. Nous avons en particulier prouver que, quelque soit la limite de l'espace de stockage réservé aux index, notre approche exige toujours un espace inférieur ou, au pire des cas, égal à celui requis par les approches basées sur la recherche des motifs fréquents fermés. Ceci est particulièrement très intéressant quand l'administrateur ne dispose pas d'assez d'espace à allouer aux index. Dans le cas contraire, l'espace épargné par notre approche peut être exploité par l'administrateur pour stocker d'autres structures physiques telles que les vues matérialisées.
- **Deuxièmement**, nous avons étendu le contexte d'extraction utilisé dans les études similaires antérieures. En effet, les travaux antérieurs ont considéré un contexte d'extraction simple où chaque requête est représentée une seule fois par les attributs qu'elle référence. Fixer l'ensemble des fréquences à 1 est une problématique puisque ceci ne prend pas en considération l'importance particulière de chacune des requêtes de la charge. L'information utilisée pour extraire les index pertinents est incomplète, ce qui biaise certainement les résultats de sélection. Pour pallier à ce problème, il faudrait adapter le contexte d'extraction en fonction des fréquences relatives aux requêtes. Nous avons ainsi proposé un nouveau contexte enrichi oú la référence à une requête est dupliquée autant de fois que sa fréquence. Intuitivement, l'exploitation des fréquences de requêtes pour construire le contexte d'extraction permet de promouvoir la sélection des attributs référencés par les requêtes les plus fréquentes. Ceci produira certainement une configuration d'index plus pertinente pour la charge de requêtes considérée. Pour ce point, nous avons également montré formellement puis expérimentalement la pertinence de notre proposition.
- **Troisièmement**, nous avons proposé une modification de la manière de mesurer les performances d'une configuration d'index donnée. En se focalisant sur la configuration conduisant au coût minimal de la charge, les approches

précédentes ont tendance à s'éloigner de la perception pratique et naturelle des administrateurs. Nous sommes convaincus que ce genre de problème n'a pas une seule solution optimisée, mais potentiellement plusieurs, en fonction de la vision de l'administrateur au compromis coût/espace car ces deux paramètres sont généralement divergeants. Nous pensons que ce compromis doit guider la prise en considération de certaines configurations que l'on peut négliger. Ainsi, une mesure de qualité est proposée afin d'exprimer au mieux un tel compromis et permet de recommander une configuration plus pertinente. Notre approche considère la configuration avec le coût minimal coût_{min} comme point de référence. Nous proposons à l'administrateur de spécifier un seuil de tolérance $\Delta_{\text{coût}}$. L'ensemble des solutions retenues est ainsi composé de toutes les configurations respectant l'espace de stockage et ayant un coût inférieur ou égal à $\text{coût}_{min} + \Delta_{\text{coût}}$. Une métrique représentant le rapport entre le coût d'une configuration et son espace de stockage est proposée pour évaluer les solutions ainsi retenues.

2. **Problème de la fragmentation verticale des données :** Dans l'environnement des entrepôts de données relationnels, le coût de traitement des requêtes, caractérisées par leur complexité, est affecté par le volume des données manipulées engendrant ainsi une détérioration de performances. Une alternative intéressante pour contourner ce problème consiste à fragmenter les données de l'entrepôt. Dans cette partie du travail nous abordons le problème de la fragmentation verticale des données. Le problème étudié consiste à diviser une table donnée en un nombre de fragments verticaux de manière à réduire le coût de traitement des requêtes en minimisant le volume des données manipulées.. Cependant, la recherche d'un schéma de fragmentation optimal est un problème Np-complet [28][29] ce qui motive le recours à l'utilisation de méthodes approchées. Dans cette partie de la thèse, nous avons étudié les deux cas de figures du problème : avec et sans spécification préalable du nombre maximal de fragments. Afin de montrer la forte adéquation de la fouille de données pour la résolution de ce problème, nous avons proposé, pour chacun des cas, une solution guidée par une technique différente de fouille de données. Nos principales contributions se résument ainsi :

   a) **Premièrement**, Nous avons proposé une amélioration du travail de Gorla et al. [30] pour la fragmentation verticale d'une table de données. La technique d'extraction des motifs fréquents utilisée dans ce travail est trop coûteuse en générant un nombre important de fragments candidats équivalents (formés des mêmes attributs). Nous avons proposé une approche guidée plutôt par l'extraction des motifs fréquents maximaux. A travers

un exemple pratique, nous avons motivé notre orientation de résolution en mettant l'accent sur le fait que l'extraction de tous les motifs fréquents est tout simplement inadaptée à la résolution du problème étudié. Nous avons plus particulièrement démontré que la complexité du processus de génération des schémas de fragmentation candidats peut être largement simplifiée, tout en assurant une qualité égale de performance. En d'autres termes, nous avons démontré que notre approche explore un espace de recherche très réduit pour proposer les mêmes schémas de fragmentation.

b) **Deuxièmement**, Nous avons étudié le problème de la fragmentation verticale dans le contexte des entrepôts de données relationnels. Notre motivation est d'aborder un problème encore peu étudié dans la littérature. Nous nous sommes penchés en particulier sur la question de la sélection des tables à fragmenter : la table des faits, les tables de dimension ou les deux à la fois ? Nous avons jugé que la fragmentation de la table des faits convient plus à notre contexte. Elle permet d'opter pour une solution cohérente d'un point de vue à la fois de la complexité du processus de la fragmentation et de l'efficacité du traitement des requêtes. En effet, d'une part, la fragmentation des seules tables de dimensions n'est pas un choix adapté pour optimiser les requêtes de jointures en étoile dont le goulot d'étranglement majeur est la jointure de la table de fait centrale (qui est généralement très volumineuse) avec les tables de dimension. D'autre part la fragmentation de l'ensemble des tables de l'entrepôt risque de faire exploser le nombre de jointures nécessaires pour le traitement des requêtes et rendre ainsi la résolution du problème particulièrement complexe.

   Inspiré par le fait qu'une charge de requêtes présente souvent de fortes dépendances entre les requêtes, nous avons exploité la classification automatique pour fragmenter la table des faits d'un entrepôt de données. Il est évident que la découverte de ces dépendances est importante dans le sens oú ces dernières deviennent des hypothèses permettant une meilleure estimation d'un schéma de fragmentation pertinent. L'administrateur est souvent confronté à un manque de connaissances sur ce sujet. Pour lui, un choix optimisé constitue donc un processus difficile et fastidieux puisqu'il s'agit à la fois d'analyser les requêtes de la charge et d'identifier d'éventuelles correspondances pour enfin déduire une solution adéquate. Nous avons noté en particulier l'analogie entre le problème de la fragmentation verticale et celui de la classification automatique. Cette dernière a l'avantage d'exprimer les appartenances des requêtes à des classes homogènes, ce qui permet de mieux connaître les références réelles des attributs et

offre ainsi un éclairage intéressant pouvant aider au processus de la fragmentation. Dans notre contexte, la technique de classification est d'intérêt important pour deux raisons. Tout d'abord, elle est en mesure de mettre en correspondance les fragments générés et les besoins des requêtes utilisateurs afin de recommander les fragments en rapport avec ces besoins. Deuxièmement, elle permet de contrôler le nombre maximal de fragments générés. Notre approche consiste tout d'abord à décrire chaque requête par les attributs qu'elle référence ainsi que sa propre fréquence d'utilisation. Cette description combine deux paramètres complémentaires qui sont, d'une part, l'importance d'une requête pour une charge donnée (par sa fréquence) et, d'autre part, le pouvoir de discrimination de cette même requête (par ses attributs). Une étape de classification des requêtes est ensuite réalisée afin d'identifier des groupes homogènes de requêtes. Finalement, les attributs référencés par les requêtes de chaque groupe identifié constitue un fragment du schéma final. Sous l'angle de cette vision, le degré de pertinence d'un fragment relativement à un ensemble de requêtes est perçu comme le degré de corrélation entre les requêtes en question. Nous visons à proposer une solution adaptative qui prend en compte les caractéristiques communes aux requêtes dans le but d'améliorer l'efficacité du processus de fragmentation. Le but à atteindre étant d'obtenir des fragments (idéalement un fragment) conformes aux besoins d'un groupe donné de requêtes.

c) **Troisièmement**, dans les deux cas de figures étudiés, nous avons proposé des modèles de coût pour estimer la pertinence des schémas de fragmentation générés. Il s'agit de formules mathématiques exprimant les coûts de traitement des requêtes par des estimations aussi simple que possible afin de faciliter, par la suite, la validation expérimentale des solutions proposées. Étant donné que dans les SGBD, le coût de traitement des requêtes est dominé par celui des jointures, nos modèles de coût permettent d'estimer les coûts associés principalement aux différentes jointures induites par les requêtes.

1.3 ORGANISATION DE LA THÈSE

Ce manuscrit est organisé en sept chapitres.

Le **premier chapitre** constitue une introduction à notre travail de thèse. L'objectif de ce chapitre est de mettre en contexte l'utilisation des techniques de fouille de

données pour la résolution du problème de la conception physique des entrepôts de données.

Le **second chapitre** est consacré à la présentation des notions de base pour la fouille de données et plus particulièrement celles relatives à l'extraction des motifs fréquents et la classification automatique, techniques utilisées pour résoudre les problématiques abordées. Ce chapitre expose une rapide et bien évidemment non exhaustive présentation des techniques utilisées. L'objectif du chapitre est de donner une vue d'ensemble des ces techniques, et d'en montrer l'intérêt pratique. Nous profiterons aussi de ce chapitre introductif pour exposer les notions dont nous aurons besoin dans la suite de cette thèse.

Le **troisième chapitre** répond principalement à deux objectifs. Le premier est de montrer l'intérêt de la notion d'indexation des données et de formaliser le problème de sélection d'index. Le second objectif est de passer en revue un état de l'art sur les principaux travaux proposés pour la résolution de ce problème. Il s'agit de donner une vue d'ensemble des méthodes proposées, et d'en illustrer la diversité. Nous finirons par nous intéresser particulièrement aux méthodes utilisant la fouille de données car nous allons nous-même fonder nos propositions sur les limites de ces dernières.

Le **quatrième chapitre** présente notre approche de résolution du problème d'index. Basée sur le concept des motifs fréquents maximaux, l'approche proposée permet de recommander un ensemble réduit d'index qui sont représentatifs et non redondants. Les motifs fréquents maximaux présentent des propriétés théoriques intéressantes qui n'ont pas été exploités par les approches précédentes. Nous montrons la consistance de l'approche proposée, tant d'un point de vue formel qu'expérimental.

Le **cinquième chapitre** présente les notions générales sur la fragmentation des bases de données. Nous nous intéressons ensuite plus particulièrement à la fragmentation verticale dans les bases et les entrepôts de données relationnels. Après une description des différents types de la fragmentation des données, nous présentons une formalisation et la complexité du problème de sélection d'un schéma de fragmentation verticale. Nous exposons ensuite un état de l'art sur la problématique étudiée.

Le **sixième chapitre** présente nos approches de résolution pour le problème de la fragmentation verticale des données. Les approches proposées consistent à analyser, à l'aide de méthodes de fouille de données, un ensemble de requêtes, afin d'en extraire des relations significatives aidant à la résolution de la problématique abordée. Nous allons étudier les deux cas de figures du problème : avec et sans spécification préalable du nombre maximal de fragments. Afin de montrer la forte adéquation de la fouille de données pour la résolution de ce problème, nous allons proposer,

pour chacun des cas, une solution guidée par une technique différente de fouille de données. Nous présentons également nos modèles de coût pour l'évaluation des schémas de fragmentation recommandés.

Le **septième chapitre** achève ce manuscrit par un bilan et quelques perspectives concernant les différents travaux effectués.

2

FOUILLE DE DONNÉES : NOTIONS DE BASE

Sommaire

2.1 INTRODUCTION

Ce chapitre a pour but de présenter les notions de base sur la fouille de données et plus particulièrement celles relatives à la recherche des motifs fréquents et la classification automatique, techniques utilisées pour résoudre les problématiques abordées. Cette présentation est non seulement nécessaire pour montrer l'intérêt des techniques choisies, mais également pour comprendre la suite de notre travail. Nous proposons ici une rapide et bien évidemment non exhaustive présentation des techniques utilisées. Notre objectif est de donner une vue d'ensemble des ces techniques, et d'en montrer l'intérêt pratique, ce qui nous permet d'envisager leur utilisation aussi bien pour la modélisation que pour la résolution des problématiques étudiées. Nous illustrons chacune des deux techniques par les algorithmes les plus connus dans le domaine et qui sont couramment utilisées et disponibles dans la plupart des outils de fouille de données.

Dans un premier temps, nous allons présenter la technique de recherche des motifs fréquents ainsi que les concepts associés. Nous illustrons les prémices de cette technique avec l'algorithme *Apriori* et nous verrons son évolution vers la notion des représentations condensées des motifs fréquents. Nous aborderons, ensuite, la classification automatique par partitionnement. Les définitions et notations nécessaires sont d'abord exposées. Enfin une illustration est présentée avec l'algorithme *k-means*.

2.2 DES DONNÉES AUX CONNAISSANCES

De nos jours les bases de données sont utilisés partout. Nous assistons ainsi à un accroissement considérable de la quantité d'informations stockées dans les bases de données. Cependant, notre capacité à en extraire des informations à forte valeur ajoutée reste limitée [31]. Pour répondre à cette pénurie de connaissances sur les données, de nouvelles méthodes d'extraction de l'information ont vu le jour, regroupées sous le terme générique de l'extraction des connaissances à partir des données, ECD (en anglais : Knowledge Discovery in Databases, KDD) [32]. Bien que les termes ECD et fouille de données (data mining en anglais) sont souvent utilisés comme des synonymes, ils ne désignent pas réellement la même chose. Plus précisément la fouille des données ne représente qu'une étape d'un processus plus général de découverte de connaissances dans les bases de données. Ce dernier ne se limite pas à l'extraction proprement dite des connaissances, mais il comporte plusieurs phases (Voir figure 4) [22] :

1. **Phase de sélection des données :** Il faut tout d'abord récupérer les données à analyser qui peuvent être issues de plusieurs sources différentes.
2. **Phase de pré-traitement :** Cette phase inclut un traitement éventuel des données sélectionnées dans la phase précédente. Il peut s'agir de leur fusion et la correction des valeurs manquantes ou erronées. Finalement les données sont mises en forme dans un format exploitable par les outils de fouille de données. Le pré-traitement des données est ainsi une phase indispensable avant qu'elles puissent être réellement exploitées dans un processus ECD.
3. **Phase de fouille de données :** Elle concerne l'utilisation d'un algorithme d'extraction de motifs sur les données préparées. Dans cette phase, l'utilisateur doit choisir le type de motifs qu'il souhaite extraire (motifs fréquents, règles d'association, clusters, etc.), et fixer les paramètres des algorithmes correspondants.
4. **Phase d'interprétation des résultats :** Les techniques de fouille de données permettent de découvrir des propriétés dans les données analysées. Cependant, ces propriétés ne peuvent pas être considérées comme de nouvelles connaissances tant qu'elles n'ont pas été validées par un expert humain. En effet, les

propriétés extraites peuvent être non intéressantes, ou voire même provenir d'erreurs dans les données. C'est pourquoi les choix relatifs à chacune des étapes du processus ECD peuvent être remis en cause. Il convient alors de réitérer ce processus en faisant des choix différents.

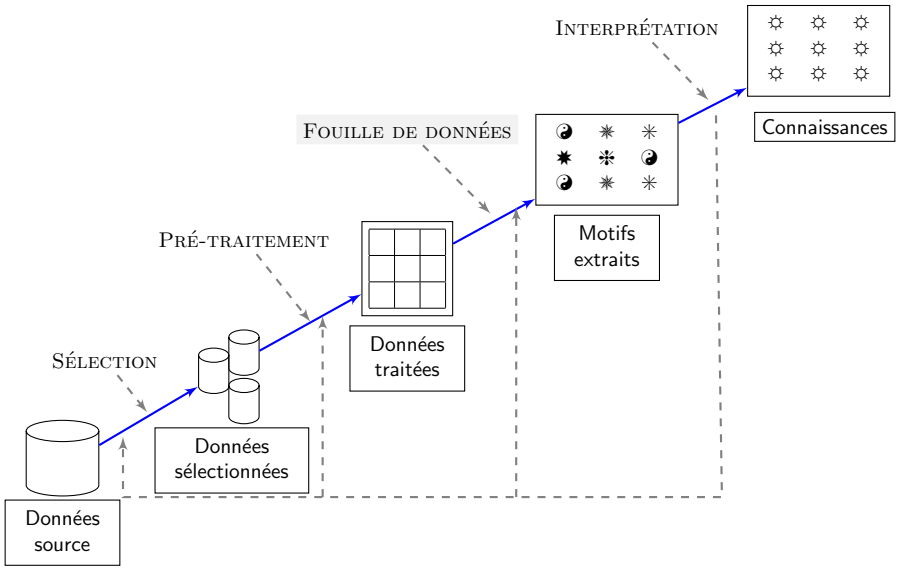


FIGURE 4: Processus d'extraction des connaissances à partir des données

La fouille de données concerne donc spécifiquement les méthodes permettant de faire ressortir des régularités ou des corrélations dans un ensemble de données. Dans un processus ECD, l'étape de fouille de données est la partie la plus complexe du point de vue algorithmique [33]. Dans ce qui suit, nous nous intéressons plus spécifiquement à deux méthodes descriptives (ou exploratoires) qui visent à mettre en évidence des connaissances existantes mais noyées dans les données : la recherche des motifs fréquents et la classification automatique.

2.3 RECHERCHE DES MOTIFS FRÉQUENTS

Le problème de la recherche des motifs fréquents, introduit dans [34], fut initialement posé pour l'analyse des bases de données des transactions de ventes (analyse du panier de la ménagère). Étant donné un ensemble de transactions, formées par les listes d'articles achetés, le but est d'identifier les groupes d'articles fréquemment achetés ensemble. Une conséquence d'une telle identification est, par exemple, de lancer des offres promotionnelles liées à la consommation des clients. D'une manière

plus générale, la recherche des motifs fréquents vise à identifier, parmi un ensemble d'objets, ceux référencés fréquemment ensemble. L'extraction des motifs fréquents est l'une des tâches clés de la fouille de données [35]. Outre que son intérêt théorique, elle a de nombreuses applications dans plusieurs domaines tels que la gestion de la relation client, la biologie moléculaire (relations entre les gènes) et la recherche de l'information [36] [37] [38]. La section suivante définit et illustre par des exemples les principaux concepts liés à la tâche d'extraction des motifs fréquents.

2.3.1 *Cadre formel-Définitions*

Definition 1 (Contexte d'extraction). *Un contexte d'extraction est un triplet* $\mathcal{C} = \{\mathcal{T}, \mathcal{I}, \mathcal{R}\}$ *oú* $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ *est un ensemble fini d'objets (ou transactions) appelé base transactionnelle, et* $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ *est un ensemble fini d'attributs (ou items) et* $\mathcal{R}$ *une relation binaire entre* $\mathcal{T}$ *et* $\mathcal{I}$ *vérifiant* $\mathcal{R} \subseteq \mathcal{T} \times \mathcal{I}$. $\forall\ t \in \mathcal{T}$ *et* $\forall\ i \in \mathcal{I}$. *On dit que la transaction* t *contient l'item* i, *si* $(t, i) \in \mathcal{R}$.

Exemple 2. *Soit* $\mathcal{T} = \{t_1, t_2, \ldots, t_{10}\}$ *une base transactionnelle et* $\mathcal{I} = \{a, b, c, d, e\}$ *un ensemble d'items. Le tableau 2 représente un contexte d'extracion. Les lignes correspondent à l'ensemble des transactions et les colonnes à l'ensemble des items. Ainsi, la transaction* t_1 *contient les items* a *et* e. *Par contre, elle ne contient pas l'item* c.

t_{id}	Items (Attributs)
t_1	a b e
t_2	b d
t_3	b c
t_4	a b d
t_5	a b c
t_6	b c e
t_7	a c
t_8	a b c e
t_9	a b c
t_{10}	b c

TABLE 2: Exemple d'un contexte d'extraction

Definition 2 (un motif (itemset)). *Soit $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ un ensemble fini d'items. Un motif (itemset) est un sous-ensemble non vide de $\mathcal{I}$. L'ensemble des motifs est ainsi l'ensemble de tous les sous-ensembles non vides de $\mathcal{I}$. Un motif constitué de k ($1 \leqslant k \leqslant m$) items est appelé un k-motif (k-itemset).*

Definition 3 (Support d'un motif). *Soit $m = \{i_1, i_2, \ldots, i_k\} \subseteq \mathcal{I}$ un motif. Le support de m, noté $\mathrm{support}(m)$ définit la proportion des transactions contenant m.*

$$\mathrm{support}(m) = \frac{|\{t \in \mathcal{T} | m \subseteq t\}|}{|\mathcal{T}|}$$

Dans le contexte précédent (Table 2), on a $\mathrm{support}(a) = \frac{6}{10}$ et $\mathrm{support}(abc) = \frac{3}{10}$.

Definition 4 (Motif fréquent). *Etant donné un seuil σ, appelé support minimum, un motif m est dit fréquent relativement à σ si $\mathrm{support}(m) \geqslant \sigma$.*

Definition 5 (Problème de recherche des motifs fréquents). *Etant donné une base de transactions $\mathcal{T}$ et un support minimum σ, l'objectif est de trouver tous les motifs fréquents relativement à σ.*

2.3.2 *Illustration de la recherche des motifs fréquents par l'algorithme Apriori*

L'extraction des motifs fréquents, étant donnés une base transactionnelle $\mathcal{T}$ et un support minimum σ, est une tâche très complexe. Nous pouvons déjà observer que l'espace de recherche est exponentiel par rapport au nombre des items dans la base transactionnelle considérée. En effet, si $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ est l'ensemble des items alors le nombre des motifs est de $2^m - 1$. Une méthode naïve pour la résolution de ce problème consiste à considérer tous les motifs possibles en calculant leurs supports. La complexité de cette approche, due à l'explosion combinatoire du nombre de motifs, est un inconvénient majeur. Si, par exemple, $m = 20$ on devra considérer $2^{20} - 1 = 1048575$ motifs !. Imaginons une base de transactions relative aux ventes de 1000 article différents !. Une propriété fondamentale permettant de réduire l'espace de recherche est la suivante :

Propriété 1 (Antimonotonicité). *Si x et y sont deux motifs tels que $x \subseteq y$, alors on a :*

$$\mathrm{Support}(x) \leqslant \sigma \Rightarrow \mathrm{Support}(y) \leqslant \sigma$$

Autrement dit : tout sur-motif d'un motif non fréquent est non fréquent.

Démonstration.

$$
\begin{aligned}
\text{On a :}\quad & \forall\, t \in \mathcal{T} : y \subseteq t \Rightarrow x \subseteq t && \text{car } x \subseteq y \\
\Rightarrow\quad & \{t \in \mathcal{T} \mid y \subseteq t\} \subseteq \{t \in \mathcal{T} \mid x \subseteq t\} \\
\Rightarrow\quad & |\{t \in \mathcal{T} \mid y \subseteq t\}| \leqslant |\{t \in \mathcal{T} \mid x \subseteq t\}| \\
\Rightarrow\quad & \frac{|\{t \in \mathcal{T} \mid y \subseteq t\}|}{|\mathcal{T}|} \leqslant \frac{|\{t \in \mathcal{T} \mid x \subseteq t\}|}{|\mathcal{T}|} \\
\Rightarrow\quad & \mathrm{support}(y) \leqslant \mathrm{support}(x) \\
\text{d'où}\quad & \mathrm{Support}(x) \leqslant \sigma \Rightarrow \mathrm{Support}(y) \leqslant \sigma
\end{aligned}
$$

□

Cette propriété est fondamentale pour les algorithmes d'extraction de motifs fréquents. En effet, si un motif de taille k n'est pas fréquent alors aucun de ces surmotifs ne le sera. Ceci permet d'élaguer l'espace de recherche en ignorant les surmotifs d'un motifs non fréquent.

Depuis l'introduction du problème de recherche des motifs fréquents, plusieurs approches furent développées pour extraire de manière efficace tous les motifs fréquents à partir d'une base de transactions. Nous présentons dans ce qui suit l'algorithme de référence en extraction des motifs fréquents : *Apriori* [34] qui est l'algorithme pionnier dans ce domaine. Il permet une réduction significative du nombre des motifs engendrés par rapport à la méthode naïve en tirant profit de la propriété d'anti-monotonicité du support minimum pour élaguer l'espace de recherche. Cet algorithme effectue un parcours dit par niveau selon une approche dite *Générer-élaguer*.

Pendant la première passe ($k = 1$), il calcule l'ensemble F_1 des 1-motifs fréquents et génère l'ensemble C_2 des 2-motifs candidats à partir des 1-motifs fréquents. Dans la seconde passe ($k = 2$), l'algorithme commence par élaguer certains 2-motifs candidats (ceux qui contiennent des 1-motifs non fréquents). Il calcule ensuite les fréquences des motifs restants pour ne retenir que ceux qui sont fréquents dans l'ensemble F_2 et génère l'ensemble C_3 des 3-motifs candidats à partir des 2-motifs fréquents. Ce processus est réitéré selon le même principe pour les motifs de taille supérieure jusqu'à ce qu'aucun nouveau motif candidat ne puisse être formé. Le pseudo-code de Apriori est présenté dans Algorithme 1.

Exemple 3. *Considérons la base transactionnelle de l'exemple précédent (Table 2). Supposons que le support minimal est fixé à 30% (0.3). Les résultats détaillés de chacune des étapes de l'algorithme Apriori sont illustrés dans la table 3.*

Algorithme 1 : Algorithme *Apriori*

Données : Une base transactionnelle $\mathcal{T}$ et un support minimum σ.

Résultat : Ensemble $\mathcal{F}$ de tous les motifs fréquents dans $\mathcal{T}$ par rapport à σ.

```
1  début
2      F = ∅ ; F_1 = {1 − motiffréquent};
3      pour (k = 2; F_{k−1} ≠ ∅; k++) faire
4          C_k =motif-gen(F_{k−1}); /*Génération et élagage des nouvaux candidats*/
5          pour tous les (transactions t ∈ T) faire
6              C_t =subset(C_k, t); /*Les candidats ∈ t*/ ;
7              pour tous les (candidats c ∈ C_t) faire
8                  c.count++;
9              fin
10             F_k = {c ∈ C_k|c.count ⩾ σ};
11         fin
12     fin
13     F = ⋃_k F_k
14 fin
```

2.3.3 *Représentations condensées des motifs fréquents*

Le nombre des motifs fréquents présents dans une base transactionnelle est généralement prohibitif surtout lorsque il s'agit de données fortement corrélées et/ou d'un seuil de support minimal très bas [39]. Ceci constitut un inconvénient double. D'une part le temps d'extraction des motifs est significatif. D'autre part, et d'un point de vue pratique, les motifs les plus pertinents sont alors noyés au milieu de connaissances triviales ou redondantes et restent donc difficilement utilisables par les experts [40].

Afin de remédier à ces inconvénients, plusieurs travaux ont été proposés en introduisant la notion de la représentation condensée des motifs fréquents [41]. Les représentations condensées fournissent une solution au problème de l'extraction de motifs fréquents en proposant un résumé de ces derniers, tout en assurant de pouvoir reconstruire l'ensemble des motifs fréquents si nécessaire. Ceci va permettre de réduire non seulement la complexité du processus d'extraction mais également le nombre de motifs extraits. D'autre part, les représentations condensées autorisent des usages multiples des motifs fréquents [41] ce qui est un point clé dans de nombreuses applications pratiques [39]. Les récents progrès sur l'extraction des motifs fréquents se situent autour de la notion de représentation condensée des motifs [42]

Etape 1

Motif	support
a	0.6
b	0.9
c	0.7
d	0.2
e	0.3

$\Rightarrow \quad F_1 = \{a\},\{b\},\{c\},\{e\} \quad \Rightarrow \quad C_2 = \{a,b\},\{a,c\},\{a,e\}$
$\{b,c\},\{b,e\},\{c,e\}$

Etape 2

Motif	support
a, b	0.5
a, c	0.4
a, e	0.2
b, c	0.6
b, e	0.3
c, e	0.2

$\Rightarrow \quad F_2 = \{a,b\},\{a,c\}$
$\{b,c\},\{b,e\} \quad \Rightarrow \quad C_3 = \{a,b,c\},\{b,c,e\}$

Etape 3

Motif	support
a, b, c	0.3
b, c, e	0.2

$\Rightarrow \quad F_3 = \{a,b,c\} \quad \Rightarrow \quad C_4 = \emptyset$

Ensemble des motifs fréquents

Motif	support
a	0.6
b	0.9
c	0.9
e	0.3
a, b	0.5
a, c	0.4
b, c	0.6
b, e	0.3
a, b, c	0.3

TABLE 3: Illustration de l'algorithme *Apriori* pour l'exemple 2

et leurs propriétés théoriques font toujours sujet de recherche. Nous nous intéressons dans ce qui suit aux deux représentations les plus utilisés en fouille de données : les motifs fréquents fermés [43] et les motifs fréquents maximaux [44].

Definition 6 (Motif fréquent fermé). *Un motif fréquent* m *est dit fermé s'il n'a pas le même support que tout ses sur-motifs, i.e :.*

$$\forall\, n \in \mathcal{I} \;:\; m \subset n \Rightarrow \mathrm{Support}(m) \neq \mathrm{Support}(n)$$

Ceci signifie simplement qu'un motif fréquent fermé est un motif fréquent qui n'est pas inclus dans un autre motif fréquent ayant le même support. Cette représentation est alors une alternative dont l'objectif est de réduire la taille de l'ensemble des motifs fréquents générés. Néanmoins, même l'ensemble des motifs fréquents fermés peut être trop grand surtout lorsque il s'agit de données fortement corrélées. Dépendant de la corrélation des données en question, le nombre des motifs fréquents fermés peut être presque égal au nombre des motifs fréquents [45]. L'utilisation des motifs fréquents fermés dans pareils contextes réduit leurs performances puisque l'espace de recherche des motifs fréquents fermés tend à se superposer avec celui des motifs fréquents [46]. Du point de vue de l'analyste ou du décideur, l'interprétation de ces motifs est souvent fastidieuse car ils contiennent généralement beaucoup de redondances empêchant ainsi de trouver les pépites de connaissances [47] ce qui affaiblit l'intérêt pratique de cette représentation.

Definition 7 (Motif fréquent maximal). *Un motif fréquent* m *est dit maximal si tout ses sur-motifs sont infréquents, i.e :.*

$$\forall\, n \in \mathcal{I} \;:\; m \subset n \Rightarrow \mathrm{Support}(n) < \sigma$$

Les motifs fréquents maximaux, appelés aussi bordure positive dans la littérature, représentent une frontière entre les motifs fréquents et les motifs non fréquents et résument ainsi tous les motifs fréquents. L'ensemble des motifs fréquents peut être simplement dérivé à partir de l'ensemble des motifs fréquents maximaux par exploitation de la propriété d'anti-monotonicité : chaque sous-ensemble non vide d'un motif fréquent maximal est un motif fréquent. Remarquons que par définition même l'ensemble des motifs fréquents maximaux correspond à un sous-ensemble des motifs fréquents fermés.

Exemple 4. *Le tableau Table 4 illustre l'ensemble des motifs fréquents, fréquents fermés et fréquents maximaux pour l'exemple précédent (Table 2). Le nombre des motifs extraits est, selon le type, de 9, 8, et 2 respectivement. Remarquons que l'ensemble de tous les motifs fréquents peut être déduit des seuls motifs maximaux* $\{b, e\}$ *et* $\{a, b, c\}$.

Motifs fréquents	Motifs fréquents fermés	Motifs fréquents maximaux
a	a	b, e
b	b	a, b, c
c	c	—
e	a, b	—
a, b	a, c	—
a, c	b, c	—
b, c	b, e	—
b, e	a, b, c	—
a, b, c	—	—

TABLE 4: Motifs fréquents, fréquents fermés et fréquents maximaux pour l'exemple 3

Exemple 5. *Afin de montrer l'intérêt des représentations condensées, considérons un exemple plus pratique. Le tableau 5 résume le nombre des différents types des motifs fréquents extraits pour quelques valeurs du support minimum, pour la base de données* ***Mushroom****. C'est une base de données de test en fouille de données qui est librement disponible sur Internet*[1]*. Les individus (transactions) sont des espèces de champignons décrites par un ensemble d'attributs, tels la forme, la taille, la couleur du chapeau, ..., etc. Elle est composée de* 8124 *transactions et de* 119 *attributs. La taille moyenne des transactions est* 23. ***Mushroom*** *est utilisée en études expérimentales pour déterminer les degrés de corrélation entre les attributs décrivant les champignons. Par exemple, on peut s'intéresser à la variable couleur de chapeau en essayant de déterminer son degré de corrélation avec la taille du champignon. Les différents types de motifs extraits montrent une réduction très significative du nombre des motifs selon la représentation condensée utilisée. On dénombre, par exemple pour un support de* 10%, 4897 *motifs fréquents fermés contre seulement* 558 *motifs fréquents maximaux, représentant ainsi une réduction d'environ* 88.61%. *Ceci est du au fait que les motifs fréquents fermés sont handicapés par la génération d'un nombre important de motifs redondants. En outre, les motifs fréquents maximaux représentent les résultats sans redondance et se révèlent extrêmement efficaces en pratique.*

1. https ://archive.ics.uci.edu/ml/datasets/Mushroom

Support(%)	fréquents	fermés	maximaux	Réduction(%)
10	574513	4897	558	88.61
20	53663	1203	160	86,70
30	2735	427	63	85,25
40	565	140	41	70,72
50	153	45	15	66,67

TABLE 5: Motifs fréquents, fréquents fermés et fréquents maximaux pour la base Mushroom

2.4 LA CLASSIFICATION AUTOMATIQUE PAR PARTITIONNEMENT

2.4.1 *Introduction*

Établir des catégories parmi un ensemble d'entités est une préoccupation universelle. La biologie classifie le vivant, la linguistique classifie les langues, l'économie les marchés, la médecine les maladies... etc. Quel que soit le contexte, l'objectif de la classification est d'établir une partition d'une collection d'objets en sous-ensembles homogènes appelés classes, partitions, ou groupes [48].

Exemple 6. *Considérons la base de données d'une société commerciale ayant des milliers de clients. La base de données contient les données décrivant chaque client (par exemple, le code postal, le salaire annuel, et l'âge) ainsi que des informations sur l'historique des achats de chaque client. Une des préoccupations de l'entreprise est de mieux comprendre ses clients afin d'améliorer ses services et ses produits. Elle souhaite partitionner la base de données des clients en plusieurs groupes homogènes en fonction des caractéristiques qui les décrivent. Par exemple, un groupe intéressant peut être composé des clients qui achètent des produits similaires. Un autre groupe intéressant peut être composé des clients qui habitent loin de la société, mais qui achètent plus que le moyen des clients. Pour chacun des groupes identifiés, des mesures d'amélioration des services seront décidées.*

Dans beaucoup d'applications on ne dispose pas de connaissances à priori sur les données analysées i.e, on ne sait pas si la collection d'objets contient des classes homogènes ou non. Dans ce cas la procédure de partitionnement est appelée classification automatique (clustering en anglais). La classification automatique est l'une des principales tâches d'exploration en fouille de données. Son but est de trouver une organisation des données cohérente et valide, qui puisse mettre en évidence les vraies structures dans un ensemble de données sans aucune connaissance à priori sur les données traitées [49] [50] [51]. Ceci implique l'extraction de l'information pertinente pour la compréhension des données étudiées et permet d'établir d'éventuelles

règles de prise de décision. La classification automatique a fait preuve de son utilité dans le domaine de la fouille des données [51] à travers ses multiples applications, et dans des domaines très diverses [50] [52] [53] [54].

2.4.2 *Formalisation et méthodologie générale de la classification automatique*

Soit un ensemble $\mathcal{O}$ de n objets (individus) décrits par d variables (attributs). Chaque objet X_i $(1 \leqslant i \leqslant n)$ est représenté par un vecteur $\{x_{i1}, x_{i2} \ldots, x_{id}\}$ de taille $1 \times d$. La valeur x_{ij} $(1 \leqslant i \leqslant n$ et $1 \leqslant j \leqslant d)$ représente une mesure d'une variable de l'objet X_i. Les objets sont ainsi représentés par une matrice de la forme :

$$\begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1d} \\ x_{21} & x_{22} & \ldots & x_{2d} \\ \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots \\ x_{n1} & x_{n2} & \ldots & x_{nd} \end{bmatrix}$$

Definition 8 (Une partition). *Une partition d'un ensemble $\mathcal{O}$ de n objets en k classes (k donné) est un ensemble de parties non vides de $\mathcal{O}$, $\{c_1, c_2, \ldots, c_k\}$ verifiant :*

- $\forall j, j' = 1, \ldots, k; j \neq j' : c_j \cap c_{j'} = \emptyset$
- $\cup_{j=1}^{k} c_j = \mathcal{O}$

Il n'est bien entendu pas possible d'énumérer de façon exhaustive toutes les partitions possibles. En effet, le nombre de partitions possibles est très important même lorsque la cardinalité de $\mathcal{O}$ est petite. Si l'on considère le partitionnement d'un ensemble de n objets en k classes, le nombre total de partitions possibles est égal à [55] :

$$\frac{1}{k!} \sum_{j=0}^{k} (-1)^{j-1} \times \binom{k}{j} \times j^n$$

Par exemple, il existe 1701 partitions possibles de 8 objets répartis en 4 classes. C'est ainsi que la plupart des techniques utilisées en classification automatique sont des heuristiques dont le but est la détermination de solutions optimales en se basant sur une mesure de similarité entre les objets à classer. Les objets à classer sont regroupés en fonction de leur similitude de manière à vérifier les deux propriétés suivantes :

1. Les objets d'une même classe (cluster ou groupe) sont aussi similaires que possible : c'est l'homogénéité intra-classe (cohésion).
2. Les objets appartenant à des classes différentes sont aussi différents que possible : c'est l'hétérogénéité inter-classe (séparation).

Le processus de la classification automatique se divise en trois étapes majeures : (1) la sélection des variables (descripteurs des objets), (2) l'application d'un algorithme de classification et (3) l'exploitation des résultats obtenus [55] :

1. **Sélection des variables :** Cette étape consiste à sélectionner les variables encodant le mieux les objets à classer. En effet les variables ne sont pas nécessairement toutes pertinentes pour les objectifs visés par la procédure de la classification.
2. **Application d'un algorithme de classification :** cette étape centrale concerne le choix de la méthode de classification à utiliser et sa paramétrisation. Pour les approches par partitionnement, l'utilisateur doit fixer à priori le nombre k de classes désirées. Un processus itératif est alors utilisé pour chercher la meilleure partition en k classes disjointes.
3. **Exploitation des résultats obtenus :** A l'issue de l'étape de classification, les experts doivent analyser les résultats obtenus afin d'en tirer des conclusions intéressantes et envisager en conséquence la prise de décisions convenables.

2.4.3 *Notion de similarité*

Les techniques de la classification automatique utilisent une mesure de similarité entre les objets à classer. La similarité est un type de comparaison qui permet de juger d'une relation de proximité entre deux objets [56]. La similarité entre deux objets permet de mesurer ce que ces objets ont en commun. Par conséquent, plus ils ont des caractéristiques en commun, plus leur similarité sera importante. Généralement, les mesures de similarité s'appuient sur la notion mathématique de distance. Ainsi, deux objets sont d'autant plus similaires, au sens de cette distance, que leur distance est plus petite.

Soit $\mathcal{O}$ un ensemble d'objets défini dans un espace $\mathbb{R}^d$ (les objets sont décrits par d variables). Une distance entre deux objets o_i et o_j, notée $d(o_i, o_j)$, est une mesure de similarité qui vérifie les propriétés mathématiques suivantes :

1. $\forall o_i, o_j \in \mathcal{O} : d(o_i, o_j) \geqslant 0$ (positivité) ;
2. $\forall o_i, o_j \in \mathcal{O} : d(o_i, o_j) = 0 \Rightarrow o_i = o_j$ (séparation) ;
3. $\forall o_i, o_j \in \mathcal{O} : d(o_i, o_j) = d(o_j, o_i)$ (symétrie) ;
4. $\forall o_i, o_j, o_k \in \mathcal{O} : d(o_i, o_j) \leqslant d(o_i, o_k) + d(o_k, o_j)$ (inégalité triangulaire) :

Chaque domaine d'application possédant ses propres données, possède également sa propre notion de distance adaptée au type des données traitées. Dans ce qui suit, nous présentons un exemple de quelques distances utilisées dans le cas des données

numériques. Le lecteur intéressé peut consulter [57] pour plus d'informations sur d'autres distances utilisées pour d'autres types de données. Une mesure générale de la distance dans le cas des données numériques est la distance de Minkowski. Étant donnés deux objets $O_i = \{x_1, x_2, \ldots, x_n\}$ et $O_j = \{y_1, y_2, \ldots, y_n\}$ définis dans un espace R^n, la distance de Minkowski est définie comme suit :

$$dist(O_i, O_j) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}$$

oú p est un entier positif. Quelques valeurs particulières du paramètre p sont plus souvent utilisées :

- Pour $p = 1$: $dist(O_i, O_j) = \sum_{i=1}^{n} |x_i - y_i|$: Distance de Manhattan ;
- Pour $p = 2$: $dist(O_i, O_j) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$: Distance Euclidienne ;
- Pour $p \to \infty$: $dist(O_i, O_j) = max_{i=1}^{n} |x_i - y_i|$: Distance de Chebychev ;

2.4.4 *Illustration avec l'algorithme k-means*

L'algorithme des k−means [58] est l'outil de classification le plus utilisé dans les applications scientifiques et industrielles [59] tout en étant simple et efficace [60] [61]. Il fonctionne selon le principe suivant : dans une première étape, on choisit aléatoirement k individus qui vont représenter les centres (appelés aussi centroïdes ou centres de gravité) des classes formant la partition initiale. Ensuite, les autres individus sont regroupés autour de ces k centres en affectant chacun d'eux au centre le plus proche. L'étape suivante consiste à recalculer les k nouveaux centres des k classes. Le processus d'affectation des individus et recalcul des nouveaux centres est réitéré jusqu'à la stabilité des classes ainsi établies. Le pseudo-code de l'algorithme k-means est présenté dans Algorithme 2.

Exemple 7. *Le tableau 6 présente un ensemble de 8 objets (des requêtes SQL par exemple) décrit chacun par 2 attributs X et Y (le nombre des attributs et la fréquence par exemple). Supposons, que les objets o_1 , o_4 et o_7 représentent les centres initiaux pour un clustering k-means avec $k = 3$. La figure 5 illustre le déroulement de l'algorithme k−means.*

2.5 SYNTHÈSE DU CHAPITRE

Ce chapitre a permis d'introduire les notions de base pour la fouille de données. Nous nous sommes focalisés sur deux techniques exploratoires à savoir : la recherche des motifs fréquents et la classification automatique par partitionnement.

Algorithme 2 : Pseudo-code de l'algorithme *K-means*

Données : Un ensemble d'objets $\mathcal{O}$ et un nombre de groupes souhaités k.

Résultat : Une partition de $\prod = \{c_1, c_2, \ldots, c_k\}$ de k groupes.

```
début
    Choisir aléatoirement une partition initiale représentée par k centroides
    {c̄1, c̄2, ..., c̄k};
    répéter
        pour tous les (oi ∈ O) faire
            Affecter oi à la classe la plus proche :||oi − c̄j(1⩽j⩽k)|| est minimale ;
        fin
        Recalculer les nouvaux centroides :c̄i = 1/||ci|| Σ_{j=1}^{||ci||} oj i = 1..., k ;
    jusqu'à (Stabilisation des classes) ; ;
fin
```

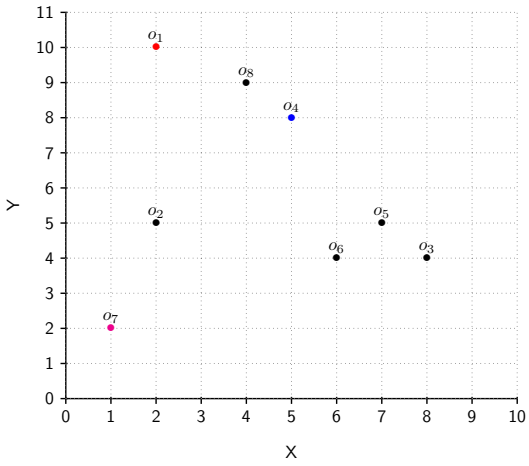
	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8
X	2	2	8	5	7	6	1	4
Y	10	5	4	8	5	4	2	9

TABLE 6: Exemple de données à classer par $k-$means

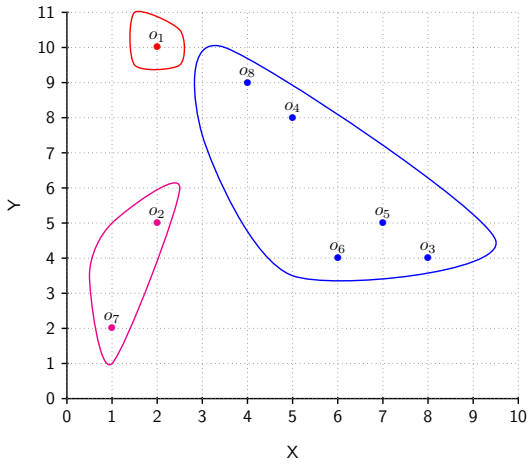
Ces techniques nous intéressent particulièrement puisque nous les avons utilisé pour résoudre les problématiques abordées dans le cadre de ce travail.

Nous avons présenté, dans un premier lieu, le concept et les définitions relatives à la recherche des motifs fréquents. Nous avons ensuite illustré la procédure de leurs extraction par l'algorithme pionnier dans le domaine : *Apriori*. Cependant, ces motifs sont généralement produits en grande quantité ce qui rend leur exploitation difficile en pratique. Des représentations condensées tels que les motifs fréquents fermés et maximaux sont alors proposées pour, d'une part, réduire la complexité de la tâche d'extraction et d'autre part, faciliter l'analyse et l'exploitation des motifs extraits. Ainsi, nous avons vu que les motifs fréquents, fréquents fermés et fréquents maximaux sont des représentations différentes de la même connaissance mais, qui se distinguent par le nombre des motifs générés. Ces différentes représentations autorisent des usages multiples des motifs fréquents ce qui est un point clé dans de nombreuses applications pratiques

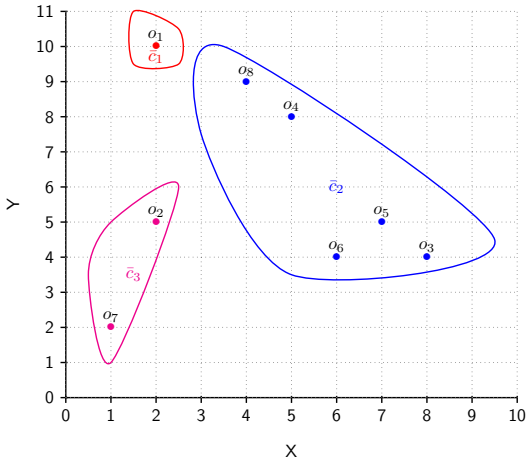
Dans un second lieu, nous avons présenté la notion de la classification automatique par partitionnement dont l'objectif est de regrouper un ensemble d'objets en plusieurs sous-ensembles homogènes. Les objets ayant des caractéristiques communes sont placés dans un même groupe. Les groupes identifiés sont alors utiles



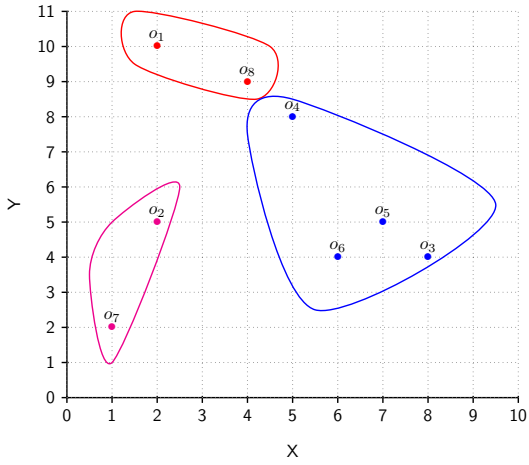
(a) Itération 1.



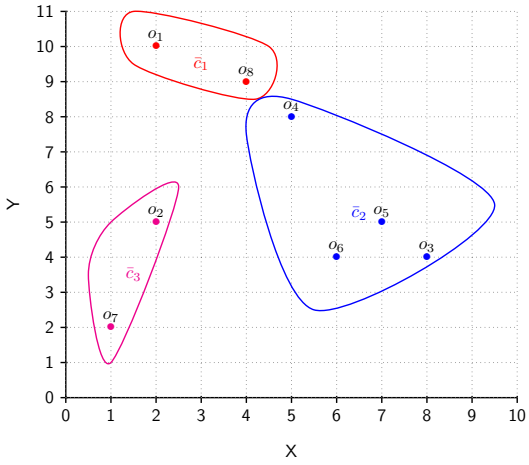
(b) Itération 2



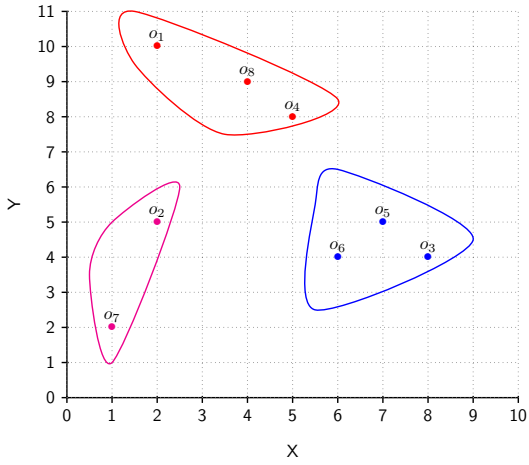
(c) Itération 3



(d) Itération 4.



(e) Itération 5



(f) Itération 6.

FIGURE 5: Illustration de l'algorithme k−means.

pour la prise de décision et la recommandation d'actions. Les définitions et notations nécessaires ont été exposées et une illustration est présentée avec l'algorithme *k-means*.

Ce chapitre a permis de montrer le caractère générique et l'intérêt pratique des techniques exposées. Elles permettent de mettre en exergue ce qui est intéressant du point de vue d'un utilisateur expert, ce qui nous a motivé pour envisager leur utilisation aussi bien pour la modélisation que pour la résolution des problématiques étudiées. Nous avons voulu illustrer dans ce chapitre toutes les définitions et notions que nous jugeons nécessaires pour aborder les problématiques étudiées qui font l'objet des chapitres suivants.

3

PROBLÈME DE SÉLECTION DES INDEX : ÉTAT DE L'ART

Sommaire

3.1 INTRODUCTION

Ce chapitre répond principalement à deux objectifs. Le premier est de montrer l'intérêt de la notion d'indexation des données et de formaliser le problème de sélection d'index. Le second objectif est de résumer les principaux travaux proposés pour la résolution de ce problème.

Nous commençons d'abord par la présentation de la notion d'indexation et son intérêt. Avant de formaliser le problème de sélection d'index, nous présentons les standards des index et plus particulièrement les B-arbre pour les bases de données et les Index de Jointure Binaires (IJB) pour le contexte des entrepôts de données.

La suite du chapitre est une revue de la littérature. Nous présentons les principales approches existantes pour la résolution du problème de sélection d'index que nous classons, selon l'approche de résolution utilisée, en trois catégories. Nous terminons ce chapitre en présentant une synthèse des différents travaux exposés.

3.2 INDEXER OU NE PAS INDEXER N'EST PAS LE PROBLÈME !

Le but d'un index est simple : accélérer l'accès aux données lors de l'exécution des requêtes. Ainsi, la performance d'un SGBD dépend non seulement des réponses aux requêtes soumises, mais également de la vitesse à laquelle les réponses sont repérées, ce qui, à son tour, dépend de la manière dont les données sont indexées. La sélection des index à mettre en place n'a jamais été une tâche facile pour un administrateur, mais sa nécessité n'a jamais été mise en doute. Imaginons par exemple, un livre, d'une centaine de page, sans index. Après plusieurs tentatives inutiles pour localiser les pages, le lecteur pourra "maudire" l'auteur pour ne pas avoir fourni un index des termes. Cet exemple simple illustre l'importance d'un index qui devient, évidement, plus que nécessaire quand la quantité de données (ou le nombre de pages dans l'exemple d'un livre) est grande. Ainsi, pour un administrateur, indexer ou ne pas indexer n'est pas le problème. Le problème fondamental est plutôt : quels sont et comment choisir les index les plus pertinents pour optimiser les performances du système ? [62].

Depuis l'apparition des SGBD relationnels, plusieurs méthodes d'indexation ont été proposées. L'objet de cette section n'est pas de répertorier et de comparer les techniques existantes, mais bien de donner au lecteur un aperçu sur les standards les plus utilisés dans les contextes des bases de données et des entrepôts de données. Cet aperçu sera suivi par un état de l'art sur le problème abordé. Le lecteur essentiellement intéressé par les différents types d'index peut consulter les références [20][63] pour une présentation plus détaillée des types d'index qui ont jalonné l'histoire de ce sujet.

3.3 INDEX B-ARBRE

Les structures d'indexation de données communément utilisées par les SGBD relationnels, tels que Oracle et Mysql, sont les B-arbres (B-trees en anglais) et leurs variantes [64]. Le plus haut niveau de l'index est appelé racine alors que le niveau le plus bas contient les feuilles. Les nœuds des niveaux intermédiaires contiennent des entrées qui pointent vers le niveau suivant dans l'index. Les feuilles pointent vers l'emplacement physique des enregistrements [65][66][67].

La recherche dans un tel arbre est similaire à celle effectuée dans un arbre binaire de recherche, c'est-à-dire en parcourant l'arbre de haut en bas et en choisissant à chaque fois le fils correspondant à la fourchette de valeur que l'on recherche. Une comparaison par rapport à la racine permet de déterminer la partie de l'arbre (à gauche ou à droite) dans laquelle se poursuivra la recherche au niveau immédia-

tement inférieur. L'examen de ce niveau de l'arbre permet à son tour d'affiner l'intervalle de recherche. Ce processus récursif s'applique jusqu'à la rencontre d'une feuille (nœud de plus bas niveau dans l'arbre) contenant la valeur recherchée ainsi que l'adresse physique de la page associée. Un B-arbre offre un excellent compromis pour les opérations de recherche et de mises à jour. Ces qualités expliquent le fait que les B-arbres et leurs variantes soient systématiquement intégrés dans la plupart des SGBD relationnels [63].

RID	IDEmploye	Nom	Grade	Salaire
r1	25	Ali	Comptable	1500
r2	40	Benameur	Ingénieur	2000
r3	21	Youcef	Enseignant	2500
...	...	...	...	...

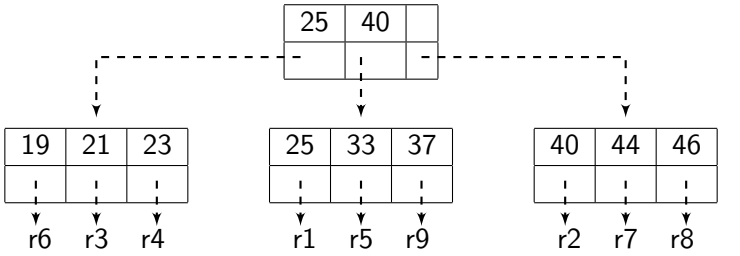


FIGURE 6: Exemple d'un index B-Arbre

Exemple 8 (Index B-arbre). *La figure 6 représente un index B-arbre crée sur l'attribut* ***IDEmploye*** *(identificateur de l'employé) de la table des employés* ***Employe****. Considérons le nœud le plus à gauche des feuilles dans l'index. Il représente trois tuples avec les identificateurs 19, 21, et 23 respectivement. A chaque valeur de ce nœud est associé un pointeur vers le tuple correspondant représenté par son RID. Cette référence spécifie l'emplacement physique du tuple sur le disque.*

3.4 INDEX DE JOINTURE BINAIRE

En raison de l'évolution vers le contexte des entrepôts de données, les techniques conventionnelles d'indexation, telles que les B-arbre, ne sont plus adaptées car les quantités volumineuses de données deviennent plus difficiles à indexer. L'évolution naturelle est donc de chercher à fournir des techniques d'indexation particulièrement

adaptées au contexte des entrepôts de données. Rappelons que les requêtes définies sur un entrepôt de données modélisé par un schéma en étoile sont caractérisées par des opérations de sélection sur les tables de dimension, suivies par de multiple opérations de jointures avec la table des faits [17]. Sachant que les jointures sont les opérations les plus coûteuses dans les SGBD relationnels [68], les index de jointure binaire [69][70][71] ont été proposés pour précalculer et donc optimiser de telles jointures.

Un index de jointure binaire est défini sur la table des faits en utilisant des attributs, non clés, appartenant à une ou plusieurs tables de dimension. Soit A un attribut relatif à une table de dimension D et ayant n valeurs distinctes $v_1, v_2 \ldots, v_n$. Supposons que la table des faits F est composée de m tuples (enregistrements). La construction de l'index de jointure binaire défini sur l'attribut A est réalisée de la manière suivante :

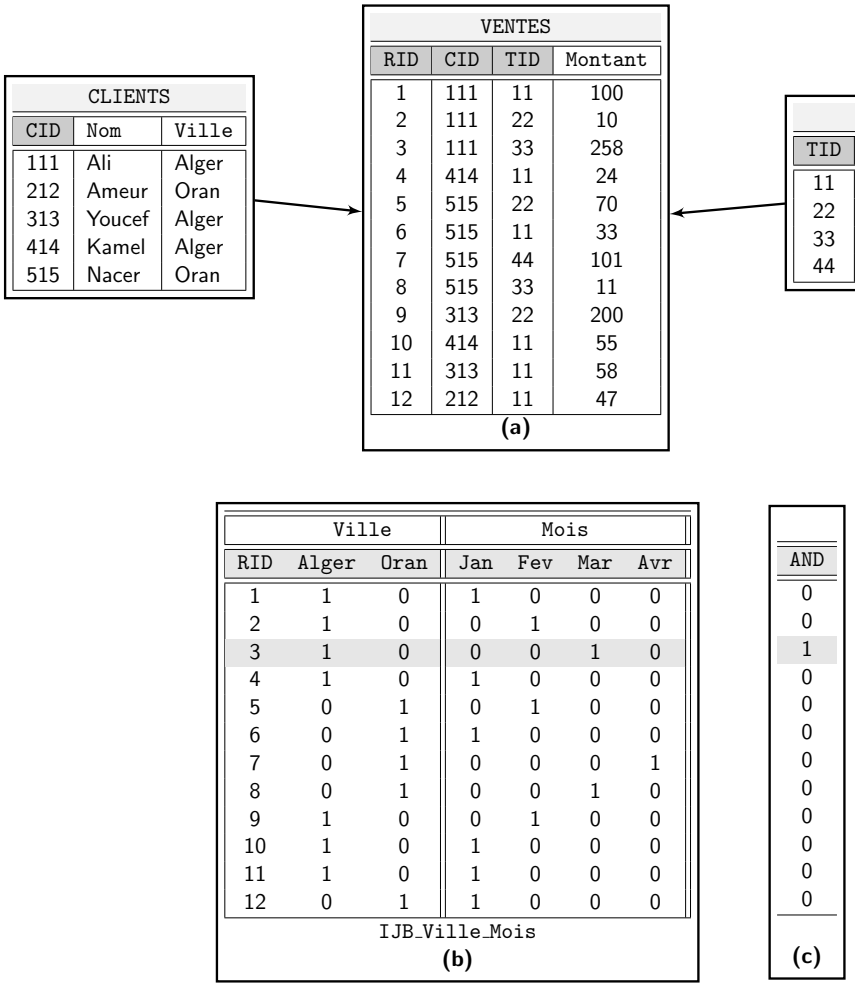
1. Création de n vecteurs composés chacun de m entrées ;
2. Le $i^{ème}$ bit du vecteur correspondant à une valeur v_k est mis à 1 si le n-uplet de rang i de la table des faits est joint avec un n-uplet de la table de dimension D tel que la valeur de A de ce n-uplet est égale à v_k . Il est mis à 0 dans le cas contraire.

Exemple 9 (Index de jointure binaire). *Considérons un entrepôt de données constitué d'une table de faits* `VENTES` *et deux tables de dimension* `CLIENT` *et* `TEMPS` *(7(a)). Soit la requête* q *:*

```
Select count(*)
From Ventes V, Client C, Temps T
Where V.CID = C.CID AND V.TID = T.TID
AND C.Ville = 'Alger' AND T.Mois = 'Mar'.
```

Afin d'optimiser le coût de q*, l'administrateur crée un index de jointure binaire sur les attributs* `Ville` *et* `Mois`*. La figure 7(b) illustre l'index crée. Pour répondre à* q*, l'optimiseur lit les vecteurs de bits associés aux valeurs* `"Alger"` *et* `"Mar"`*, réalise un* `AND` *et en fin il calcule le nombre de* `'1'` *dans le vecteur résultat (Figure 7(c)).*

Les index de jointure binaire sont particulièrement adaptés au contexte des entrepôts de données. Ils sont très bénéfiques pour les requêtes de type Count(*). La réponse à ce type de requêtes ne nécessite aucun accès aux données dans les tables. Il suffit, en effet, de compter le nombre de 1 dans le vecteur résultat des opérations logiques AND effectuées [20]. D'autre part, les opérations logiques sur les bits, qui s'opèrent directement en mémoire vive, les rendent très efficaces pour l'optimisation des opérations de jointures. De plus, l'espace nécessaire au stockage des ces index est réduit, notamment quand la cardinalité des attributs indexés est relativement faible,



CLIENTS

CID	Nom	Ville
111	Ali	Alger
212	Ameur	Oran
313	Youcef	Alger
414	Kamel	Alger
515	Nacer	Oran

VENTES

RID	CID	TID	Montant
1	111	11	100
2	111	22	10
3	111	33	258
4	414	11	24
5	515	22	70
6	515	11	33
7	515	44	101
8	515	33	11
9	313	22	200
10	414	11	55
11	313	11	58
12	212	11	47

(a)

TEMPS

TID	Mois	Annee
11	Jan	2010
22	Fev	2010
33	Mar	2010
44	Avr	2010

	Ville		Mois			
RID	Alger	Oran	Jan	Fev	Mar	Avr
1	1	0	1	0	0	0
2	1	0	0	1	0	0
3	1	0	0	0	1	0
4	1	0	1	0	0	0
5	0	1	0	1	0	0
6	0	1	1	0	0	0
7	0	1	0	0	0	1
8	0	1	0	0	1	0
9	1	0	0	1	0	0
10	1	0	1	0	0	0
11	1	0	1	0	0	0
12	0	1	1	0	0	0

IJB_Ville_Mois

(b)

AND
0
0
1
0
0
0
0
0
0
0
0

(c)

FIGURE 7: Exemple d'un index de jointure binaire

ce qui est en généal le cas dans les dimensions d'un entrepôt [14]. C'est ainsi qu'actuellement la majorité des SGBD, tels que Oracle, Sybase et IBM supportent ce type d'index [72]. Notons que l'hypothèse sur la faible cardinalité des attributs indexés ne constitue plus une limite pour ce type d'index. Plusieurs travaux de recherche ont montré leur efficacité même pour les attributs de grande cardinalité en proposant des techniques de compression des index construits [72][73][74][75][76][77]. Des études expérimentales ont prouvé l'efficacité des index compressés même pour des attributs ayant une cardinalité de 200000 valeurs distinctes [78].

3.5 FORMALISATION DU PROBLÈME DE SÉLECTION D'INDEX

Un système de base de données traite un ensemble de requêtes $Q = \{q_1, q_2, \ldots,\}$ soumises respectivement à des moments différents. Notons, que si une requête q_i est traitée par le système à un temps t_i, il est probable qu'elle sera soumise au système à un instant $t_j > t_i$. Ainsi, la sélection des index optimaux pour les requêtes soumises pendant une période de temps est susceptible d'améliorer les performances du système, car ces mêmes requêtes peuvent être soumises dans le "futur". C'est pour cette raison que, dans la pratique, les administrateurs s'intéressent à optimiser le traitement des requêtes les plus fréquentes. Se concentrer sur de telles requêtes conduit à des gains de performance significatifs pour le système [79]. En pratique les requêtes qui sont prise en compte représentent environ seulement 20% de l'ensemble des requêtes actives et réalisent 80% des accès aux données [80]. L'ensemble des requêtes sélectionnées pour le processus d'optimisation est appelé *charge de travail*. Il revient à l'administrateur de réaliser une telle sélection pour différentes périodes du fonctionnement du système [79].

Soit $W = \{q_1^{f_1}, q_2^{f_2}, \ldots, q_n^{f_n}\}$ une charge de n requêtes où chaque requête q_i est caractérisée par sa fréquence d'utilisation f_i. Une première étape très importante pour la procédure de sélection des index est l'identification des attributs pertinents susceptibles d'être indexés. Ces derniers sont appelés attributs indexables [20].

Definition 9 (Attribut indexable). *Les attributs indexables sont ceux présents dans les prédicats de sélection dans les clauses* `Where` *des requêtes. Un prédicat de sélection défini sur un attribut* a_i *d'une table* T *possède la forme suivante :*

$$T.a_i \ \theta \ Valeur$$

Où θ *représente un opérateur de comparaison dans l'ensemble* $\{=, <, >, <=, >=\}$ *et* $Valeur \in Domaine(a_i)$. $Domaine(a_i)$ *étant l'ensemble des valeurs possibles de l'attribut* a_i.

A titre d'exemple, si la clause `WHERE` d'une requête q contient le prédicat $T.a = 10$, un index sur l'attribut a peut être utilisé pour répondre à la requête en récupérant tous les tuples (enregistrements) de T vérifiant la condition spécifiée dans le prédicat.

Soit $A_W = \{a_1, a_2, \ldots, a_k\}$ l'ensemble des attributs indexables pour une charge de requêtes donnée W. Etant donné qu'un index peut être créé sur un ou plusieurs attributs, alors tout sous-ensemble non vide $I \subseteq A_W$ représente un index potentiel. Si $I_W = \{I_1, I_2, \ldots, I_m\}$ designe l'ensemble des tous les index possibles pour la charge W, alors tout sous-ensemble non vide $C \subseteq I_W$ est appelé configuration d'index. Rappelons que pour k attributs indexables, on a exactement $2^k - 1$ index différents conduisant à $2^{2^k-1} - 1$ configurations possibles.

Pour une requête $q \in W$ et une configuration d'index C_i, on note le coût d'exécution de q, exploitant C_i, par $\chi(q, C_i)$. Si $S(I)$ désigne l'espace nécessaire pour stocker l'index I, alors l'espace nécessaire pour stocker C_i est donné par $S(C_i) = \sum_{I \in C_i} S(I)$.

Definition 10 (Problème de sélection d'index). *Étant donnée une charge de requêtes* $W = \{q_1^{f_1}, q_2^{f_2}, \ldots, q_n^{f_n}\}$ *et une limite d'espace de stockage* S_{max}*, l'objectif du problème de sélection d'index est de trouver, parmi toutes les configurations possibles, une configuration* C_{opt} *qui minimise le coût de la charge* W *et respecte la limite* S_{max} *:*

$$\begin{cases} C_{opt} = \underset{C_i}{\operatorname{argmin}} \sum_{q \in W} \chi(q, C_i) * f_i \\ \\ \text{et } \sum_{I \in C_i} S(I) \leqslant S_{max} \end{cases}$$

3.6 ÉTAT DE L'ART

Théoriquement, le problème de sélection d'index peut être résolu par une énumération exhaustive de toutes les solutions (configurations) possibles. Cependant, comme nous l'avons déjà souligné, une telle énumération conduit à des approches de complexité exponentielle inutilisables en pratique. Le problème de la sélection des index est NP-complet [21]. Pour ce type de problèmes, la résolution exacte est très compliquée, au regard du temps d'exécution prohibitif nécessaire, et fournit ainsi une justification théorique pour l'utilisation des méthodes approchées [81]. L'approche naturelle pour résoudre un problème NP-complet est de proposer des approches d'approximation dont l'objectif est de choisir la meilleure solution parmi un ensemble réduit de solutions possibles en procédant à un élagage de l'espace de

recherche initial. Les approches basées sur ce principe cherchent donc le meilleur compromis entre l'exploration de l'espace de recherche et la qualité des résultats obtenus.

Dans la littérature dédiée, différentes approches ont été proposées pour recommander une configuration d'index optimisée. Les approches proposées se composent généralement de deux étapes [20] (1) identification des attributs indéxables pour la charge étudiée et (2) sélection d'une configuration optimisée d'index. Les premiers travaux sur le problème de sélection d'index font appel à l'expertise de l'administrateur pour identifier manuellement l'ensemble des attributs indexables [82][83][84]. Dans les travaux plus récents, la tendance est de recourir à une approche automatique qui s'appuie sur une analyse syntaxique des requêtes pour identifier les attributs indexables [22] [85][86][87].

Nous allons présenter, dans ce qui suit, les principales approches qui ont été proposées pour la résolution du problème de sélection d'index. Notre but est de donner une vue d'ensemble des méthodes proposées, et d'en illustrer la diversité. C'est pourquoi nous les avons regroupé en fonction du type de l'approche utilisée. Tout d'abord les méthodes basées sur des heuristiques gloutonnes. Nous présentons ensuite les approches basées sur des méta-heuristiques. Nous finirons par nous intéresser particulièrement aux méthodes utilisant la fouille de données car nous allons nous-même fonder notre approche de résolution sur les limites de ces dernières.

3.6.1 *Travaux basés sur des heuristiques gloutonnes*

3.6.1.1 *Travaux de K.Y. Whang*

K.Y. Whang [82] a proposé deux approches de sélection d'index, l'une ascendante (ADD) et l'autre descendante (DROP). L'approche ascendante (ADD) commence par initialiser la configuration d'index par l'ensemble vide. Un processus itératif permet de générer une configuration optimisée. À chaque étape, un index susceptible de réduire l'exécution des requêtes est ajouté à la configuration existante. Le processus s'arrête lorsque tous les index sont épuisés ou aucune réduction de coût n'est possible. L'approche descendante (DROP) considère une configuration initiale contenant tous les index possibles. Elle procède ensuite, itérativement, à l'élimination de l'index engendrant la plus grande décroissance du coût d'exécution de la charge de requêtes. Quand l'élimination d'un seul index ne réduit pas le coût d'exécution des requêtes, l'approche élimine deux index à la fois, ensuite trois index, et ainsi de suite jusqu'à ce que ça ne soit plus possible.

3.6.1.2 *Travaux de Frank et al.*

Frank et al. [83] proposent un outil d'aide à la sélection d'une configuration d'index pour une base de données. L'outil proposé utilise, comme entrée, une charge de requêtes et une configuration initiale d'index $X = \{i_1, i_1, \ldots, i_n\}$. En sortie, l'outil recommande un sous-ensemble $X' \subseteq X$ d'index pertinents. En exploitant l'optimiseur du SGBD, l'outil permet, pour chaque requête, d'estimer le gain de performance qu'apporte l'utilisation d'un ensemble d'index. Le gain étant défini par la différence entre le coût d'exécution des requêtes sans et avec les index considérés. Le gain ainsi estimé est représenté sous forme d'un graphe. La méthode proposée peut être résumée comme suit :

1. Une requête de la charge est soumise à l'optimiseur de requêtes avec un ensemble initial d'index.
2. Les index utilisés pour traiter la requête courante ainsi que leurs gains sont stockés.
3. Tant qu'il existe une requête non traitée, aller à l'étape 1.
4. Les gains de chaque index pour l'ensemble des requêtes sont cumulés.
5. Les index présentant un gain total positif sont enfin recommandés par l'outil.

3.6.1.3 *Travaux de Chaudhuri et al.*

Chaudhuri et al. [84][22] ont développé un outil de sélection d'index IST (Index Selection Tool) dans le cadre du projet de recherche *AutoAdmin* lancé par Microsoft [1] pour l'auto-administration d'une base de données. L'outil IST, dont l'architecture générale est représentée à la Figure 8, recommande un ensemble d'index pertinents pour une charge de requêtes donnée. Afin de réaliser les meilleurs choix, IST exploite le coût estimé par l'optimiseur de requêtes du SGBD. Cette tâche est assurée par le module d'*évaluation du coût*. Pour les index non encore matérialisés, IST a besoin de simuler leur présence en faisant appel au module *What-if*. L'approche de sélection procède en plusieurs étapes :

1. **Sélection des index candidats :** Cette phase concerne l'énumération des attributs indexables pour chaque requête dans la charge. Chaque attribut est considéré comme un index mono-attribut. Les configurations construites à partir de ces attributs sont ensuite évaluées en utilisant le module d'*évaluation des coûts* ou le moule *what-if*. La meilleure configuration i.e celle qui génère un coût minimum est retenue. Le but de cette étape est de sélectionner une meilleure configuration pour chaque requête d'une manière indépendante. Enfin, l'union

1. http ://www.research.microsoft.com/dmx/AutoAdmin

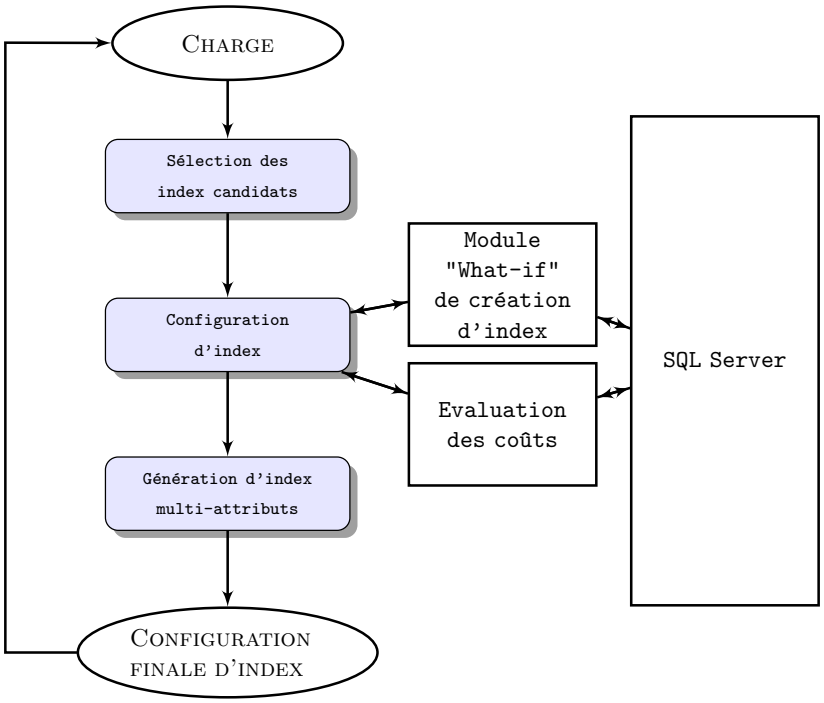


FIGURE 8: Approche de Chaudhuri et al.

des configurations obtenues pour toutes les requêtes représente l'ensemble des index candidats.

2. **Élagage de l'ensemble des index candidats :** L'objectif de cette étape est de garder les index offrant les meilleurs gains de performance. À partir des n index candidats construits dans l'étape précédente, les k meilleurs sont sélectionnés d'une manière gloutonne comme suit :
   a) Identifier la meilleure configuration C_0 de taille $m (m << k)$,
   b) Parmi les index non sélectionnées, ajouter à C_0 celui qui engendre le coût minimal,
   c) Tant que k n'est pas atteint, aller à (b).
3. **Génération des index multi-attributs :** Les meilleurs index sélectionnés dans l'étape précédente sont mono-attribut. Ils sont alors exploités pour construire des index multi-attributs. Pour réaliser cette tâche, l'outil IST utilise deux fonctions : MC-LEAD et MC-ALL. La fonction MC-LEAD génère un index sur deux attributs en combinant un index mono-attribut avec un attribut indexable (non encore matérialisé). La fonction MC-ALL génère un index sur deux attributs en combinant deux index mono-attributs. La génération des index multi-attributs de taille supérieure à 2 se fait selon le même principe.

3.6.1.4 *Travaux de Golfarelli et al.*

Golfarelli et al. [85] ont proposé une approche heuristique de sélection d'index dans les entrepôts de données. L'approche proposée dont l'architecture générale est présentée à la figure 9 permet de recommander un ensemble optimal d'index étant donnée une charge de requêtes et une contrainte d'espace de stockage. Compte tenu

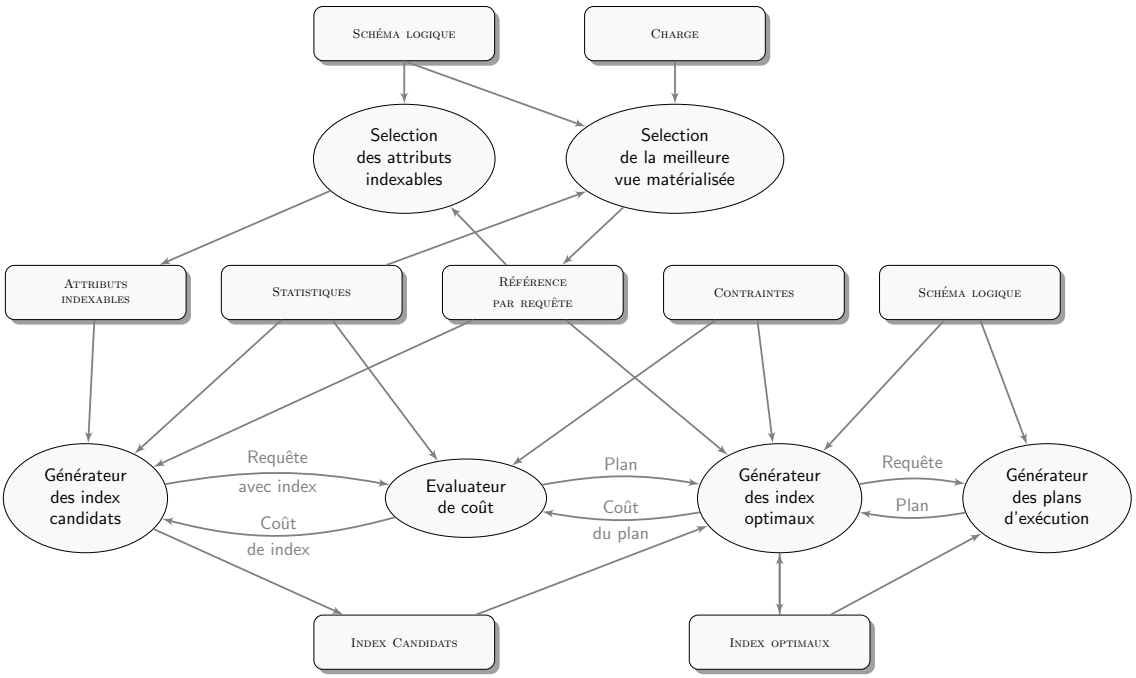


FIGURE 9: Approche de Golfarelli et al.

de la charge en entrée, les auteurs déterminent un ensemble potentiel d'index candidats. Cette tâche est assurée par Le *sélecteur d'attributs indexables*. En se basant sur une analyse des requêtes, il permet de déterminer quels sont les attributs des tables de dimensions qui sont pertinents à indexer. Ces derniers sont exploités par le *générateur d'index candidats* qui évalue quelle est la technique d'indexation qui convient le mieux à un attribut indexable et génère en conséquence l'ensemble des index candidats. Un algorithme glouton est utilisé ensuite pour choisir progressivement, à partir des index candidats, les index les plus bénéfiques tant que la contrainte d'espace est satisfaite. Afin d'évaluer la performance de index manipulés, les auteurs ont défini un modèle d'optimiseur. Ce dernier simulent les différents plans d'exécution d'une requête pour en élire le meilleur. Ce choix est guidé par un modèle de coût proposé par les auteurs. Le coût d'un plan est exprimé en nombre de pages disques à lire pour répondre à une requête donnée.

3.6.1.5 *Travaux de Boukhalfa et al.*

Boukhalfa et al. [20][86][87] ont étudié le problème de sélection des index de jointure binaires IJB dans les entrepôts de données. Un IJB peut être défini sur un ou plusieurs attributs $\{a_1, a_2, , a_n\}$ où chaque attribut $a_i (1 \leqslant i \leqslant n)$ peut appartenir à n'importe quelle table de dimension. Les auteurs ont proposé deux heuristiques pour la sélection d'une configuration optimisée en considérant les deux cas de figure : (1) les index mono-attributs et (2) les index multi-attributs (figure 10). Les ap-

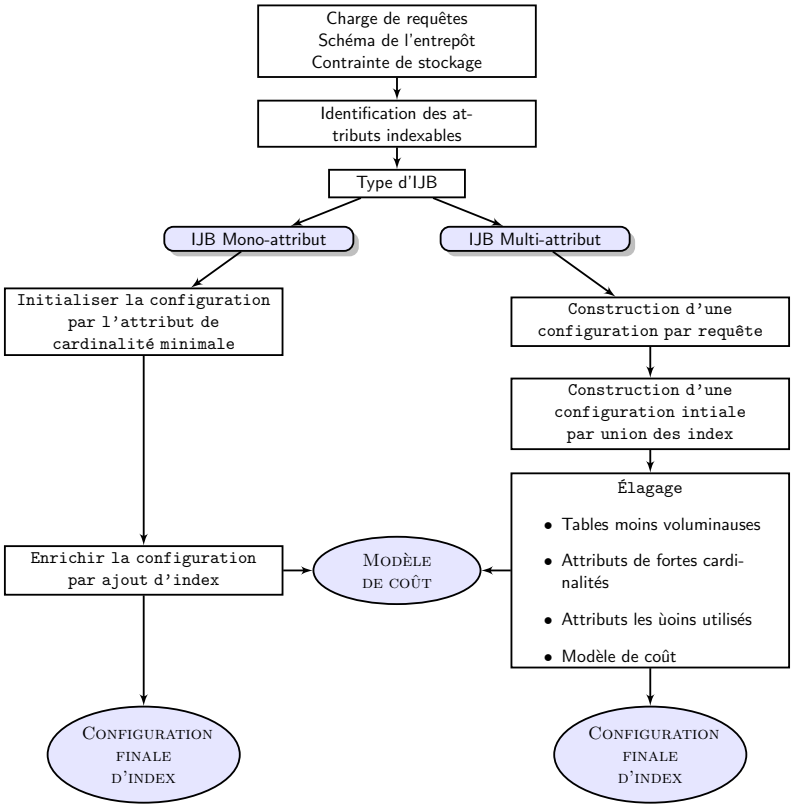


FIGURE 10: Approche de Boukhalfa et al.

proches proposées prennent en entrée une charge de m requêtes les plus fréquentes $\{q_1, q_2, \ldots, q_m\}$ et une limite maximale d'espace disque réservée aux index. Comme résultat, elles recommandent une configuration d'IJB mono et multi-attributs permettant de réduire le coût d'exécution de la charge, en respectant la contrainte d'espace de stockage.

1. **Sélection d'une configuration d'index mono-attributs :** L'approche proposée est composée de trois phases : (1) l'identification des attributs indexables, (2) l'initialisation de la configuration et (3) l'enrichissement de la configuration

courante par l'ajout de nouveaux index. Dans la première étape, l'ensemble des requêtes est analysé afin d'extraire les attributs indexables. Ensuite, une procédure d'élagage est réalisée en évitant les attributs de fortes cardinalités. l'ensemble des attributs indexables retenus, caractérisés par de faible et moyenne cardinalité, constitue les attributs indexables candidats. Les étapes suivantes décrivent une démarche gloutonne pour la sélection d'une configuration d'index. Elle commence par initialiser la configuration par l'index défini sur l'attribut ayant la cardinalité minimale noté IJB_{min}. Cette configuration est enrichie itérativement par l'ajout de l'index engendrant le meilleur gain de performance. La procédure s'arrête lorsqu'aucun gain de performance n'est possible ou que la limite de l'espace de stockage est atteinte.

2. **Sélection d'une configuration d'index multi-attributs :**

   L'approche proposée pour la sélection d'une configuration multi-attributs commence par choisir un index par requête. L'idée est de procéder à l'optimisation de chaque requête d'une manière indépendante. La configuration recherchée est ainsi initialisée par l'union de tous les index préalablement choisis. Les auteurs estiment que, même si la configuration ainsi formée permet de satisfaire toutes les requêtes, elle risque d'être très volumineuse et de violer la contrainte de stockage. Dans ce cas, une procédure d'élagage est nécessaire pour réduire la taille de la configuration. L'approche proposée est composée des étapes suivantes :

   a) **Identification des attributs indexables :** l'analyse des requêtes de la charge permet d'identifier, pour chaque requête, l'ensemble des attributs indexables. Cet ensemble est élagué en ne retenant que les attributs de faible et moyenne cardinalité. L'ensemble des attributs indexables est l'union de tous les attributs indexables identifiés pour chaque requête.

   b) **Construction d'une configuration par requête :** une configuration est associée à chaque requête de la charge. Elle est composée de l'index IJB définit sur tous les attributs indexables de la requête.

   c) **Construction d'une configuration initiale :** Les index définis dans l'étape précédente permettent d'optimiser séparément les requêtes de la charge. Le but de cette étape est de construire une configuration initiale qui permet d'optimiser toutes les requêtes de la charge. Cette configuration est formée, naturellement, de l'union des index définis dans l'étape précédente : $C_0 = \cup_{i=1}^{n} \mathrm{IJB}_i$.

   d) **Construction d'une configuration finale :** La configuration initiale obtenue dans l'étape précédente est bénéfique pour la totalité des requêtes, car

chaque index a été défini pour optimiser une requête séparément. Toutefois, le nombre d'index ainsi généré est d'autant plus important que la charge en entrée est volumineuse. La création de tous ces index peut ne pas être réalisable en pratique à cause de la taille de l'espace de stockage alloué aux index. Dans une telle situation, les auteurs proposent différentes stratégies d'élagage :

i. **Élimination des attributs de forte cardinalité (AFC) :** La taille d'un IJB est proportionnelle à la cardinalité des attributs qui le composent. Afin de réduire la taille de la configuration générée, cette stratégie élague itérativement les attributs de forte cardinalité jusqu'à ce que la taille de la configuration devienne inférieure à la limite de l'espace de stockage.

ii. **Élimination des attributs appartenant aux tables moins volumineuses (TMV) :** L'intuition derrière cette stratégie est qu'il n'est pas intéressant de créer un index sur des tables de petites tailles. En effet, le coût des jointures entre la table des faits et des tables de dimension volumineuses est plus important par rapport aux jointures impliquant des tables de dimension de faible taille.

iii. **Élimination des attributs les moins utilisés (AMU) :** Cette stratégie favorise l'indexation des attributs les plus utilisés. L'élagage procède ainsi à l'élimination des attributs les moins utilisés par les requêtes.

iv. **Élimination des attributs apportant le moins de réduction de coût(MC) :** Dans cette stratégie, l'élimination d'un attribut est guidée par un modèle de coût permettant de réaliser une évaluation des configurations intermédiaires issues de chaque élimination. Pour k attributs, les k éliminations sont évaluées afin d'éliminer l'attribut induisant le minimum de gain. Un élagage itératif est ainsi réalisé jusqu'à satisfaction de la contrainte de l'espace de stockage.

3.6.2 *Travaux basés sur des métaheuristiques*

3.6.2.1 *Travaux de Kratica et al.*

Kratica et al. [88] ont proposé un algorithme génétique pour résoudre le problème de sélection d'index. La méthode proposée fait évoluer un ensemble de solutions appelé "population" (configurations dans notre contexte) et permet de s'assurer que les individus (ici index) performants seront conservés, alors que les individus peu adaptés seront progressivement éliminés de la population. La population de départ

est formée d'un ensemble d'index candidats. La fonction objectif à optimiser est le coût de la charge en présence d'une configuration d'index. Ceci définit l'ensemble des individus élus pour survivre à la prochaine génération. La construction combinatoire des configurations d'index est réalisée à l'aide des opérateurs génétiques de croisement, de mutation et de sélection. Pour chaque requête q , la configuration C_q apportant un gain maximal est recherchée. Ensuite, la somme des gains apportés par une configuration pour l'ensemble de toutes les requêtes est calculée. Si cette valeur est négative, la construction des index n'apporte aucun bénéfice. A chaque génération d'une nouvelle population, les individus (index) multiples sont éliminés de la population. Pour combiner deux individus, l'opérateur de croisement uniforme, décrit dans [89], est utilisé. Pour un taux de mutation donné, le nombre de gènes en mutation dans une génération est prédit en utilisant le théorème central limite pour une distribution gaussienne [63]. La procédure de mutation est réalisée uniquement sur le nombre précédent de gènes choisis aléatoirement. Le taux de mutation change d'une génération à une autre suivant une loi proposée par les auteurs. La méthode de sélection utilisée est la *fine grained tournament selection* proposée dans [90].

3.6.2.2 *Travaux de Valentin et al.*

Valentin et al. [91] ont proposé un système pour recommander un ensemble d'index. Ce système admet en entrée une charge de requêtes et renvoie en sortie une configuration d'index qui optimise les requêtes de la charge. Les index candidats, appelés index virtuels dans ce système, sont extraits des requêtes de la charge en effectuant une analyse syntaxique. En présence de ces index et des statistiques de la base de données, l'optimiseur génère le plan d'exécution optimal d'une requête donnée. Si ce plan contient un ou plusieurs index, ces derniers sont recommandés. La génération du plan optimal et la recommandation d'index sont réitérées pour chaque requête de la charge. L'ensemble des index ainsi recommandé est ensuite modélisé comme le problème du sac à dos. Formellement, le problème de sac à dos est posé comme suit : Soit un ensemble d'objets $O = \{o_1, o_2, \ldots, o_n\}$ oú chaque objet $o_i (1 \leqslant i \leqslant n)$ est caractérisé par sa taille $t(o_i)$ et son bénéfice $b(o_i)$. Le sac à dos est de taille T.

L'objectif est de trouver un sous-ensemble d'objets $O_{opt} \subseteq O$ ayant un bénéfice maximum et dont la taille ne dépasse pas T. Par analogie, le problème de sélection d'index est assimilé au problème de sac à dos tel que :

- un index représente un objet,
- le bénéfice d'un index représente le gain apporté par l'index,
- la taille d'un index représente l'espace disque nécessaire pour stocker l'index en question,

– la taille du sac à dos correspond à l'espace disque alloué par l'administrateur pour stocker l'ensemble des index sélectionnés

3.6.2.3 *Travaux de Drias et al.*

Drias et al. [92] ont proposé une approche, dite ACS-BJIS, pour la recommandation d'un ensemble d'index de jointures binaires guidée par les colonies de fourmis. Les règles de mise à jour du phéromone sont conçues de telle manière à affecter plus de phéromone aux éléments (attributs) qui devraient être choisis par les fourmis. L'approche proposée fonctionne selon les étapes suivantes :

1. Initialisation du phéromone
2. Pour chaque fourmi de la génération actuelle
   a) Construire une solution en choisissant les index selon les règles de transition et de mise à jour.
   b) Identifier la meilleure solution de cette génération.
3. Si le critère d'arrêt est atteint alors `Fin`, sinon allez à `2`

L'entrée de l'approche proposée correspond aux informations relatives aux tailles des tables de l'entrepôt de données ainsi que tous les attributs indexables extraits des clauses `where` de la charge de travail des requêtes. Le nombre des index retenus à la fin est limité par un seuil de l'espace de stockage.

Une structure de données dynamique implémentant une file d'attente FIFO (First In First Out) est utilisé pour sauvegarder les index les plus intéressants identifiés par la colonie. L'index le plus pertinent est mis à la tête de la file d'attente alors que le moins pertinent apparaît à la fin de celle-ci. La taille de la file d'attente dépend de la limite de stockage réservée aux index. Lorsque la file d'attente est pleine, l'index en cours de traitement est comparé à celui ayant la moindre performance (situé donc à l'extrémité de la file). Si l'index traité est de meilleure qualité, alors il est inséré à la position appropriée dans la file. Celui de l'extrémité est tout simplement supprimé.

3.6.3 *Travaux basés sur la fouille de données*

3.6.3.1 *Travaux de Aouiche et al.*

Aouiche et al. [63][93][94] ont étudié le problème de sélection des IJB dans le contexte des entrepôts de données modélisés par un schéma en étoile. Au meilleur de nos connaissance, il s'agit du premier travail qui s'est intéressé spécifiquement à ce type d'index. L'approche proposée se base sur une technique de fouille de données et plus spécifiquement la recherche des motifs fréquents fermés. L'intuition derrière le travail proposé est que la pertinence d'un index est fortement liée

avec sa fréquence d'utilisation dans l'ensemble des requêtes d'une charge. Ainsi, la recherche des motifs fréquents semble une solution naturelle au problème étudié. L'approche de sélection proposée par les auteurs est composée des étapes illustrées

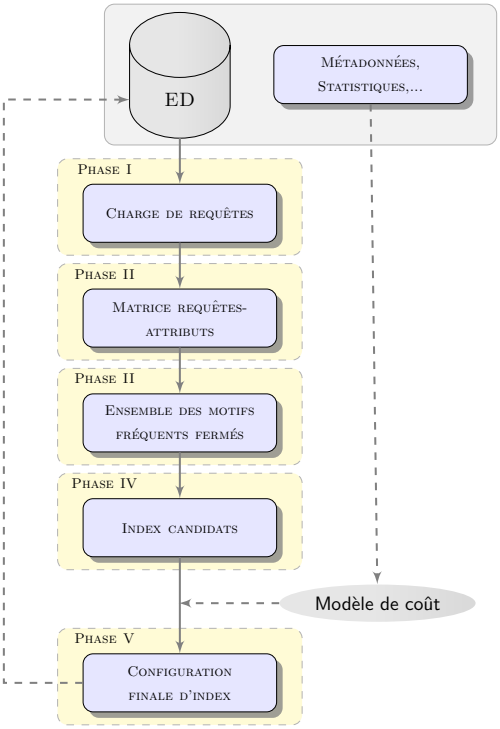


FIGURE 11: Approche de Aouiche et al.

par la figure 11

1. **Extraction de la charge de requêtes :** la charge de requêtes est extraite à partir du journal des transactions sauvegardé par le système durant une période de temps. Cette période est fixée par l'administrateur.
2. **Extraction des attributs indexables :** Les requêtes présentes dans la charge sont traitées par un analyseur syntaxique afin d'en extraire tous les attributs inexables. Ces attributs sont ceux qui font l'objet de prédicats de sélection dans les clauses WHERE des requêtes
3. **Construction d'un contexte de recherche des motifs fréquents :** le contexte d'extraction est représenté par une matrice requête-attributs construite à partir des requêtes et des attributs indexables. Les lignes dans cette matrice représentent les requêtes et les colonnes les attributs indexables utilisés par chaque requête. La j^{eme} case d'une ligne i dans cette matrice est mise à 1 si la requête q_i utilise l'attribut indexable a_j. Elle est mise à 0 sinon.

4. **Application de l'algorithme CLOSE et construction de l'ensemble des index candidats :** parmi les algorithmes de recherche des motifs fréquents fermés, les auteurs utilisent l'algorithme Close. Ce dernier exploite le contexte d'extraction construit dans l'étape précédente et donne lieu à un ensemble de motifs fréquents fermés
5. **Construction de l'ensemble des index candidats :** L'ensemble des index candidats est construit à partir des motifs fréquents fermés résultant de l'application de l'algorithme CLOSE. Le but de cette étape est de vérifier si les attributs contenus dans chaque motif fréquent correspondent bien à un index de jointure binaire.
6. **Construction de la configuration finale d'index :** Les auteurs proposent un algorithme glouton pour choisir une configuration finale d'index. Itérativement, l'algorithme proposé choisit les index optimaux. Afin d'évaluer la fonction objective, les auteurs propose deux modèles de coût. Le premier permet d'estimer l'espace nécessaire pour stocker les index sélectionnés. Le deuxième, quand à lui, calcule le coût d'exécution des requêtes en présence de ces index.

3.6.3.2 *Travaux de Bellatreche et al.*

Bellatreche et al. [95][96] exploitent la même technique de recherche des motifs fréquents fermés en y intégrant d'autre paramètre dans le processus de recherche. Les auteurs remarquent que la recherche basée sur la fréquence des motifs peut ignorer des index sur des attributs non fréquemment utilisés mais qui appartiennent à des tables volumineuses, ce qui ne permet pas d'optimiser une opération de jointure. Les auteurs proposent une adaptation de l'approche proposée par Aouiche et al. [93]. Le processus de sélection repose désormais sur une fonction fitness permettant de pénaliser les index définis sur des tables de dimensions de petites tailles. Pour un motif fréquent m_i la fonction fitness est définie comme suit : $fintess(m_i) = \frac{1}{n}(\sum_{j=1}^{n} \alpha_j \times sup_j)$ oú n représente le nombre des attributs non clés a_j dans m_i, sup_j désigne le support de a_j et α_j est un paramètre définit par : $\alpha_j = \frac{|D_j|}{|F|}$ oú $|D_j|$ et $|F|$ désignent respectivement le nombre de pages nécessaires pour stocker la table de dimension D_j et la table des faits F. L'ensemble des motifs fréquents fermés générés est ainsi élagué par la fonction fitness. Le résultat de la procédure d'élagage constitue l'ensemble des index candidats. Une approche gloutonne est ensuite adoptée pour recommander une configuration d'index. Elle commence par choisir l'index défini sur l'attribut ayant la cardinalité minimale. Itérativement, d'autres attributs sont sélectionnés jusqu'à atteindre la limite de l'espace de stockage.

3.7 SYNTHÈSE DU CHAPITRE

À l'issue de ce travail bibliographique, nous dressons une synthèse des travaux que nous avons présentées dans ce chapitre. Le tableau 7 recense ces travaux classés en fonction du contexte d'utilisation, du type d'index, du modèle de coût, et de la technique de résolution utilisée.

Les différents travaux relatifs au problème de la sélection d'index, exposés dans ce chapitre, nous montrent la variété des approches proposées, mais aussi la difficulté du problème étudié. Les premiers travaux proposés sont basés particulièrement sur des heuristiques gloutonnes. L'idée est de rechercher une solution de bonne qualité en modifiant itérativement les bonnes solutions trouvées. Le traitement de la charge se fait généralement d'une manière séquentielle requête par requête. Un des inconvénients majeurs d'un tel traitement réside dans le fait que si l'espace disque disponible est limité, alors cette stratégie risque d'ignorer des index très pertinents. Une autre limitation de ces approches est le fait de faire appel à l'optimiseur pour estimer le coût des requêtes ce qui va, d'une part, rendre la sélection d'index dépendante d'un SGBD donné. D'autre part, c'est une opération coûteuse pour l'optimiseur car il doit générer et tester plusieurs plans d'exécution des requêtes qui est lui-même un problème d'optimisation complexe [97].

D'autres approches, basées sur des métaheuristiques, ont été également proposées. Malgré le succès théoriques des métahueristiques, elles souffrent, en pratique, du problème de calibrage des paramètres, étape importante avant leur utilisation. Les métaheuristiques ont fait leurs preuves en théorie mais ne sont pas toujours accessibles et faciles à comprendre pour un non-spécialiste. L'ajustement, par exemple, d'un algorithme génétique est très délicat. Un administrateur doit comprendre la signification de chaque paramètre (sélection, croisement, mutation) pour espérer fixer des valeurs convenables.

Une autre direction qui a été récemment poursuivie consiste en l'exploitation des techniques de fouille de données et plus particulièrement la recherche des motifs fréquents fermés. Ces travaux font figure de référence dans la mesure oú ils s'appuient sur un socle théorique solide et ne nécessitent pas beaucoup de paramètres. Le seul paramètre à fixer est le support exprimant le degré de corrélation des attributs souhaité par l'administrateur. Cependant, et d'un point de vue théorique, les propriétés des motifs fréquents fermés ne justifient pas leur utilisation pour la résolution du problème de sélection d'index. Ceci est dû en particulier au nombre important des index générés ainsi que leur redondance. De plus, les études proposées sont basées sur une simplification assez réductrice du contexte d'extraction des index. Le contexte considéré représente les requêtes équitablement comme si elles ont toutes

la même fréquence. Or c'est en tenant compte des fréquences des requêtes que les index extraits peuvent être les plus intéressants. Nous avons identifié trois points principaux qui sont susceptibles d'être améliorés dans ces approches. Ce constat a motivé la proposition d'une nouvelle approche qui sera présentée au chapitre suivant.

Travaux	Contexte		Type d'index		Approche			Modèle de coût	
	BDD	EDD	Mono attribut	Multi attribut	Heuristique	Méta heuristique	FDD	Mathématique	Optimiseur
K.Y Whang [82]	x		x		x			x	
Frank et al. [83]	x		x		x				x
Chaudhuri et al. [84][22]	x		x	x	x				x
Golfarelli et al. [85]		x	x	x	x				x
Boukhalfa et al. [20][86]		x	x	x	x			x	
Kratica et al. [88]	x		x	x		x		x	
Valentin et al. [91]	x		x	x		x			x
Drias et al. [92]		x	x	x		x			x
Aouiche et al. [63][93]		x	x	x			x	x	
Bellatreche et al. [95][96]		x	x	x			x	x	

Table 7: Synthèse des principaux travaux sur le problème de sélection d'index

4

EXTRACTION DES MOTIFS FRÉQUENTS MAXIMAUX POUR LA SÉLECTION DES IJB

Sommaire

4.1 INTRODUCTION

Les requêtes définies sur les entrepôts de données modélisés selon un schéma en étoile, dites requêtes de jointure en étoile, nécessitent souvent plusieurs jointures entre la table de faits et les tables de dimensions. La jointure étant l'opération la plus coûteuse dans l'algèbre relationnelle, le coût de telles requêtes est prohibitif [98]. Les techniques proposées pour réduire le temps de calcul des jointures dans les bases de données, comme les jointures par hachage [98] ne sont efficaces que quand la jointure s'applique à deux tables et que le volume de données est relativement faible [63]. Dans le contexte des entrepôts de données, vu le volume important des

données manipulées, les index de jointure binaire ont été proposés pour un traitement efficace de ces jointures. En effet, les jointures sont préalablement calculées au moment de la création de ces index. Elles ne sont donc pas calculées lors de l'exécution des requêtes [63]. En plus, la structure binaire de ces index facilite l'exécution des opérations courantes AND, OR, Not ou Count qui opèrent directement sur les index (donc en mémoire) et non pas sur les données sources [63]. D'autre part, l'espace disque occupé par les bitmaps est faible relativement à d'autre type d'index tels que les B-arbre [99], C'est ainsi que des travaux récents se sont particulièrement intéressés au problème de la sélection de ce type d'index [86][87][93][94].

Ce chapitre présente une nouvelle approche pour la sélection des index de jointure binaires. Basée sur le concept des motifs fréquents maximaux, l'approche proposée permet de recommander un ensemble réduit d'index qui soient représentatifs et non redondants. Les motifs fréquents maximaux présentent des propriétés théoriques intéressantes qui n'ont pas été exploités par les approches précédentes. Nous montrons la consistance de l'approche proposée, tant d'un point de vue formel qu'expérimental. Pour faciliter la lecture nous appellerons index fermés (maximaux) les index générés en utilisant les motifs fréquents fermés (maximaux).

4.2 CRITÈRES DE L'EFFICACITÉ D'UNE APPROCHE DE SÉLECTION D'INDEX

Les approches proposées pour la résolution du problème de sélection d'index sont destinées à faciliter la tâche d'un administrateur et a dégager le plus rapidement possible des informations utiles afin d'orienter l'administrateur vers les bons choix de sélection et de prise de décision. Dans la plupart des travaux sur le problème de sélection des index, le seul critère qui est considéré important pour évaluer une approche de résolution est le gain réalisé par la configuration finale recommandée, sur le coût total de traitement de la charge. Nous sommes convaincus que si la qualité d'une solution repose pour beaucoup sur le gain de coût réalisé, elle dépend également de la façon dont elle est identifiée. Dans cette section, nous voulons souligner que la qualité d'une approche de résolution doit tenir compte de trois critères que nous considérons comme indicateurs de l'efficacité d'une approche de sélection d'une configuration optimisée d'index.

1. **La stratégie d'élagage de l'espace de recherche initial :** L'élagage de l'espace de recherche doit être le plus efficace possible tout en préservant la production de résultats pertinents et utiles à l'administrateur. Plus le nombre des index manipulés est élevé, plus il y a de traitements pour vérifier leur utilité et plus il est difficile de trouver rapidement une solution. Afin d'éviter la prolifération des index inutiles, il est judicieux de générer le moins d'index pertinents possible.

Cela permet d'une part, de réduire la complexité du problème de sélection d'index et d'autre part, de minimiser l'espace nécessaire pour le stockage des index retenus.

2. **La redondance des index générés :** Un deuxième problème, souvent négligé, mais qui constitue un des obstacles majeurs à l'obtention de bonnes performances réside dans la redondance des index générés. La non prise en compte des dépendances d'inclusion entre les index crée des problèmes d'efficacité. Dans un ouvrage de référence dans le domaine [100], l'auteur confirme que les index définis sur des attributs appartenant à un autre index plus large, apportent rarement un gain supplémentaire par rapport à l'index initial et occupent, de plus, inutilement un espace additionnel. La propriété majeure qui devra être maintenue pour préserver la pertinence des index est relative à la non-redondance, qui doit incontestablement préserver les index les plus avantageux.
3. **Le nombre des solutions considérées :** Cette orientation de recommandation d'une configuration pertinente est principalement liée à une observation pratique relative au comportement réel des administrateurs. En pratique, un administrateur peut opter pour une configuration qui permet de gagner énormément en espace de stockage au prix d'une faible perte, ou une perte raisonnable, du coût de performance. Nous sommes convaincus que le problème étudié n'a pas une seule solution optimisée, mais potentiellement plusieurs, en fonction de la vision de l'administrateur au compromis coût/espace car ces deux paramètres sont généralement divergeants. Au lieu de donner l'avantage à la configuration conduisant au coût minimal, comme c'est le cas dans les travaux antérieurs, nous pensons qu'il est plus approprié d'avoir un critère pour choisir parmi plusieurs configurations qui se comportent raisonnablement bien. Cette remarque nous offre la possibilité d'explorer plusieurs solutions et ainsi d'améliorer la recherche d'une bonne solution.

4.3 NOTRE APPROCHE

4.3.1 *Motivation*

Intuitivement dans une stratégie de recommandation d'une configuration pertinente d'index, le rôle de la co-occurrence des attributs indexables est essentiel : un index, définit sur un sous-ensemble des attributs indexables ne sera pas suggéré si la fréquence de ses apparitions n'atteint pas un certain seuil. Cette évidence reflète la nature même du problème de sélection d'index qui s'appuie sur les requêtes les

plus fréquentes. Observons également que les requêtes d'une charge donnée peuvent exploiter des index en commun selon qu'elles partagent un sous-ensemble, ou idéalement, l'ensemble des attributs indexables. Le problème de sélection d'index présente ainsi une analogie avec celui de la recherche des motifs fréquents : les requêtes jouent le rôle des transactions à analyser et les attributs indexables sont les items décrivant les transactions. Cette analogie n'est pas fortuite. Le concept des motifs fréquents offre dans un premier temps, un support mathématique solide pour la modélisation du problème de sélection d'index. Ensuite, en termes de résolution, nous pouvons profiter des progrès en matière d'extraction de motifs, et notamment les représentations condensées de ces derniers. De tels progrès permettent une exploration plus efficace de l'espace de recherche.

Exemple 10. *Créer le contexte d'extraction consiste à doter chaque requête de l'ensemble des attributs indexables qu'elle référence. Ainsi dans le contexte illustré par le tableau 8, la requête* q_3 *est décrite par les attributs* a *et* d*. Le contexte d'extraction ainsi formé décrit les éventuelles corrélations (identifiées sous forme de motifs) entre les différentes requêtes considérées. Intuitivement, deux ou plusieurs requêtes sont dépendantes d'un même index si elles sont décrites par le même ensemble d'attributs indexables. Elles sont toutefois indépendantes au sens où il n'existe pas d'attributs communs dans leurs déscriptions.*

Requêtes (Transactions)	Attributs indexables (Items)
q_1	a b c
q_2	a b e
q_3	a d
q_4	a c
q_5	b c
q_6	b d c

TABLE 8: Exemple de requêtes et le contexte d'extraction corréspondant

Il est important de noter aussi que cette formulation fait abstraction du type d'index qu'il soit mono ou multi-attributs. La pertinence d'un index sélectionné dépend de la corrélation réelle de ses attributs et non de son type. Ainsi, la configuration finale recommandée peut contenir à la fois des index mono-attributs et des index multi-attributs. Cette formulation présente un double avantage. D'une part nous ne

nous préoccupons pas du type d'index et donc nous n'avons pas besoin de proposer deux approches distinctes selon le type considéré. D'autre part, il n'est pas nécessaire de faire appel à un processus itératif pour générer les index multi-attributs. Ces deux aspects sont pris en charge par les techniques de recherche des motifs fréquents.

L'adaptation des motifs fréquents pour le problème de sélection d'index ne semble pas suffisante car elle engendre une question évidente mais importante : quel type de motifs faut-il extraire ? Faut-il extraire tous les motifs ?, Si oui à quel prix ? En d'autres termes quels type de motifs est plus adapté au problème étudié ? Le recours à la recherche des motifs fréquents nécessite une appréhension préalable des différentes représentations condensées permettant de mettre en place une stratégie de fouille adaptée. L'objectif, ainsi qu'il est exposé dans [101], étant d'éviter des résultats non pertinents liés à une mauvaise appréciation du contexte d'utilisation des techniques de fouille de données.

Nous sommes convaincus que le choix du type des motifs à exploiter, pour résoudre quelconque problème, est une tâche primordiale et fortement dépendante du problème étudié et de la sortie que l'on souhaite obtenir. En effet, ce choix définit systématiquement le nombre et la redondance des motifs engendrés. Ce qui a un impact majeur, d'une part sur la pertinence des motifs extraits et, d'autre part, sur la complexité de la procédure d'extraction elle même. Au delà des difficultés de la modélisation, nous étions notamment confrontés à ce problème lors de l'adaptation de la technique de la recherche des motifs fréquents au problème de la sélection d'index. Il est, en effet, nécessaire de s'assurer d'une transposition adéquate du problème dans un modèle qui se prête mieux à sa résolution.

Paradoxalement, alors que la représentation condensée via les motifs fréquents maximaux semble particulièrement adaptée au problème de sélection d'index, les travaux antérieurs [86][87][93][94] ont utilisé les motifs fréquents fermés. D'un point de vue théorique, les propriétés des motifs fréquents fermés ne justifient pas leur utilisation pour la résolution du problème de sélection d'index. Pour qu'une modélisation via les motifs fréquents soit efficace, ses résultats doivent satisfaire certains critères de qualité entre autres la concision et la non-redondance [102]. Nous avons identifié deux limites qui illustrent le fait que ce type de motifs n'est pas forcement bien adapté au problème de sélection d'index :

1. D'une part le nombre important des index manipulés. En effet, le nombre des index extraits peut être si grand qu'il affaiblit l'intérêt pratique de l'élagage effectué. Plus le nombre des index manipulés est grand, moins est l'efficacité de l'approche. Ceci est évidemment plus remarquable dans le cas des charges de grande taille et/ou présentant de fortes corrélations entre les attributs. L'utilisation des motifs fréquents fermés dans pareils contextes réduit leurs perfor-

mances puisque l'espace de recherche des motifs fermés tend à se superposer avec celui des motifs fréquents [46] (voir l'exemple 3).

2. D'autre part, la sélection se fait au détriment de l'intérêt des index retenus, puisque la plupart des index ainsi choisis sont redondants et donc moins intéressants. Il se peut en effet que les index générés soient certes intéressants mais se recoupent. Par exemple deux index qui ne diffèrent que par la présence ou l'absence d'un attribut peuvent être considérés comme redondants. Comme nous l'avons déjà noté, cette redondance est un phénomène néfaste pour la pertinence des configurations générées. Les index redondants apportent rarement un gain supplémentaire par rapport à l'index initial et occupent, de plus, inutilement un espace additionnel. L'élagage ainsi réalisé risque de nécessiter lui même un deuxième élagage pour remédier à ce phénomène.

Les propriétés intéressantes des motifs fréquents maximaux permettent de résoudre simultanément ces deux problèmes. Motivé par ce constat, nous proposons une approche qui représente une alternative plus efficace. Nous visons à éviter les limites observées afin de réaliser le meilleur élagage possible tout en préservant la production de résultats pertinents et utiles à l'administrateur. De plus, extraire les index maximaux définit une contrainte supplémentaire sur les résultats souhaités. Or il est bien connu que, dans le domaine de fouille des motifs, l'utilisation des contraintes permet de réduire l'espace de recherche, ce qui contribue de manière significative à atteindre de meilleurs niveaux de performance et de passage à l'échelle [103][104]. Dans ce qui suit, nous allons démontrer formellement, puis expérimentalement, les avantages de nos propositions.

4.3.2 *Un espace de stokage réduit*

Notre approche est basée sur l'intuition que l'utilisation des index maximaux permet un élagage plus efficace de l'espace de recherche, ce qui conduit à un gain substantiel en terme d'espace de stockage des index générés. Rappelons qu'un index de jointure binaire est construit sur la table des faits en utilisant un ou plusieurs attributs indexables. Ainsi, l'index formé contient autant de lignes que le nombre d'enregistrements dans la table des faits. La taille d'un index I^{a}_{ijb} définit sur un attribut a est donc $\frac{|a|\times||F||}{8}$ octets [70] oú $|a|$ représente la cardinalité de l'attribut a et $||F||$ désigne le nombre d'enregistrements dans la table des faits F. Soit

$I_{ijb}^{a_1,\ldots,a_k} = \{a_1, a_2, \ldots, a_k\}$ un index construit sur k attributs. L'espace occupé par $I_{ijb}^{a_1,\ldots,a_k}$, noté $size(I_{ijb}^{a_1,\ldots,a_k})$ est :

$$size(I_{ijb}^{a_1,\ldots,a_k}) = \left(\sum_{i=1}^{k} \frac{|a_i|}{8}\right) \times \|F\|$$

Si $C_{ijb} = \{I_1, I_2, \ldots, I_m\}$ désigne une configuration composée de m index et $Size(C_{ijb})$ représente l'espace disque nécessaire pour stocker C_{ijb}, alors on a :

$$Size(C_{ijb}) = \sum_{j=1}^{m} size(I_j)$$

.

Proposition 1. *Si C_{Close} et C_{Max} désignent respectivement l'ensemble des index fermés et maximaux générés pour une charge de requêtes donnée $W = \{q_1^{f_1}, q_2^{f_2}, \ldots, q_n^{f_n}\}$, alors on a : $Size(C_{Close}) \geqslant Size(C_{Max})$.*

Démonstration. Ce résultat est une conséquence directe des propriétés des motifs fréquents fermés et maximaux. Plus précisément, soient $I_i \in C_{Close}$ et $I_j \in C_{Close}$ deux index potentiellement utiles pour une requête q.

- Si $(I_i \not\subset I_j \wedge I_j \not\subset I_i)$ alors, aucun index généré n'est inclus dans un autre index. C'est la définition même des motifs fréquents maximaux et on a alors $Size(C_{Close}) = Size(C_{Max})$. C'est souvent le cas pour les valeurs élevées du support minimum.
- Sinon, supposons que $I_i \subset I_j$. Par exemple,

$$I_i = \{a_1, \ldots, a_k\} \text{ et } I_j = \{a_1, \ldots, a_k, a_{k+1}, \ldots, a_s\}$$

  - Si la requête q peut utiliser l'index I_j, alors l'index I_i peut être éliminé de la configuration. On ne garde ainsi que les index non inclus dans d'autre index. On se retrouve dans le cas précédent et nous avons : $Size(C_{Close}) = Size(C_{Max})$.
  - Sinon, certains index sont dupliqués et nous avons forcément : $Size(C_{Close}) > Size(C_{Max})$.

□

4.3.3 *Un contexte d'extraction enrichi*

Il n'y a aucun doute que les fréquences des requêtes jouent un rôle important quant à la sélection d'un ensemble d'index pertinents. La prise en compte de ce paramètre a un impact majeur sur la qualité des index recommandés. Les travaux

antérieurs ont assumé un contexte d'extraction simple oú chaque requête est représentée une seule fois par les attributs qu'elle référence. Ceci est loin d'être réaliste car il implique simplement que les requêtes ont la même importance d'utilisation. Or, chaque requête est dotée de sa fréquence reflétant la probabilité de son utilisation et donc son importance. La non prise en considération de ce paramètre implique forcément une sélection biaisée des index retenus. Elle peut conduire à la recommandation d'index relatifs à des requêtes moins fréquentes et l'élimination par la même voie, d'autres index pertinents pour les requêtes les plus fréquentes. Pour remédier à ce problème, nous utilisons un contexte d'extraction enrichi qui représente une extension du contexte précédent. Chaque ligne i correspondant à la requête q_i est dupliquée f_i fois. Ceci va garantir l'utilisation des données fiables et pouvoir ainsi réaliser une fouille de qualité. En effet, la qualité des index extraits va de pair avec la qualité des données soumises au traitement. Le nouveau contexte d'extraction contient alors $\sum_{i=1}^{n} f_i$ lignes (n désigne le nombre des requêtes de la charge). Le nombre de colonnes dans chaque ligne dépend du nombre d'attributs référencés par la requête en question. Pour $f_i = 1 (1 \leqslant i \leqslant n)$, on retrouve le cas simple utilisé dans les travaux antérieurs. L'utilisation du contexte enrichi garantit la sélection des index relatifs aux requêtes les plus fréquentes. Ceci va certainement conduire à recommander des configurations plus pertinentes pour la charge considérée.

Proposition 2. *Soint $\mathbb{C}_W$ et $\mathbb{C}_{WF}$ les deux contextes d'extraction sans et avec les fréquences des requêtes respectivement. Un attribut indexable a qui est fréquent (infréquent) dans le contexte $\mathbb{C}_W$ peut devenir infréquent (fréquent) dans le contexte $\mathbb{C}_{WF}$.*

Démonstration. En utilisant le contexte $\mathbb{C}_W$, le nombre des transactions est égal à n. Notons Q_a l'ensemble des requêtes pour lesquelles a est un attribut indexable. nous avons $\mathrm{Support}(a) = \frac{|Q_a|}{n}$. Supposons que a est fréquent pour une valeur donnée λ du support minimum i.e $\mathrm{Support}(a) \geqslant \lambda$. En exploitant le contexte $\mathbb{C}_{WF}$ le nombre des transactions devient $N = \sum_{i=1}^{n} f_i \geqslant n$ et $\mathrm{Support}(a) = \frac{|Q_a|+l}{N} (l \geqslant 0)$. Cette valeur est inférieure à λ i.e a devient infréquent si $l < N * \lambda - |Q_a|$. De la même manière, un attribut indexable a qui est infréquent, peut devenir fréquent si $l \geqslant N * \lambda - |Q_a|$. □

Exemple 11. *Pour illustrer la proposition précédente, considérons l'exemple suivant. Soient une charge de 6 requêtes avec les fréquences $1, 3, 5, 1, 2, 3$ respectivement et l'ensemble des attributs indexables a, b, c, d, e. La Figure 12 illustre les caractéristiques des requêtes et les contextes d'extraction correspondants $\mathbb{C}_W$ et $\mathbb{C}_{WF}$. Supposons que $\lambda = 0.5$ (50%). Il est clair que l'attribut c est fréquent par rapport au contexte $\mathbb{C}_W$. Son support est $\mathrm{Support}(c) = \frac{4}{6} = 0.66$. En utilisant le contexte $\mathbb{C}_{WF}$, nous avons $\mathrm{Support}(c) = \frac{7}{16} = 0.43 < 0.5$ et c devient infréquent. De même, l'attribut d qui est infréquent par rapport au premier contexte,*

($\text{Support}(d) = \frac{2}{6} = 0.33$*), devient fréquent pour le deuxième contexte (*$\text{Support}(d) = \frac{8}{16} = 0.5$*).*

Requête	Fréquence	Attributs	$\mathbb{C}_{\mathbb{W}}$	$\mathbb{C}_{\mathbb{WF}}$
q_1	1	a b c	a b c	a b c
q_2	3	a b e	a b e	a b e
q_3	5	a d	a d	a b e
q_4	1	a c	a c	a b e
q_5	2	b c	b c	a d
q_6	3	b d c	b d c	a d
				a d
				a d
				a d
				a c
				b c
				b c
				b d c
				b d c
				b d c

FIGURE 12: Exemple de requêtes et les contextes d'extraction corréspondants

4.3.4 *Une performance optimisée*

Le rôle fondamental d'un index est de permettre au système de trouver les données recherchées avec le minimum d'opérations possibles. Afin d'illustrer l'avantage de l'utilisation des index maximaux, considérons l'exemple suivant : Soient a, b, et c trois attributs indexables. Supposons que $\mathbb{I}_{Close} = \{[a], [bc], [abc]\}$ et $\mathbb{I}_{Max} = \{[abc]\}$ désignent respectivement l'ensemble des index fermés et maximaux. Remarquons que $[a]$ et $[bc]$ ne sont pas maximaux car $[a] \subset [abc]$ et $[bc] \subset [abc]$). Soit une requête q qui référence les trois attributs a, b, et c. Afin d'optimiser l'évaluation de q, le système peut exploiter les index disponibles :

1. **Utilisation de l'ensemble $\mathbb{I}_{Close}$**
   - Le système peut utiliser l'index [abc].
   - Une deuxième alternative consiste à utiliser les index [a] et [bc]. Dans ce cas, le système identifie, séparément, les tuples satisfaisant q dans chacun des index [a] et [bc], Finalement une jointure des tuples retenus est réalisée en utilisant le ROWID pour répondre à q.
2. **Utilisation de l'ensemble $\mathbb{I}_{Max}$**
   - Le système utilise l'index [abc].

Il est clair que dans le premier cas, si l'index [abc] n'est pas exploité, le système doit réaliser des opérations supplémentaires pour évaluer la requête q. Remarquons que la requête q bénéficie significativement de l'index [abc], s'il est utilisé, car il est construit sur tous les attributs référencés par q. Nous pouvons dire que, les index fermés étant beaucoup plus nombreux, leur exploitation nécessite l'évaluation de plusieurs alternatives possibles ce qui peut dégrader les performances d'évaluation des requêtes.

Inversement, supposons que la requête q référence uniquement l'attribut a (ou uniquement les attributs b et c) :

1. **Utilisation de l'ensemble $\mathbb{I}_{Close}$**
   - Le système exploite l'index [a] (ou l'index [bc]).
2. **Utilisation de l'ensemble $\mathbb{I}_{Max}$**
   - Le système utilise l'index [abc].

La requête q bénéficie significativement de tous les index considérés car chaque index utilisé est définit sur tous les attributs référencés par q. Cependant, dans le deuxième cas l'index [abc] est plus large que nécessaire.

En résumé, nous pouvons dire que l'exploitation de l'un ou l'autre des ensembles $\mathbb{I}_{Close}$ et $\mathbb{I}_{Max}$ a un impact sur le coût d'évaluation des requêtes :

1. **Utilisation des index fermés :** Le système peut exploiter les index les plus larges, qui sont finalement des index maximaux (rappelons que les index maximaux sont fermés par définition). Une alternative consiste à réaliser plusieurs combinaisons sur des index de tailles plus petites. Dans ce cas de figure, l'évaluation nécessite des tâches supplémentaires. En effet, le système doit évaluer les coûts de plusieurs alternatives pour définir la meilleure combinaison des index disponibles. C'est une opération très coûteuse pour le système car il doit générer et tester plusieurs plans d'exécution des requêtes qui est lui-même un problème d'optimisation complexe [97].
2. **Utilisation des index maximaux :** Le système dispose des index les plus larges possibles. Ceci est intéressant dans le sens ou un index définit sur k attributs

$\{a_1, a_2, \ldots, a_k\}$ peut être exploité pour évaluer toutes les requêtes référençant j attributs $\{a_1, a_2, \ldots, a_j\}$ $(1 \leqslant j \leqslant k)$. En fonction du nombre d'attributs considérés, ce type d'index permet d'éviter les combinaisons additionnelles. Pour certaines requêtes, la seule lecture de l'index suffit pour trouver les données recherchées. Dans la littérature, ce type d'index est dit index couvrant (covering index en anglais) car il couvre la totalité des attributs référencés par la requête. Exploiter les index couvrants est une technique très utilisée pour optimiser le traitement des requêtes dans les systèmes de gestion des bases de données [105]. L'inconvénient dans ce cas de figure réside dans l'utilisation des index larges pour les requêtes référençant un nombre réduit d'attributs. Vu la nature complexe des requêtes de jointures en étoile impliquant souvent un nombre important d'attributs, les index maximaux semblent les plus adaptés. Dans la littérature [106], un index de jointures binaires est dit complet s'il est construit en joignant toutes les tables de dimension avec la table de fait. Un index de jointure partiel est construit en joignant certaines des tables de dimension avec la table de fait. En conséquence, l'index complet est bénéfique pour n'importe quelle requête posée sur le schéma en étoile. Il exige cependant beaucoup d'espace pour son stockage.

4.3.5 *Une mesure de qualité améliorée*

Les approches proposées pour la résolution du problème de sélection d'index ont souvent considéré le coût d'évaluation de la charge des requêtes comme le facteur clé pour recommander une configuration pertinente d'index. Parmi toutes les configurations respectant la contrainte de stockage, celle conduisant au coût minimum de la charge est recommandée. Cependant, des solutions intéressantes risquent d'être écartées par cette démarche. Nous sommes persuadés que la difficulté principale du problème de sélection des index est qu'il n'existe pas de définition *exacte* de la solution recherchée. Nous pouvons simplement exprimer le fait qu'une solution est préférable à une autre. Parfois obtenir un petit accroissement en performance demande beaucoup d'espace de stockage ce qui n'est pas toujours justifié pour l'administrateur. Dès lors nous pensons que la résolution de ce problème ne doit pas consister à considérer une seule solution mais plutôt un ensemble de solutions satisfaisantes.

Pour illustrer cette observation, considérons deux configurations C_1 et C_2. Notons par $Cost(\mathbb{W}, C_1)$ et $Cost(\mathbb{W}, C_2)$ le coût de traitement de la charge en exploitant les configurations C_1 et C_2 respectivement. Supposons que $Size(C_1) < S$ et $Size(C_2) < S$ où S est la limite de stockage fixée par l'administrateur. Supposons de plus que $Cost(\mathbb{W}, C_1) < Cost(\mathbb{W}, C_2)$ mais que les deux coûts sont suffisamment

'proches'. Supposons, par contre, que $Size(C_1) >> Size(C_2)$. Si nous nous focalisons sur le coût de la charge, nous allons recommander la configuration C_1. En pratique, il est très probable que l'administrateur opte pour la configuration C_2 qui permet de gagner énormément en espace de stockage au prix d'une faible perte de performance. L'intérêt de notre observation est de s'appuyer sur un comportement réel des administrateurs contrairement à la majeure partie des approches proposées. Ceci nous a mené à définir une mesure de qualité simple et en accord avec le comportement réel des administrateurs. En plus des coûts, habituellement pris en considération, la nouvelle mesure évalue le compromis coût/espace et permet donc de mieux évaluer la qualité d'une configuration d'index. Le cas idéal est obtenu pour une configuration C conduisant à un coût minimal de la charge et nécessitant, en même temps, un minimum d'espace de stockage. Formellement, nous définissons une mesure de qualité Profit exprimée sous forme d'un rapport entre le gain induit par une configuration d'index et l'espace nécessaire pour stocker la dite configuration :

$$\mathtt{Profit}(C) = \frac{cost(\mathbb{W}, \emptyset) - cost(\mathbb{W}, C)}{Size(C)}$$

Le numérateur exprime la différence entre le coût de traitement de la charge $\mathbb{W}$ sans aucun index et en exploitant la configuration C. Une importante différence implique une meilleure performance pour la configuration C. Le dénominateur, exprimant l'espace induit par C, est utilisé pour pénaliser les configurations présentant un besoin prohibitif en espace de stockage. L'objectif est donc de maximiser le ratio ainsi définit. Évidement cette mesure de qualité est proposée pour évaluer un ensemble de configurations qui se comportent 'raisonnablement bien' et que nous appelons *ensemble des solutions optimisées*. Les configurations ayant des faibles performances ne sont pas concernées même si l'espace nécessaire pour leur stockage est réduit.

Maintenant, comment définir *l'ensemble des solutions optimisées* ? Nous proposons de spécifier un seuil de tolérance. Notre point de référence est la configuration C conduisant au coût minimal $cost_{min}$ de la charge. Les configurations 'proches' de C i.e ayant un coût $\leqslant cost_{min} + \Delta_{cost}$ sont considérées comme intéressantes. C'est à l'administrateur de décider à quel point une configuration est intéressante en spécifiant Δ_{cost}. Par exemple, il peut juger qu'une configuration est intéressante si la différence entre son coût et celui du point de référence ne dépasse pas 10%. Finalement, la qualité des configurations retenues est évaluée avec le ratio Profitt permettant de contrôler le compromis coût/espace.

4.3.6 *Aperçu général de l'approche proposée*

Dans cette section, nous allons résumer les propositions présentées ci-dessus et décrire les principales étapes de notre approche [107]. L'approche proposée permet de recommander un ensemble optimisé des index de jointures binaires en prenant en entrée les données suivantes :

1. Une charge représentative de l'activité du système composée de n requêtes avec leurs fréquences respectives $\mathbb{W} = \{q_1^{f_1}, q_2^{f_2}, ..., q_n^{f_n}\}$,
2. Une limite maximale S de stockage alloué par l'administrateur pour l'ensemble des index recommandés,
3. Un ensemble de valeurs du support minimum décrivant différents degrés de corrélation désirés par l'administrateur,
4. Un seuil de tolérance Δ_{cost}, ajusté par l'administrateur, et exprimant à quel point un ensemble de configurations est jugé intéressant.

Les principales étapes de notre approche sont décrites dans ce qui suit (voir l'algorithme 3) :

1. **Construction du contexte d'extraction :** Cette première étape consiste à examiner la charge de requêtes. L'utilisation des attributs par l'ensemble des requêtes est décrite sous forme d'un contexte d'extraction. Ce dernier est crée sous forme d'un fichier texte oú chaque ligne désigne une requête q_i et chaque colonne un attribut indexable a_j référencé par la requête en question. Chaque ligne i, représentant une requête q_i, est dupliquée f_i fois.
2. **Sélection des configurations candidates :** Cette étape permet un élagage de l'espace de recherche en exploitant le contexte d'extraction élaboré dans l'étape précédente. Selon le degré de corrélation désiré par l'administrateur, les index non fréquents sont élagués. Afin d'assurer un élagage efficace, ce dernier est réalisé en exploitant le concept des motifs fréquents maximaux. Pour une valeur donnée du support minimum, si la configuration correspondante viole la contrainte de stockage, alors elle est éliminée. Le résultat de cette étape est composé d'un ensemble de configurations candidates.
3. **Identification d'un ensemble de configurations optimisées :** Parmi l'ensemble des configuration retenues précédemment, celle conduisant au coût minimum de la charge est identifiée comme un point de référence. Les configurations restantes sont évaluées selon le paramètre Δ_{cost}. Un ensemble de configurations optimisées est ainsi dégagé.
4. **Évaluation du profit des configurations résultantes :** Les configurations retenues dans l'étape précédente seront évaluées en tenant compte de l'espace de

Algorithme 3 : Max_IJB

Données :

- Une charge de requêtes $\mathbb{W} = \{q_1^{f_1}, q_2^{f_2}, ..., q_n^{f_n}\}$;
- Une limite de stocage S;
- Valeurs du support minimum;
- $\Delta_{cost} = 10\%$ ./* `A ajuster par l'administrateur */`

Résultat :

- $\mathbb{C}_{BJI}^{Opt}$: Une configuration optimisée d'IJB;

début

`// Construction du contexte d'extraction` $\mathbb{EC}$

$\mathbb{EC} \longleftarrow \emptyset$;

Row $\longleftarrow$ "";

pour *(Chaque* $q_i \in \mathbb{W}$*)* **faire**

Row $\longleftarrow$ Attributs_Indexables(q_i);

pour $j = 1 \rightarrow f_i$ **faire**

Writeln($\mathbb{EC}$, Row);

fin

fin

`// Identification des index maximaux en exploitant le contexte` $\mathbb{EC}$

$\mathbb{C}_{BJI}^{Conf}[..] \longleftarrow \emptyset$;`// Ensemble des configurations générées`

pour *(Chaque valeur* σ *du support)* **faire**

$\mathbb{C}_{BJI} \longleftarrow$ FPMAX($\mathbb{EC}, \sigma$);

si $Size(\mathbb{C}_{BJI}) < S$ **alors**

$\mathbb{C}_{BJI}^{Conf}[..] \longleftarrow \mathbb{C}_{BJI}^{Conf}[..] \cup \mathbb{C}_{BJI}$;

fin

fin

`// Sélection d'un ensemble de configurations optimisées` $\mathbb{SOC}$

$\mathbb{C}_{costmin} \longleftarrow$ Configuration avec coût minimal parmi $\mathbb{C}_{BJI}^{Conf}$;

$\mathbb{SOC}[..] \longleftarrow \mathbb{C}_{costmin}$;

pour $(i \leftarrow 1, size(\mathbb{C}_{BJI}^{Conf}), i++)$ **faire**

si $(Cost(\mathbb{C}_{BJI}^{Conf}[i]) - Cost(\mathbb{C}_{costmin}) < \Delta_{cost})$ **alors**

$\mathbb{SOC}[..] \longleftarrow \mathbb{SOC}[..] \cup \mathbb{C}_{BJI}^{Conf}[i]$;

fin

fin

`// Évaluation du profit de chacune des configurations optimisées`

$\mathbb{C}_{BJI}^{Opt} \longleftarrow \mathbb{SOC}[1]$;

pour $(i \leftarrow 2, size(\mathbb{SOC}), i++)$ **faire**

si $\frac{Cost(\mathbb{C}_{BJI}^{Opt})}{Size(\mathbb{C}_{BJI}^{Opt})} < \frac{Cost(\mathbb{SOC}[i])}{Size(\mathbb{SOC}[i])}$ **alors**

$\mathbb{C}_{BJI}^{Opt} \longleftarrow \mathbb{SOC}[i]$;

fin

fin

Retourner $\mathbb{C}_{BJI}^{Opt}$;

fin

stockage correspondant à chacune d'elles. Un profit relatif à chaque configuration est établi. La configuration présentant le meilleur profit est alors recommandée.

4.4 ÉTUDE EXPÉRIMENTALE

Les sections précédentes ont permis la description d'une approche basée sur la recherche des motifs fréquents maximaux pour la sélection d'un ensemble d'index pertinents pour une charge de requêtes donnée. Cette section est consacrée aux études expérimentales concernant la faisabilité et la validité de nos propositions. Pour pouvoir identifier les index maximaux, nous avons besoin d'implémenter un algorithme de recherche des motifs fréquents maximaux. Une application permettant l'évaluation des performances des index recommandés est aussi nécessaire. Ainsi, les implémentations réalisées se répartissent selon deux objectifs : (1) implémenter un algorithme de recherche des motifs fréquents maximaux et (2) développer une application afin d'évaluer les performances de l'approche proposée. Nous avons choisi d'utiliser le langage Java, afin d'obtenir un programme portable et exécutable sur n'importe quelle machine disposant d'une implémentation de la machine virtuelle Java. Dans ce qui suit, nous décrivons les données utilisées et l'outillage mis en place, ainsi que les différentes étapes de l'étude expérimentale. Enfin, nous présentons et analysons les résultats obtenus

4.4.1 *Benchmark et requêtes*

Afin d'avoir un point de référence pour notre approche, nous avons mené une évaluation expérimentale sur le même benchmark et la même charge utilisés dans [20]. Nous avons donc utilisé l'entrepôt de données issu du banc d'essais APB1 [108]. L'entrepôt considéré est constitué d'une table de faits `Actvars` et de quatre tables de dimension : `Prodlevel`, `Custlevel`, `Timelevel` et `Chanlevel`. La table 9 illustre les caractéristiques des tables de l'entrepôt.

La charge de requêtes considérée est composée de 60 requêtes (Voir annexe B). Chaque requête est caractérisée par sa fréquence d'utilisation ainsi que l'ensemble des attributs indexables qu'elle référence. La table 10 résume l'ensemble des attributs considérés.

Table	Nombre d'enregistrements	Taille de l'enregistrement(Octets)
Actvars	24 786 000	74
Prodlevel	9 000	72
Custlevel	900	24
Timelevel	24	36
Chanlevel	9	24

TABLE 9: Caractéristiques des tables du benchmatk APB-1

Attribut	Cardinalité
ClassLevel	605
GroupLevel	300
FamilyLevel	75
LineLevel	15
DivisionLevel	4
YearLevel	2
MonthLevel	12
QuarterLevel	4
RetailerLevel	99
CityLevel	4
GenderLevel	2
AllLevel	5

TABLE 10: Caractéristiques des attributs indexables

4.4.2 *Méthodologie d'évaluation*

Évidemment, nous avons comparé les performances de notre approche à celle basée sur la recherche des motifs fréquents fermés dont notre approche représente l'extension. Nous avons comparé les performances des deux approches dans les situations suivantes :

1. **Situation A :** Cette situation exploite un contexte d'extraction simple tel qu'il était utilisé dans les travaux antérieurs. Dans cette situation, chaque requête est représentée, une seule fois, par les attributs indexables qu'elle référence. Le

contexte considéré est ainsi formé de n lignes où n représente le nombre des requêtes dans la charge. Le nombre des colonnes dans chaque ligne correspond au nombre des attributs indexables référencés par la requête en question.

2. **Situation B :** Cette situation exploite un contexte d'extraction étendu tel que celui qui est décrit dans la section 4.3.3.
3. **Situation C :** Nous étudions dans cette situation l'impact de l'utilisation de la fonction profit définie dans la section 4.3.5 sur la recommandation d'une configuration optimisée.

Dans tous les tests effectués, les configurations des index fermés et maximaux sont générées et comparées. Nous avons évalué la qualité des configurations obtenues par rapport à trois principaux aspects :

- Le nombre des index générés,
- l'espace nécessaire pour stocker les index générés,
- la performance des index i.e le coût de la charge en présence des index.

Afin d'estimer le gain de performance induit par les différentes configurations générées, nous avons utilisé le modèle de coût proposé par Aouiche et al. dans [63]. Les paramètres utilisés dans ce modèle de coût sont résumés dans le tableau 11. Le coût d'exécution d'une requête q en présence d'une configuration d'index C est estimé par le nombre d'entrées-sorties nécessaires pour évaluer la requête en question. Deux phases sont nécessaires pour évaluer une requête exploitant un index de jointures binaires : la phase de parcours de l'index et la phase de lecture des n-uplets de la table indexée. La phase de parcours de l'index correspond à toutes les opérations d'entrées/sorties nécessaires pour rechercher les bitmaps permettant d'évaluer une requête donnée. La phase de lecture des n-uplets de la table indexée inclut des opérations d'entrées/sorties additionnelles permettant de lire directement les données à partir du disque [63].

Le coût d'évaluation d'une requête q_i, exploitant un index de jointure binaire, défini sur un attribut a_j, est donné par [63] :

$$Cost(q_i, I_{ijb}^{a_j}) = \log_m |a_j| - 1 + \frac{|a_j|}{m-1} + d \times \frac{||F||}{8 \times S_p} + P_F(1 - e^{\frac{N_r}{P_F}})$$

avec $N_r = d \times \frac{|F|}{|a_j|}$ et $m = \frac{S_p}{w(a_j)+S_{ptr}}$ où $w(a_j)$ et S_{ptr} désignent respectivement la taille en octets de l'attribut a_j et du pointeur d'une page système.

Dans le cas où la requête n'exploite pas les index i.e. aucun index ne correspond aux attributs de q, les jointures sont réalisées par hachage. Le nombre d'entrées/sorties nécessaires pour joindre deux tables R et S est donné par : $C_h = 3 \times (P_s + P_R)$ [109].

Symbole	Description
$\|X\|$	Cardinalité de l'attribut X
$\|\|T\|\|$	Nombre de tuples dans la table T
S_p	Taille de la page système en octets
P_T	Nombre des pages nécessaires pour stocker la table T
d	Nombres de prédicats sur l'attribut indexé
N_r	Nombre des tuples accédés par l'index IJB

TABLE 11: Paramètres du modèle de coût de Aouiche et al.

4.4.3 *Implémentation de l'algorithme FPmax*

Après la formalisation du problème, restait le choix d'un algorithme de recherche des motifs fréquents maximaux. Ce choix est primordial pour notre étude expérimentale. Les algorithmes proposés sont nombreux et la littérature est très riche [110]. Afin de les rendre exploitables par la communauté scientifique, les principaux algorithmes proposés ont été comparés grâce à l'atelier FIMI (Frequent Itemset Mining Implementation) [1] [111] associé à la conférence ICDM (International Conference on Data Mining). L'objectif de l'atelier est l'évaluation expérimentale des différents algorithmes proposés sur des jeux de données communs et en mettant à disposition de la communauté les résultats de leurs performances. Au cours de ces ateliers une vingtaine d'algorithmes ont été testés. Les algorithmes testés appartiennent à trois catégories selon le type des motifs extraits : fréquents, fréquents fermés, et fréquents maximaux [112]. Au moment de la réalisation de notre étude expérimentale, les actes de l'atelier désigne Fpmax [113] (Cf. annexe A) comme étant le meilleur algorithme dans sa catégorie (recherche des motifs fréquents maximaux). Notre choix s'est donc naturellement porté sur cet algorithme. Aussi, dans une première étape nous avons implémenté l'algorithme FPmax en java [114]. Afin d'évaluer la performance de notre implémentation, nous avons réalisé une étude expérimentale sur les bases `Mushroom` et `Retail`. Ces deux jeux de données sont communément utilisés par la communauté de fouille de données et sont disponibles sur le site de l'atelier FIMI (http ://http ://fimi.ua.ac.be/). Nous avons choisi d'utiliser deux jeux de données avec des caractéristiques différentes en termes du nombre des items, la longueur moyenne des transactions et le nombre des transactions. Le tableau 12 présente les caractéristiques des jeux de données utilisés.

1. http ://fimi.ua.ac.be/

Jeux de données	Nombre des transactions	Nombre des items	Longueur moyenne des transactions
Mushroom	8 124	119	23
Retail	88 162	16 469	30

TABLE 12: Caractéristiques jeux de données Mushroom et Retail

Nous avons comparé notre implémentation avec celles des trois algorithmes les plus performants selon FIMI : MAFIA [115], AFOPT [116], et FPMAX* [117]. Les codes source de ces algorithmes sont écrits en c++ et sont disponibles sur le site de l'atelier. Nous les avons compilé avec le compilateur `gcc 4.2.4(i486-linux-gnu)`. Notre implémentation java était compilée avec le compilateur `javac1.6.0_0-internal`. Les expérimentations ont été réalisées sur un pentiumIV 3.4GHZ Intel(R) Xenon(TM) avec 1Go de mémoire vive tournant sous Linux Ubuntu 8.04(hardy). Les temps d'exécution reportés sont ceux obtenus avec la commande `time` de linux. Les figures 13 et 14 illustrent les résultats obtenus pour les bases *mushroom* et *retail* respectivement.

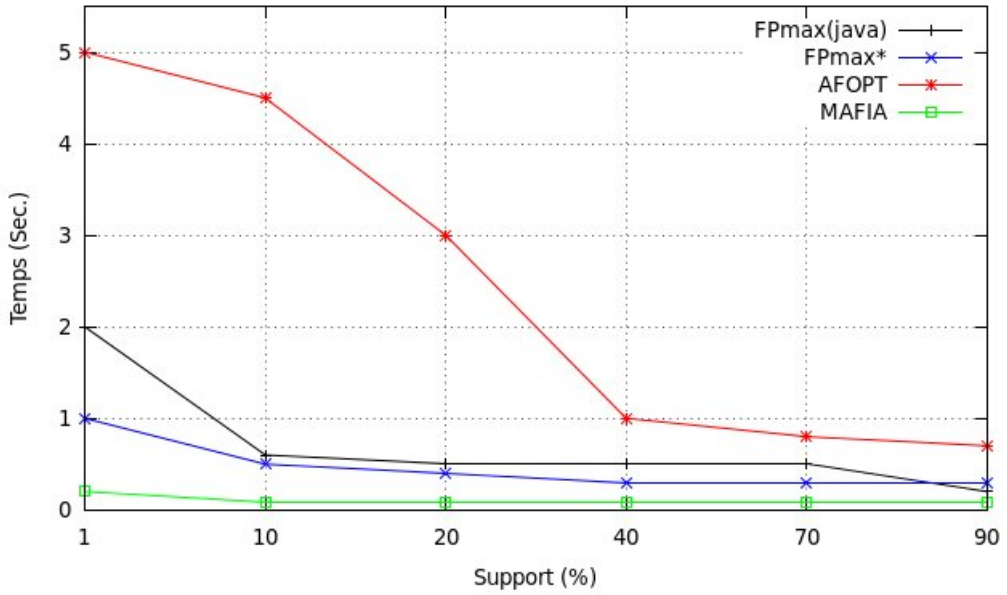


FIGURE 13: Temps d'exécution vs Support (Mushroom).

Pour la base Mushroom, notre implémentation est meilleure que AFOPT sur l'ensemble des valeurs du support. Elle a tendance à avoir les mêmes performances que

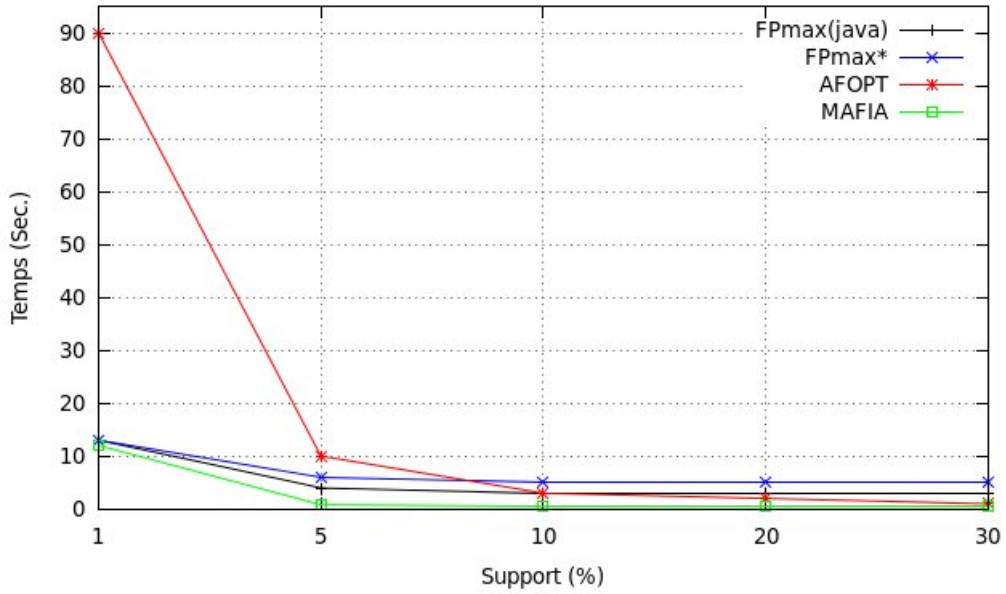


FIGURE 14: Temps d'exécution vs Support (Retail).

FPMAX* sur la majeure partie des valeurs du support. Pour la base Retail, notre implémentation présente une meilleure performance que FPMAX* sur l'ensemble des valeurs du support. Elle présente les mêmes performances que AFOPT pour les valeurs du support entre 10 et 20 et une meilleure performance pour les petites valeurs du support. Pour l'ensemble des deux jeux considérés, MAFIA détient les meilleures performances. Nous pouvons dire que les performances de notre implémentation sont ainsi comparables aux meilleures implémentations présentées dans l'atelier FIMI.

4.4.4 *CIST une application basée sur la découverte de motifs pour la sélection d'index*

Afin de valider nos contributions, nous avons développé une application pour manipuler les données et tester les performances des configurations générées. L'ensemble des fonctionnalités est facilement utilisable grâce aux interfaces graphiques. Une partie est dédiée à la manipulation des données. Une deuxième est dédiée à la visualisations des résultats obtenus. La partie fouille permet de mettre en œuvre le module de découverte des motifs à partir des contextes d'extractions fournis. Une interface utilisateur permet la sélection du contexte d'extraction et les paramètres relatifs à l'exploration de ce dernier. La figure 15 illustre l'interface principale de l'application. Elle est composée d'onglets permettant de :

- charger un contexte d'extraction,
- paramétrer la procédure de fouille (spécification du support minimum),

– extraire les index pertinents pour le contexte considéré.

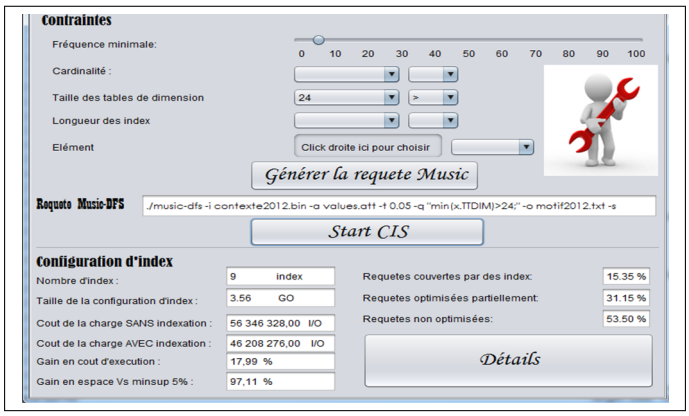


FIGURE 15: CIST : Interface principale.

Les résultats sont affichés (partie inférieure de l'interface) et l'administrateur est en mesure de récupérer des informations sur :

– le nombre des index,
– l'espace de stockage requis pour les index,
– le coût de la charge avec et sans les index,
– le pourcentage des requêtes optimisées,
– ...etc.

Le bouton «Détails» permet d'afficher plus de détails (enregistrés dans un fichier texte). Ces détails sont illustrés par la figure 16 :

1. Pour chaque index :
   – les attributs sur lesquels il est définit,
   – sa taille,
   – son coût de chargement et son coût d'accès,
2. Pour chaque requête :
   – son coût non optimisé et son coût optimisé.

Des onglets supplémentaires "Schéma de l'EDD" et "Tables et Attributs" permettent à l'administrateur de visualiser le schéma de l'entrepôt ainsi que les propriétés des tables et des attributs (figure 17).

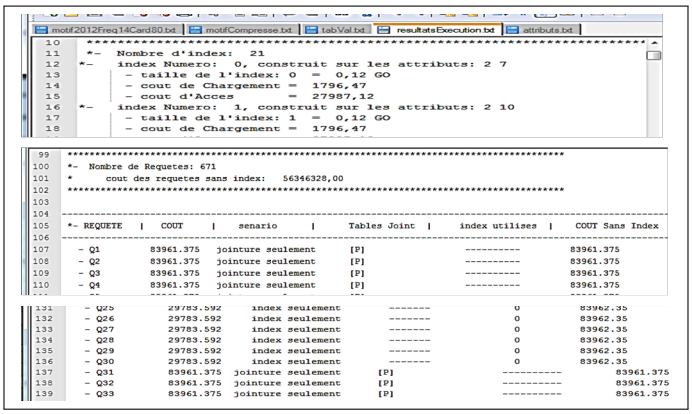


FIGURE 16: CIST : Affichage détaillé des résultats.

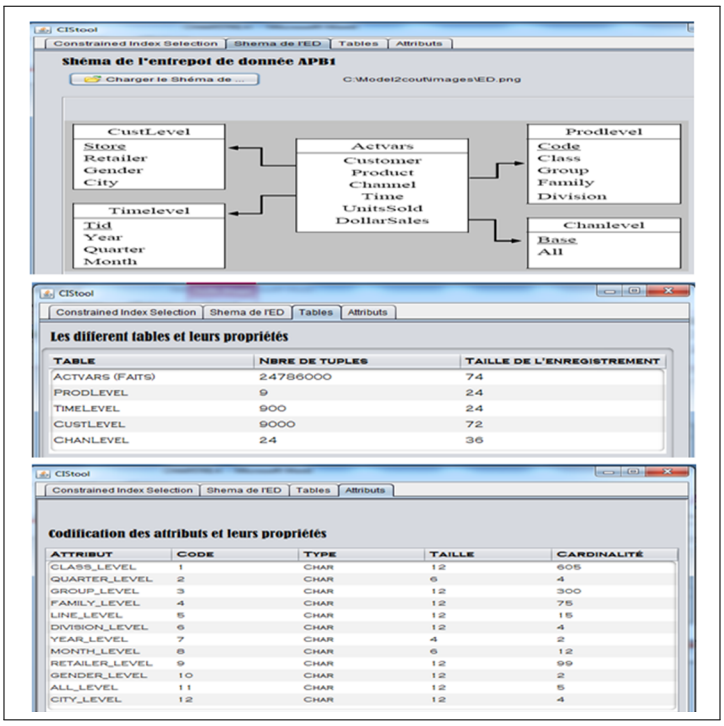


FIGURE 17: CIST : Interface des Tables et Attributs.

4.4.5 *Résultats et discussion*

4.4.5.1 *Efficacité de l'approche proposée*

L'objectif de cette première expérience (situation A) est de montrer la pertinence de l'approche proposée. Ainsi, nous avons effectué plusieurs tests pour différentes valeurs du support. Pour chacune des valeurs considérées, nous avons reporté le nombre des index générés, l'espace disque nécessaire à leur stockage et le coût de la charge en exploitant les index en question.

La figure 18 illustre le nombre des index générés pour différentes valeurs du support minimum. On observe en particulier une décroissance du nombre des index générés lorsque le support augmente suffisamment. Cette décroissance illustre le caractère anti-monotone des motifs fréquents. Au delà de la valeur 30% du support aucun index n'est généré. Nous remarquons également qu'au pire des cas notre approche génère le même nombre d'index que celle basée sur la recherche des motifs fréquents fermés. C'est le cas pour les valeurs $20\%, 25, \%$ et 30% du support. Pour les petites valeurs, nous constatons que notre approche permet une réduction significative du nombre des index. Les résultats obtenus confirme bien l'efficacité de l'élagage réalisé par notre approche. Rappelons que nous réduisons le nombre des index sans pour autant sacrifier leur pertinence. Les index éliminés sont, en effet, forcément inclus dans les index retenus. L'élagage de l'espace de recherche a un im-

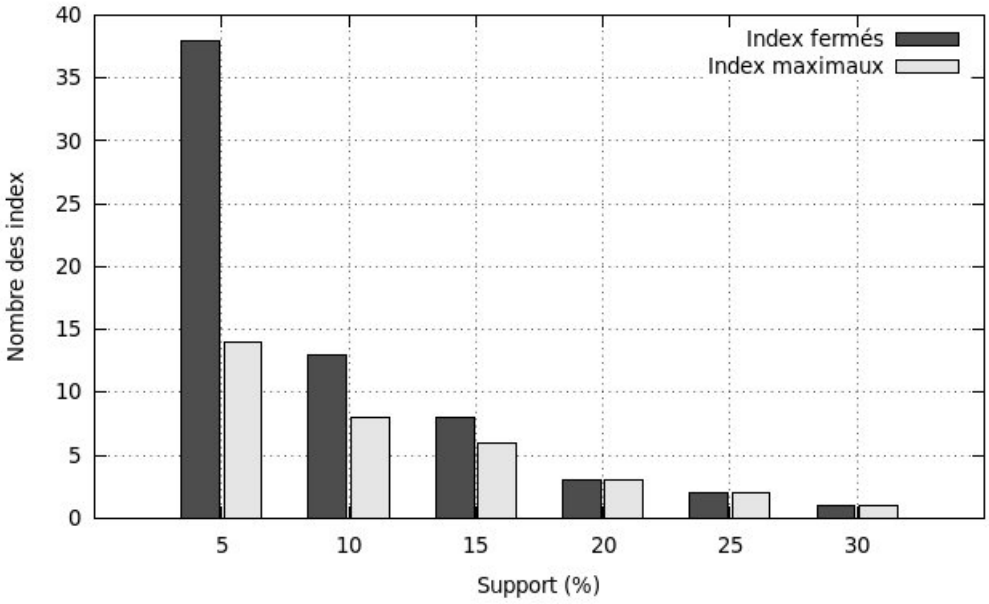


FIGURE 18: Nombre des index vs Support.

pact évident et important sur l'espace de stockage requis. La figure 19 illustre cet évidence. Les résultats obtenus traduisent une nette réduction de l'espace requis, en particulier pour les petites valeurs du support.

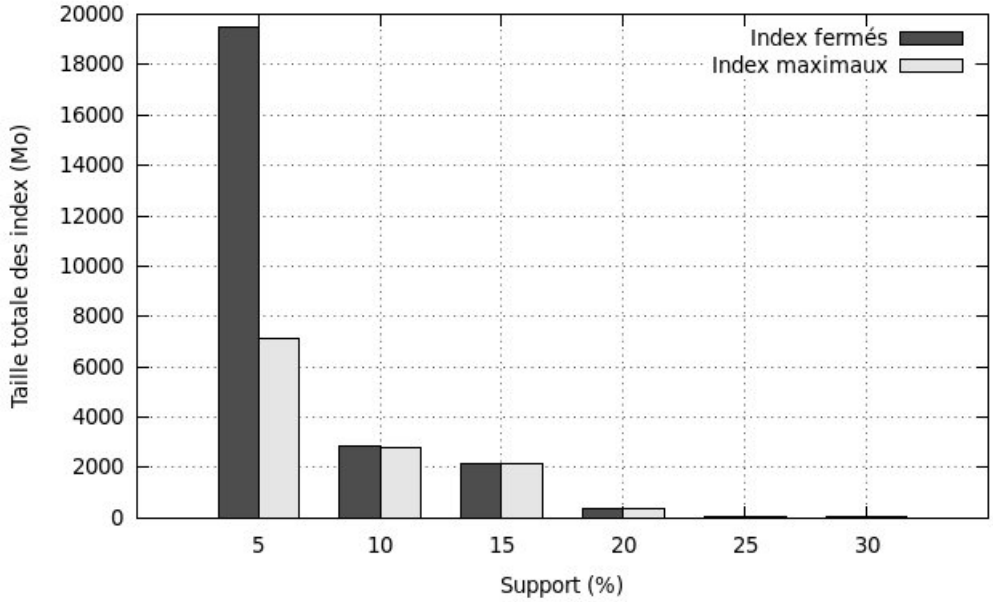


FIGURE 19: Taille des index vs Support.

Enfin le nombre des index, et par conséquence l'espace de stockage requis, apparaissent logiquement corrélées avec la valeur du support. Nous pouvons ainsi confirmer que indépendamment de la limite de l'espace de stockage S, notre approche exige toujours un espace inférieur ou, au pire des cas, égale aux approches basées sur la recherche des motifs fréquents fermés. Ceci est particulièrement très intéressant quand l'administrateur ne dispose pas d'assez d'espace à allouer aux index. Dans le cas contraire, l'espace épargné par notre approche peut être exploité par l'administrateur pour stocker d'autres structures physiques telles que les vues matérialisées.

En terme de performances, illustrées sur la figure 20, notre approche de sélection est meilleure pour les trois premières valeurs du support. Pour les autres valeurs, les deux approches génèrent les mêmes configurations et exhibent ainsi les mêmes performances. Évidement, nous allons nous focaliser sur la configuration avec le plus grand nombre d'index car elle est potentiellement la plus utile. Cette configuration est obtenue pour la valeur 5% du support. Pour cette valeur, la différence entre les deux approches est plus explicite. Notre approche obtient des résultats très nettement meilleurs pour cette valeur du support. Ceci s'explique par le fait que les

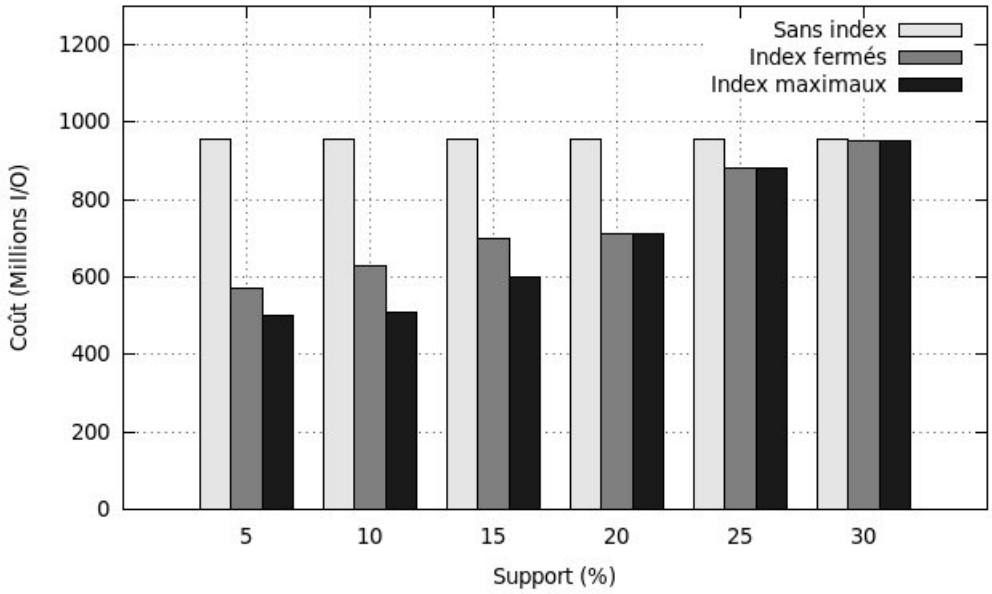


FIGURE 20: Coût de la charge vs Support.

index fermés sont presque trois fois plus nombreux que les index maximaux. Ce facteur de trois s'explique par une mauvaise stratégie d'élagage. Les conséquences sont explicites, d'une part sur l'espace de stockage requis et, d'autre part, sur les performances de la configuration générée. Les réductions obtenues sont de l'ordre de 62% pour l'espace de stockage et de 12.27% pour le coût de la charge.

4.4.5.2 *Impact de l'utilisation du contexte d'extraction enrichi*

Afin d'affiner le processus de sélection (situation B) nous avons exploité les fréquences des requêtes pour enrichir le contexte d'extraction (Cf. section 4.3.3). L'extension apportée à ce contexte signifie simplement l'enrichir par plus d'informations. Ceci permet d'obtenir des données à jour et par conséquent d'éviter les erreurs liées à l'inexactitude des informations relatives aux requêtes. Dans cette expérience, nous avons considéré les deux contextes (1) sans les fréquences C_W et (2) avec les fréquences C_{WF}. Nous avons ensuite généré les index maximaux dans les deux cas de figure. La figure 21 illustre le nombre des index générés. Observons que, selon le contexte d'extraction utilisé, nous obtenons un nombre d'index différent. Ceci confirme la proposition énoncée dans la section 4.3.3. Il s'explique simplement par le fait que, en allant d'un contexte à l'autre, certains index deviennent infréquents et sont éliminés. Par la même voie d'autres index émergent en devenant fréquents. L'impact est évident sur l'espace de stockage (Figure 22) et les performances des configurations générées (Figure 23).

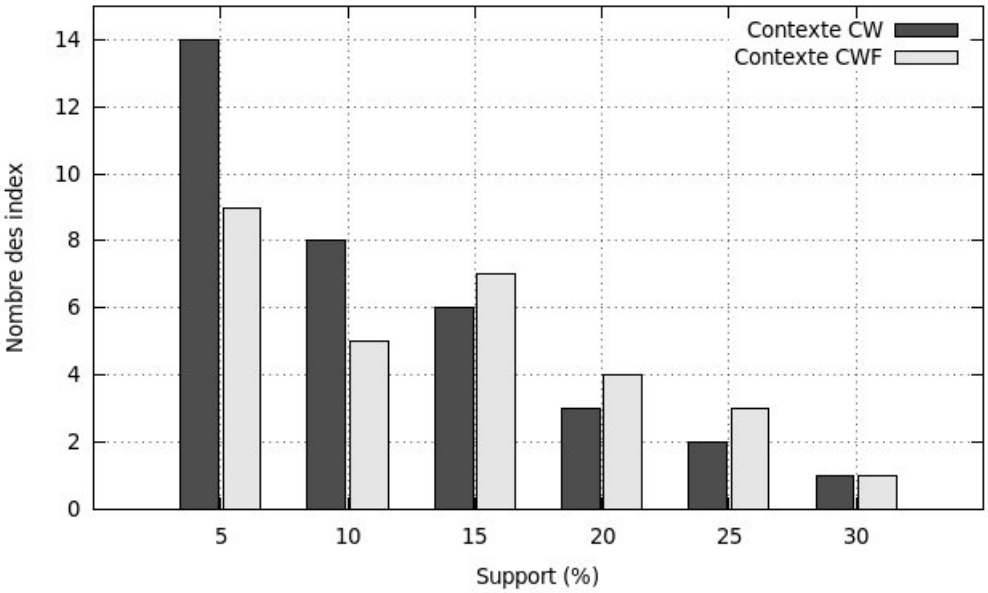


FIGURE 21: Nombre des index vs Support.

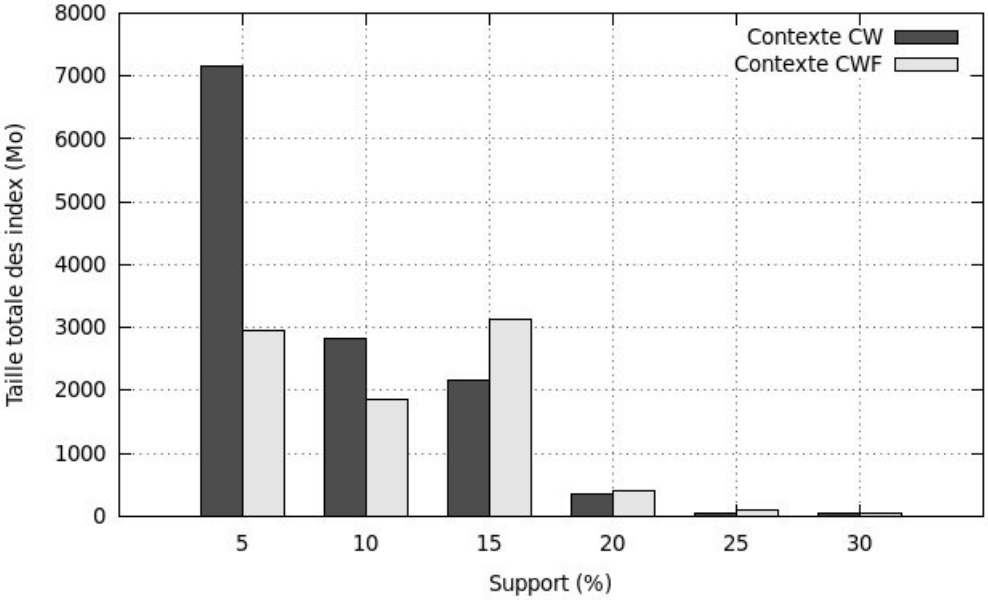


FIGURE 22: Taille des index vs Support (situation B).

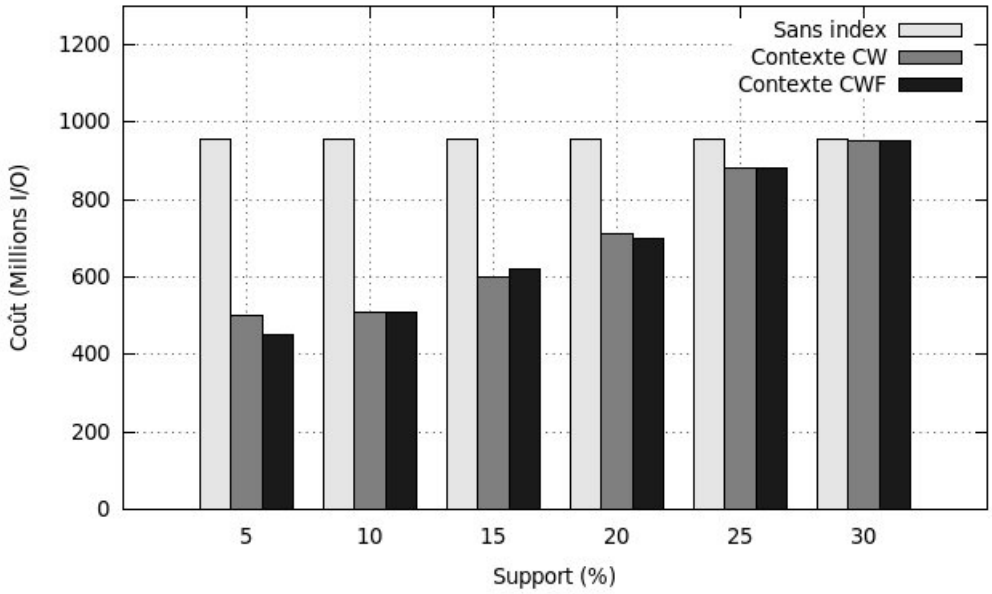


FIGURE 23: Coût de la charge vs Support (situation B).

Si on considère à nouveau la meilleure configuration (obtenue pour un support de 5%), l'exploitation du contexte C_{WF} permet des réductions de l'ordre de 9.98% et de 58.56% pour le coût de la charge et l'espace de stockage respectivement. Ceci est prévisible car les index générés, en utilisant le dit contexte, sont ceux liés aux requêtes les plus fréquentes. Les résultats obtenus illustrent bien l'intérêt de notre proposition concernant l'enrichissement du contexte d'extraction.

4.4.5.3 *Fonction profit*

Comme nous l'avons motivé dans le section 4.3.5 nous avons jugé intéressant d'explorer une autre manière de recommander une configuration pertinente. Notre critère de sélection ne se limite pas au coût de la charge qu'il faut minimiser mais plutôt de maximiser le profit de la configuration retenue. Rappelons que cette orientation est inspirée du comportement réel des administrateurs et leurs vision au compromis coût/espace. Ainsi, l'objectif de cette expérimentation est d'évaluer un ensemble de bonnes configurations en utilisant la fonction *profit* proposée. Nous avons reproduit les expérimentations précédentes avec la fonction *profit*. Les tableaux 13 et 14 récapitulent les résultats obtenus pour les index fermés et les index maximaux respectivement. Si nous considérons, par exemple, un seuil de tolérance $\Delta_{cost} = 11\%$, nous remarquons alors que deux configurations sont concernées par la fonction profit. Il s'agit des configurations relatives aux valeurs 5% et 10% du support. Pour l'ensemble des deux cas, la configuration des index maximaux avec le support 10% exhibe le meilleur profit. Nous expliquons l'intérêt pratique de cette configuration

par le fait qu'elle permet une réduction d'environ 40.00% de l'espace de stockage contre une perte de performance de 2.04% seulement.

	Coût de la charge				
Support(%)	**Sans index**	**Index fermés**	$\Delta_{cost}(\%)$	**Espace(Mo)**	**Profit**
5	954.9320	570.3737	0	19459.80	0.02
10	954.9320	630.6053	10.56	2883.81	0.11
15	954.9320	700.0732	22.74	2159.90	0.11
20	954.9320	710.6966	24.60	342.75	0.71
25	954.9320	880.1102	54.30	50.23	1.48
30	954.9320	950.6397	66.66	35.46	0.12

TABLE 13: Evaluation du Profit (Index fermés)

	Coût de la charge				
Support(%)	**Sans index**	**Index maximaux**	$\Delta_{cost}(\%)$	**Espace(Mo)**	**Profit**
5	954.9320	500.3526	0	7156.84	0.06
10	954.9320	510.6053	2.04	2815.85	0.15
15	954.9320	600.0732	19.93	2159.90	0.16
20	954.9320	710.6966	42.03	342.75	0.71
25	954.9320	880.1102	75.89	50.23	1.48
30	954.9320	950.6397	89.99	35.46	0.12

TABLE 14: Evaluation du Profit (Index maximaux)

4.5 SYNTHÈSE DU CHAPITRE

Plusieurs directions ont été poursuivies afin d'optimiser la sélection d'un ensemble pertinent d'index pour une charge de requêtes donnée. Avec l'avènement des tech-

niques de fouille de données et leur généralisation, quelques propositions de résolution, s'appuyant sur la fouille de données, commencent à émerger. Elles ont été appliquées particulièrement à la sélection des index de jointure binaire. Les travaux initiateurs, basés sur la recherche des motifs fréquents fermés, font figure de référence dans la mesure ou ils s'appuient sur un socle théorique solide. Cependant, l'analyse de ces travaux nous a permis de mettre en évidence l'inadéquation des motifs fréquents fermés pour optimiser la résolution du problème abordé. Le nombre des index manipulés ainsi que leur redondance sont deux limites majeurs qui entravent l'efficacité de leur utilisation, plus particulièrement pour des charges volumineuses et/ou les petites valeurs du support. Ce constat a motivé la proposition d'une nouvelle approche de résolution en recherchant une modélisation adéquate du problème qui se prête mieux à sa résolution. Nous avons bénéficié en particulier des propriétés intéressantes des motifs fréquents maximaux pour proposer notre approche en ayant en vue l'idée d'avoir une garantie théorique de nos propositions. La nouvelle approche présente trois principaux avantages :

Premièrement, la solution proposée permet un élagage judicieux de l'espace de recherche en évitant complètement la génération dupliquée des index. Elle permet de réduire autant que possible le nombre des index maintenus sans sacrifier la qualité des index retenus. La minimisation du nombre d'index pertinents est une direction intéressante pour minimiser par la même voie l'espace de stockage requis. Nous avons en particulier prouvé que, quelque soit la limite de l'espace de stockage réservé aux index, notre approche exige toujours un espace inférieur ou, au pire des cas, égal aux approches basées sur la recherche des motifs fréquents fermés.

Deuxièmement, nous avons étendu le contexte d'extraction utilisé jusqu'à présent. Fixer l'ensemble des fréqeunces à '1' est une problématique puisque'il ne prend pas en considération l'importance particulière de chaque requête dans la charge. L'information utilisée pour valider les index pertinents est incomplète, ce qui biaise certainement les résultats de sélection. Pour pallier ce problème, il faudrait adapter le contexte d'extraction en fonction des fréquences relatives aux requêtes. Nous avons ainsi proposé un nouveau contexte enrichi où la référence à une requête est dupliquée autant de fois que sa fréquence.

Troisièmement, nous avons proposé une modification de la manière d'évaluer les performances d'une configuration donnée. En se focalisant sur la configuration conduisant au coût minimal de la charge, les approches précédentes ont tendance à s'éloigner de la perception pratique et naturelle des administrateurs. Nous sommes convaincus que ce genre de problème n'a pas une seule solution optimisée, mais potentiellement plusieurs, en fonction de la vision de l'administrateur au compromis coût/espace car ces deux paramètres sont généralement divergeants. Nous pensons

que ce compromis doit guider la prise en considération de certaines configurations que l'on peut négliger. La mesure de qualité proposée exprime au mieux un tel compromis et permet de recommander la configuration conduisant à un gain substantiel en espace de stockage au prix d'une faible perte de coût. Pour résoudre ce problème applicatif, nous nous donnons comme objectif de satisfaire deux contraintes à priori antagonistes.

Notre étude expérimentale a été réalisée en deux temps. Dans un premier temps nous avons implémenté un algorithme de recherche des motifs fréquents maximaux, FPMAX en l'occurrence. Les performances de notre implémentation ont été testées sur les jeux de données *Mushroom* et *Retail* disponibles sur le site des ateliers FIMI. Les résultats obtenus montrent que les performances de notre implémentation sont comparables aux meilleures implémentations présentées dans FIMI. Cette implémentation a servi pour la validation de notre approche de sélection d'index. Nous avons ensuite comparé les performances de notre approche à celle basée sur la recherche des motifs fréquents fermés dont notre approche représente l'extension. Les différentes expérimentations menées ont montré l'intérêt de nos propositions. Grâce aux propriétés des motifs exploités, la différence entre les deux approches est plus explicite pour les configurations potentiellement utile i.e comportant le plus grand nombre d'index. Pour ce cas de figure les réductions obtenus sont de l'ordre de 62.00% et de 63.23% pour le nombre d'index et l'espace de stockage respectivement. La configuration générée par notre approche réalise une amélioration du coût de la charge d'environ 12.27%. Nous avons également évalué les différentes configurations en appliquant la mesure de qualité proposée dans la section 4.3.5. La meilleure solution obtenue dans cette expérimentation permet une réduction d'environ 40.00% de l'espace de stockage contre une perte de performance de 2.04% seulement. En pratique c'est une solution très intéréssante car un administrateur peut sacrifier un peu du coût d'évaluation des requêtes au prix d'une réduction intéressante de l'espace de stockage.

5

FRAGMENTATION VERTICALE DES BASES DE DONNÉES

Sommaire

5.1 INTRODUCTION

La quête d'une efficacité de traitement, justifiée par la diversité et la complexité des requêtes utilisateurs, contraint les administrateurs à considérer différentes techniques d'optimisations de performances. Une des techniques d'optimisation permettant de réduire le coût de traitement des requêtes en minimisant le volume des données manipulées est la fragmentation. Ce chapitre présente, dans un premier temps, les notions générales sur la fragmentation des bases de données. Nous nous intéressons ensuite plus particulièrement à la fragmentation verticale dans les bases et les entrepôts de données relationnels.

Après une description des différents types de la fragmentation des données, nous présentons la formalisation et la complexité du problème de sélection d'un schéma de fragmentation verticale. Nous exposons ensuite un état de l'art sur la problématique étudiée. L'état de l'art couvre deux contextes : les bases de données et les entrepôts de données. Nous exposons et montrons la variété des approches de résolution proposées allant des affinités à la fouille de données passant par les métaheuristiques avant de conclure par une synthèse qui fait ressortir les avantages et les limites des différentes techniques de résolution proposées.

5.2 GÉNÉRALITÉS SUR LA FRAGMENTATION DES DONNÉES

Paradoxalement, un système de base de données accède souvent à une grande quantité de données pour en extraire ou de mettre à jour un nombre relativement faible de valeurs répondant à une requête d'un utilisateur. Un exemple typique est la projection sur un seul attribut d'une table dont le schéma se compose d'un nombre important d'attributs. Ainsi, l'efficacité de traitement des requêtes est limitée par un nombre important d'opérations de transfert de données (Opérations d'Entrée/Sortie) en accédant à la mémoire secondaire. Ces opérations représentent le facteur clé qui caractérise les performances des systèmes de bases de données [118].

La quête d'une efficacité de traitement, justifiée par la diversité et la complexité des requêtes utilisateurs, contraint les administrateurs à considérer différentes techniques d'optimisations de performances. Une des techniques d'optimisation permettant de réduire le coût de traitement d'une requête en minimisant le volume des données manipulées est la fragmentation. Elle représente un moyen efficace pour améliorer les performances dans les systèmes de base de données où un pourcentage significatif du temps de traitement des requêtes est épuisé lors des parcours exhaustifs des tables. La fragmentation consiste à diviser un ensemble de données en plusieurs fragments, appelés aussi partitions, de manière à ce que la combinaison des fragments recouvre l'intégralité des données sources sans ajout ni perte d'information [119] [120]. Le résultat d'un processus de fragmentation est un ensemble de fragments définis par un schéma de fragmentation [17]. Un schéma de fragmentation doit présenter les trois propriétés suivantes [17] :

- La complétude qui assure que chaque instance de l'ensemble fragmenté appartient à au moins un fragment.
- La reconstruction qui garantit la reconstitution de l'ensemble de données sources sans perte ni ajout d'information.
- La disjonction qui indique que les fragments doivent être disjoints deux à deux.

Outre sa nature intuitive, la fragmentation permet une meilleure flexibilité dans la gestion des données. En effet, les partitions de plus petites tailles peuvent être gérées et manipulées collectivement ou individuellement [20]. D'autre part, lors des opérations de maintenance, la base de données est généralement non accessible, ce qui pénalise les utilisateurs du système. La fragmentation permet de cibler la tâche de maintenance où seules certaines partitions seront concernées par cette tâche, ce qui réduit le temps d'arrêt du système. Les partitions non concernées par la maintenance resteront toujours disponibles. Lors de la fragmentation d'une table, l'administrateur peut spécifier un emplacement physique pour stocker chaque partition. Si les emplacements physiques des différentes partitions se trouvent sur des espaces physiques

différents, alors une panne au niveau d'un espace de stockage n'affecte que les partitions stockées sur ce dernier, les autres partitions resteront toujours disponibles [20].

La fragmentation est utile dans plusieurs contextes [121][122][123][124] :

- Dans le cas des données centralisées, les applications nécessitent souvent un ensemble réduit des données pour répondre aux requêtes des utilisateurs. Par conséquent, la fragmentation peut aider à réduire l'accès aux données non pertinentes, en réduisant ainsi le nombre d'accès au disque, et améliorer ainsi les performances du système ;
- Dans le cas des données distribuées, la fragmentation est utilisée comme une étape préliminaire fondamentale pour trouver le meilleur schéma de fragmentation avant d'affecter les différents fragments aux différents sites distants ;
- La fragmentation des données permet l'exécution parallèle des requêtes en les remplaçant par un ensemble de sous-requêtes opérant chacune sur des fragments différents ce qui réduit considérablement le coût d'exécution des requêtes.

Dans la littérature, trois types de fragmentation sont définis [17][125][126] : la fragmentation verticale, la fragmentation horizontale et la fragmentation mixte.

- La fragmentation **verticale** partitionne une table T en k fragments $F_1, \dots F_k$ (k le nombre de fragments). Les fragments sont obtenus par des opérations de projection selon un ou plusieurs attributs a_i ($i = 1..n$ où n est le nombre d'attributs). Chaque fragments doit contenir les clés permettant la reconstruction des données source (jointure des fragments).
- La fragmentation **horizontale** partitionne une table T selon des prédicats de sélections. Un fragment horizontal est obtenu par une opération de restriction sur l'ensemble T. La reconstruction des données source à partir des fragments horizontaux est obtenue par une opération d'union. Il esiste deux variantes de ce type de fragmentation [20] : la fragmentation primaire et la fragmentation dérivée.
  - La fragmentation horizontale **primaire** d'une table T est réalisée en utilisant un ou plusieurs attributs appelés clés de fragmentation. Chaque partition de la table est définie par une conjonction de prédicats définis sur la clé de fragmentation. Toute requête utilisant un prédicat de sélection défini sur un ou plusieurs attributs contenus dans la clé de fragmentation sera dirigée vers les partitions vérifiant ce prédicat (partitions pertinentes). Toutes les autres partitions ne seront pas concernées par cette requête. Ainsi, ce type de fragmentation permet de réduire le coût de traitement d'une requête en minimisant le nombre d'accès aux enregistrements non nécessaires.

- La fragmentation horizontale **dérivée** consiste à partitionner une table T selon les fragments d'une autre Table S. Elle permet de partitionner la jointure entre ces deux tables en plusieurs sous-jointures. Chaque sous-jointure est effectuée entre une partition de T et la partition de S qui a permis de la générer. Lors de l'exécution d'une requête impliquant une jointure entre la table T et S, seules les jointures contenant des partitions pertinentes seront effectuées. La fragmentation horizontale dérivée réduit ainsi le coût de traitement d'une opération de jointure entre les tables T et S.

– La fragmentation **mixte** partitionne verticalement des fragments horizontaux ou horizontalement des fragments verticaux.

La figure 24 schématise le principe de la fragmentation horizontale et la fragmentation verticale.

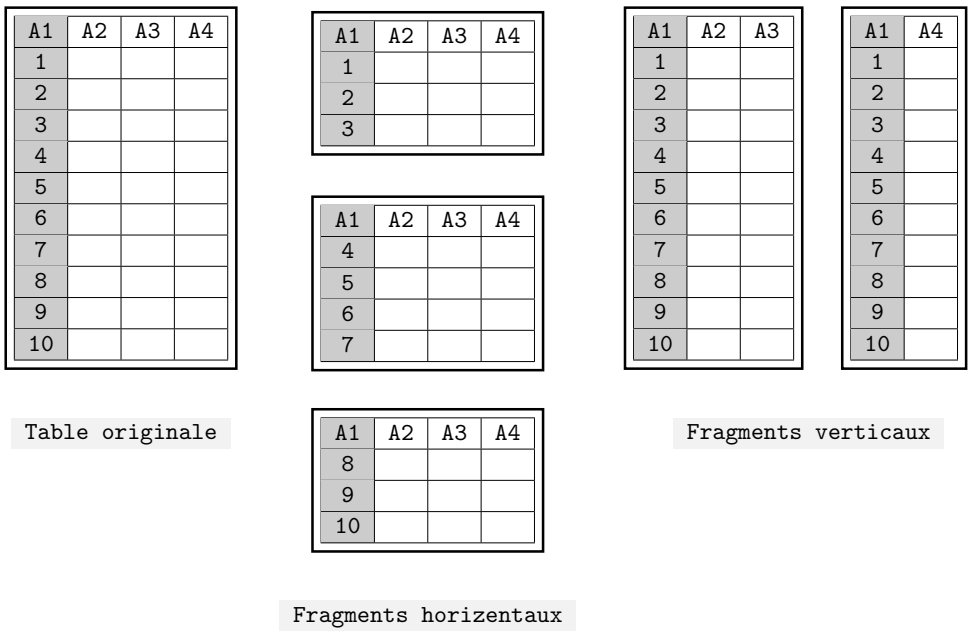


FIGURE 24: Fragmentaion horizontale et Fragmentation verticale (l'attribut `A1` est la clé).

En raison de la complexité à la fois des problèmes de la fragmentation verticale et horizontale, elles étaient souvent traitées séparément, même si en pratique elles sont utilisées pour atteindre les mêmes objectifs, à savoir, l'amélioration de la performance du système. Ceci est motivé par le fait que chacun des problèmes a un immense espace de recherche de solutions optimales et est assez difficile pour être étudier en lui-même. Les deux problèmes sont prouvés à être NP-complet [23][29]. Ainsi, les travaux portant sur le problème de la fragmentation d'une manière géné-

rale ont mis l'accent sur la réduction de sa complexité et d'en proposer des solutions approchées.

Les premiers travaux sur la fragmentation ont été proposés dans le contexte des bases de données relationnelles centralisées. Avec l'émergence de nouveaux modèles de données, les approches existantes de la fragmentation ont été largement adoptées en prenant en considération les caractéristiques de chacun des modèles émergents. A titre d'exemple, dans un contexte centralisé, la fragmentation vise à réduire le coût de traitement des requêtes alors que dans le contexte distribué, elle vise, en plus, une meilleure répartition des données sur des sites géographiquement distants ce qui favorise le traitement parallèle des requêtes. Des approches ont été ainsi proposées pour les bases de données distribuées [121][122][123][127] [128][129][130][131], les bases de données orientées objet [132][133][134][135][136][137][138][139][140], les bases de données parallèles [141][142][143] [144] et les entrepôts de données [17][20] [145][146][147] [148][149][150]. La littérature à ce sujet est donc vaste. La présentation exhaustive des travaux relatifs aux différents contexte est loin d'être l'objectif de ce travail. Nous nous intéressons dans la suite à la fragmentation verticale. Cette orientation est motivée par le fait qu'elle était peu étudiée dans le contexte des entrepôts de données relationnels [145].

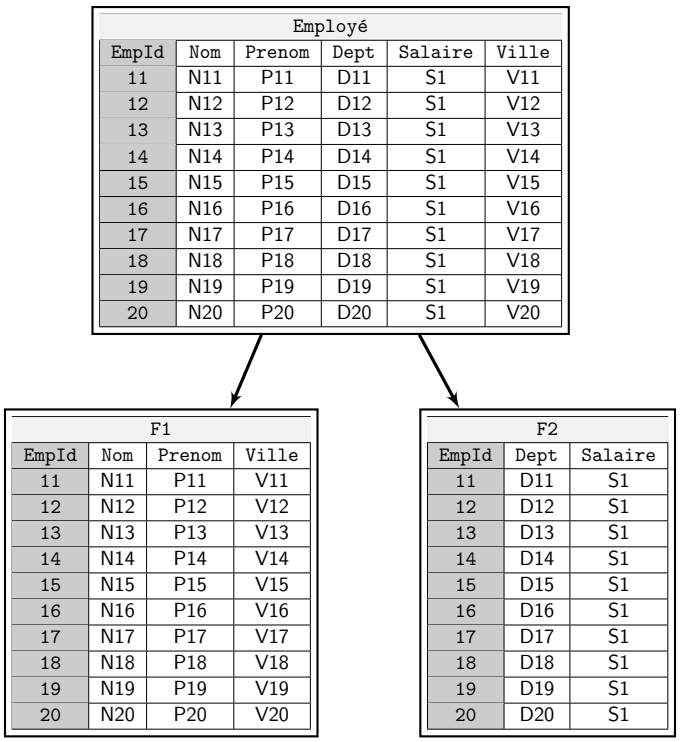
5.3 LA FRAGMENTATION VERTICALE DES DONNÉES

5.3.1 *Principe*

Une table de données est stockée avec la totalité des attributs qui la décrivent. Pendant le traitement d'une requête, tous les attributs sont chargés dans la mémoire centrale, même si seulement une partie d'entre eux est nécessaire. Si la base est organisée de manière à récupérer exclusivement les attributs nécessaires, le volume des données manipulées sera réduit. Cette organisation des données est dite fragmentation verticale. Intuitivement, la fragmentation verticale consiste à regrouper les attributs fréquemment utilisés ensemble par les requêtes. Elle vise ainsi à associer les fragments pour les besoins des requêtes en faisant apparaître une organisation significative des données. Un fragment vertical est construit par une opération de projection. L'ensemble des données source est reconstruit par jointure des fragments [29]. La fragmentation verticale est ainsi une technique permettant une économie substantielle des coûts de traitement des requêtes utilisateur en réduisant le nombre d'accès disque pour la récupération des données pertinentes [127][128]. Intuitivement, un fragment est non pertinent pour une requête donnée s'il ne contient aucun attribut référencé par la dite requête. Ainsi, le gain de traitement des requêtes est fonction

du volume des fragments non pertinents par rapport au volume de la table initiale. Plus le volume des fragments non pertinents est grand, meilleure est la performance de traitement d'une requête.

Exemple 12. *La figure 25 illustre le principe de la fragmentation verticale. Considérons une table relative aux employés d'une entreprise (table d'origine). Supposons que l'administrateur remarque que les différents départements exploitent, à travers les requêtes, fréquemment les attributs Nom, Prenom et Ville. Dans une telle situation, il peut partitionner la tables des employés en deux fragments :* F_1*(Nom, Prenom, Ville) et* F_2*(Dept, Salaire). Ceci permet d'améliorer le traitement des requêtes en évitant l'accès aux informations inutiles.*



Employé

EmpId	Nom	Prenom	Dept	Salaire	Ville
11	N11	P11	D11	S1	V11
12	N12	P12	D12	S1	V12
13	N13	P13	D13	S1	V13
14	N14	P14	D14	S1	V14
15	N15	P15	D15	S1	V15
16	N16	P16	D16	S1	V16
17	N17	P17	D17	S1	V17
18	N18	P18	D18	S1	V18
19	N19	P19	D19	S1	V19
20	N20	P20	D20	S1	V20

F1

EmpId	Nom	Prenom	Ville
11	N11	P11	V11
12	N12	P12	V12
13	N13	P13	V13
14	N14	P14	V14
15	N15	P15	V15
16	N16	P16	V16
17	N17	P17	V17
18	N18	P18	V18
19	N19	P19	V19
20	N20	P20	V20

F2

EmpId	Dept	Salaire
11	D11	S1
12	D12	S1
13	D13	S1
14	D14	S1
15	D15	S1
16	D16	S1
17	D17	S1
18	D18	S1
19	D19	S1
20	D20	S1

FIGURE 25: Exemple de la fragmentation verticale

5.3.2 *Formalisation et complexité*

Les attributs référencés par une requête sont pertinents à son exécution. Ainsi, le schéma de fragmentation idéal est obtenu si chaque requête accède à un seul fragment contenant exactement les attributs qu'elle référence évitant ainsi des jointures supplémentaires. Cette situation est irréaliste car les requêtes exploitent souvent en

commun différents attributs, ce qui complexifie la procédure de la fragmentation verticale. En effet, la meilleure répartition des attributs pour une requête n'est pas nécessairement avantageuse pour une autre. En considérant une charge de requête, l'objectif réaliste consiste à choisir un schéma de fragmentation minimisant le coût de traitement de la charge en assurant le minimum d'accès possibles aux fragments construits. Une formulation générale du problème de la fragmentation verticale est la suivante :

Definition 11 (Problème de la fragmentation verticale). *Etant données une table* $\mathcal{T} = \{a_1, a_2, \ldots, a_k\}$ *de* k *attributs et une charge de* n *requêtes* $W = \{q_1^{f_1}, q_2^{f_2}, \ldots, q_n^{f_n}\}$*, où chaque requête* q_i *est caractérisée par sa fréquence d'utilisation* f_i*, l'objectif est de sélectionner, parmi tous les schémas de fragmentation possibles, un schéma de fragmentation* $\mathcal{F} = \{F_1, F_2, \ldots, F_m\}$ *tel que :*

1. *Chaque fragment* $F_i \in \mathcal{F}$ *est composé d'un sous-ensemble des attributs de* $\mathcal{T}$ *plus l'identificateur (clé) qui est utilisé pour les éventuelles opérations de jointures.*
2. $\forall F_i \in \mathcal{F}, \forall F_j \in \mathcal{F} : F_i \cap F_j = \emptyset$ *(sauf pour la clé de jointure)*
3. $\mathcal{T} = \bowtie_{i=1}^{m} F_i$
4. *Le coût du traitement de la charge de travail* $\mathcal{W}$ *en exploitant le schéma de partitionnement* $\mathcal{F}$ *est minimum.*

Notons que certaines formulations considèrent le nombre des fragments comme une donnée fixée au préalable par l'administrateur [124][128][131][145]. La sélection d'un schéma de fragmentation verticale optimal est un problème NP-complet [28][29]. A titre comparatif, il est plus compliqué que celui de la fragmentation horizontale en raison du nombre exponentiel d'alternatives possibles [123]. En effet, si m est le nombre d'attributs concernés par le processus de fragmentation, alors le nombre de fragments possibles est donnée par le nombre de Bell, qui est approximativement $B(m) = m^m$ [151]. Par conséquent, trouver un schéma de fragmentation approprié est une tâche fastidieuse, même pour un administrateur chevronné. Avec ce nombre de fragments possibles, il est impossible d'obtenir une solution optimale en comparant les différents schémas de fragmentation de manière exhaustive [124]. Ainsi, les travaux portant sur le problème de la fragmentation verticale ont mis l'accent sur la réduction de sa complexité et d'en trouver des solutions optimisées.

5.4 ÉTAT DE L'ART

Comme nous l'avons indiqué plus haut, les premiers travaux relatifs à la fragmentation verticale ont été proposés dans le contexte des bases de données relationnelles

centralisées. Avec l'émergence de nouevaux modèles de données, les approches existantes de la fragmentation ont été largement adoptées en prenant en considération les caractéristiques de chacun des modèles émergents. Les différents travaux liés à d'autres contexte tels que les bases de données distribuées, orientée objet ou parallèles , s'inspirent largement de ceux proposés pour les bases de données relationnelles centralisées. C'est pourquoi nous nous limitons à présenter ici les principaux travaux proposés pour la fragmentation verticale pour le modèle relationnel centralisé. Nous les avons regroupé en deux catégories selon le contexte étudié : (1) les bases de données et (2) les entrepôts de données.

5.4.1 *Fragmentation verticale dans les bases de données*

Les premières solutions proposées pour la résolution du problème de la fragmentation verticale sont basées sur la notion d'affinité des attributs. Avant d'exposer les principaux travaux, nous présentons à travers un exemple la notion d'affinité et son utilisation pour le regroupement des attributs formant les fragments. L'exemple est extrait de [152].

La première étape consiste à créer la matrice d'usage des attributs (AUM : Attribute Usage Matrix). Elle contient une ligne pour chaque requête et une colonne pour chaque attribut. Chaque case $AUM(q_i, a_j)$ est mise à 1 si la requête q_i référence l'attribut a_j et 0 sinon :

$$AUM(q_i, a_j) = \begin{cases} 1 & \text{si } q_i \text{ référence } a_j \\ 0 & \text{sinon.} \end{cases}$$

La matrice AUM est associée à un vecteur contenant les fréquences des requêtes. La table 15 illustre un exemple de la matrice d'usage.

La deuxième étape permet de construire la matrice d'affinité par pair d'attributs. L'affinité entre deux attributs a_i et a_j est définie comme suit :

$$Aff(a_i, a_j) = \sum_{k=1}^{n} AUM(q_k, a_i) * AUM(q_k, a_j) * f_k$$

n étant le nombre des requêtes. La matrice d'affinité correspondante à l'exemple précédent est illustrée par le tableau 16. A partir de la matrice d'affinité, on procède à un regroupement des attributs ayant les plus hautes affinités.

5.4.1.1 *Travaux de Hoffer and Severance*

Hoffer and Severance [153] proposent une approche pour la fragmentation verticale basée sur l'algorithme BEA (Bond Energy Algorithm) [154]. En exploitant la ma-

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	Fréq.(q_i)
q_1	1	0	0	0	1	0	1	0	0	0	25
q_2	0	1	1	0	0	0	0	1	1	0	50
q_3	0	0	0	1	0	1	0	0	0	1	25
q_4	0	1	0	0	0	0	1	1	0	0	35
q_5	1	1	1	0	1	0	1	1	1	0	25
q_6	1	0	0	0	1	0	0	0	0	0	25
q_7	0	0	1	0	0	0	0	0	1	0	25
q_8	0	0	1	1	0	1	0	0	1	1	15

TABLE 15: Exemple de la matrice d'usage des attributs : AUM

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
a_1	75	25	25	0	75	0	50	25	25	0
a_2	25	110	75	0	25	0	60	110	75	0
a_3	25	75	115	15	25	15	25	75	115	15
a_4	0	0	15	40	0	40	0	0	0	40
a_5	75	25	25	0	75	0	50	25	25	0
a_6	0	0	15	40	0	40	0	0	15	40
a_7	50	60	25	0	50	0	85	60	60	0
a_8	25	110	75	0	25	0	60	110	75	0
a_9	25	75	115	15	25	15	25	75	115	15
a_{10}	0	0	15	40	0	40	0	0	15	40

TABLE 16: Exemple de la matrice d'affinité

trice d'affinité des attributs, l'algorithme BEA procède par permutation des lignes et de colonnes afin d'identifier des ensembles d'attributs de telle sorte que la valeur de la fonction d'affinité est maximisée. Le tableau 17 indique le résultat de l'application de l'algorithme BEA à l'exemple précédent. Cependant, dans l'approche proposée, il est difficile de déterminer le nombre de fragments et les attributs liés à chacun d'eux. L'interprétation est subjective et nécessite une intervention humaine et ne peut pas être considérée comme fiable [155].

	a_5	a_1	a_7	a_2	a_8	a_3	a_9	a_{10}	a_4	a_6
a_5	75	75	50	25	25	25	25	0	0	0
a_1	75	75	50	25	25	25	25	0	0	0
a_7	50	50	85	60	60	25	60	0	0	0
a_2	25	25	60	110	110	75	75	0	0	0
a_8	25	25	60	110	110	75	75	0	0	0
a_3	25	25	25	75	75	115	115	15	15	15
a_9	25	25	25	75	75	115	115	15	15	15
a_{10}	0	0	0	0	0	15	15	40	40	40
a_4	0	0	0	0	0	15	15	40	40	40
a_6	0	0	0	0	0	15	15	40	40	40

TABLE 17: Résultat de l'algorithme BEA

5.4.1.2 *Travaux de Navathe et al.*

Navathe et al. [156] ont étendu la démarche précédente en ajoutant une deuxième phase à l'algorithme BEA dans un effort pour réduire la subjectivité de l'interprétation finale. L'approche proposée est dite fragmentation binaire car elle génère deux fragments. Dans une première étape, les données en entrée (requêtes et fréquences) permettent de créer les matrices d'utilisation des attributs (AUM) et d'affinité attribut (AAM). C'est au biais de cette dernière qu'un premier regroupement des attributs sera effectué de la même manière que dans l'approche précédente. Le résultat de ce traitement, appelé matrice d'affinité fragmentée CAM (Clustered Affinity Matrix) sera l'entrée de la seconde étape qui représente la contribution des auteurs. La deuxième étape consiste à diviser de façon récursive la matrice de CAM en deux moitiés en minimisant le nombre de requêtes qui accèdent à des attributs communs aux deux moitiés. L'inconvénient majeur de cette approche est que la solution proposée ne contient que deux groupes d'attributs. Pour notre exemple le résultat de la procédure de fragmentation est illustré dans le tableau 18.

5.4.1.3 *Travaux de Navathe et Ra*

Navathe et Ra [157] ont modélisé le travail précédent en proposant un algorithme de fragmentation verticale utilisant une technique graphique. L'approche proposée construit la matrice d'affinité des attributs qui sera ensuite représentée par un graphe, appelé graphe d'affinité. Les fragments sont alors construits en fonction du poids des arêtes. Le graphe d'affinité, sert de base pour former des cycles, appelés

	a_5	a_1	a_7	a_2	a_8	a_3	a_9	a_{10}	a_4	a_6
a_5	75	75	50	25	25	25	25	0	0	0
a_1	75	75	50	25	25	25	25	0	0	0
a_7	50	50	85	60	60	25	60	0	0	0
a_2	25	25	60	110	110	75	75	0	0	0
a_8	25	25	60	110	110	75	75	0	0	0
a_3	25	25	25	75	75	115	115	15	15	15
a_9	25	25	25	75	75	115	115	15	15	15
a_{10}	0	0	0	0	0	15	15	40	40	40
a_4	0	0	0	0	0	15	15	40	40	40
a_6	0	0	0	0	0	15	15	40	40	40

TABLE 18: Fragmentation binaire de Navathe et al.

cycles d'affinité, qui définissent les fragments identifiés. Les Figures 26 et 27 illustrent respectivement le graphe d'affinité et les cycles d'affinité relatifs à l'exemple précédent.

5.4.1.4 *Travaux de Wesley et al.*

Wesley et al. [158] Proposent une autre technique pour la fragmentation verticale binaire dite Optimal Binary Partitioning algorithm (OBP). Elle construit les différents fragments selon une structure arborescente. Pour chaque requête, les attributs sont scindés en deux groupes : Ceux qui sont référencés par la requête et ceux qui ne le sont pas. L'algorithme produit enfin un arbre dont les feuilles contiennent les groupes d'attributs. Une fonction de coût est ensuite appliquée afin de déterminer le partitionnement binaire optimal lors de la fusion des feuilles adjacentes de l'arbre. Au même titre que les approches de partitionnement binaire, l'approche proposée ne produit que deux fragments d'attributs.

5.4.1.5 *Travaux de Agrawal et al.*

Agrawal et al. [159] Proposent une solution automatique pour la fragmentation verticale. Exploitant la matrice d'affinité, l'approche proposée identifie les ensembles d'attributs fréquents et retient les top-k ensembles ordonnés selon leurs supports. Chaque ensemble d'attributs retenu définit une partition binaire. L'algorithme détermine ensuite l'optimalité des partitions binaires formés pour chaque requête de la

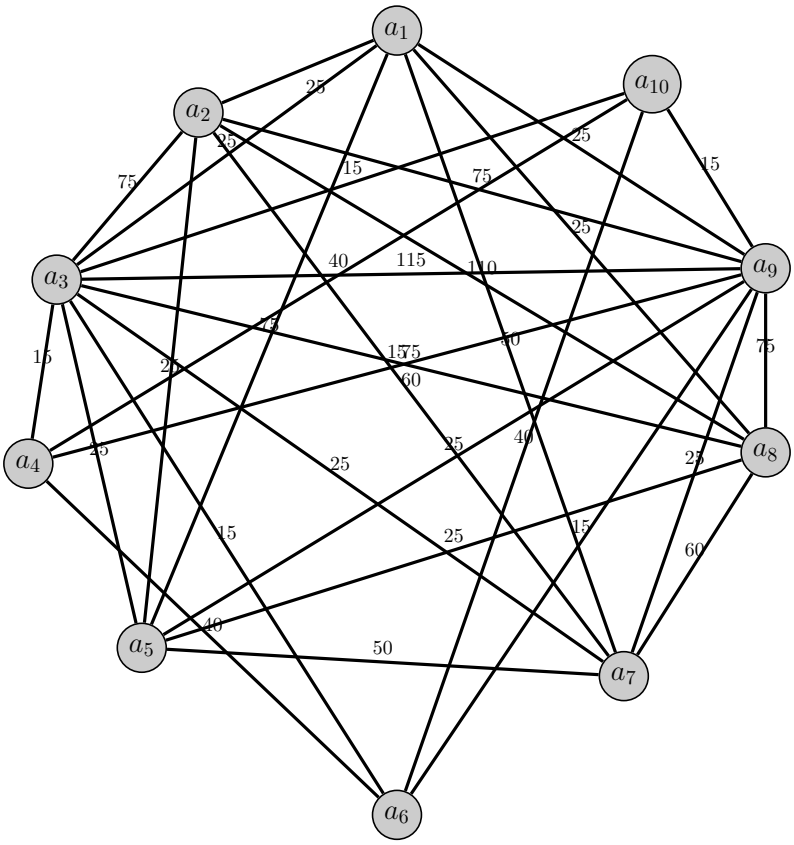


FIGURE 26: Exemple d'un graphe d'affinité

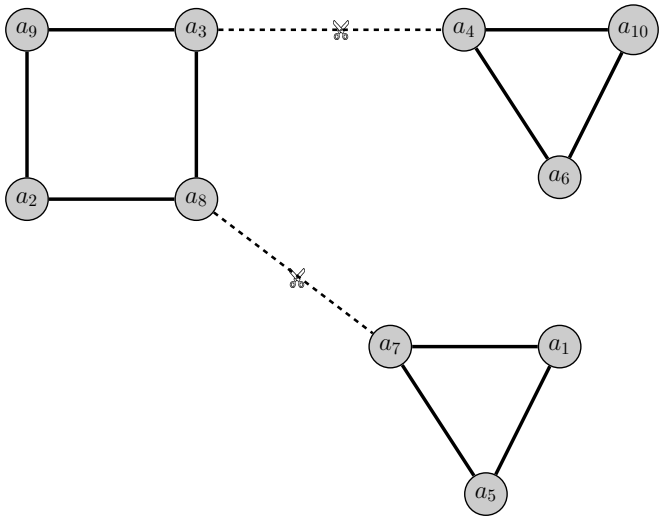


FIGURE 27: Exemple de cycles d'affinité

charge. La pertinence d'une partition pour une requête donnée est estimée par l'optimiseur du système. Une étape de fusion permet ensuite de combiner les groupes binaires résultant et d'évaluer le coût de toutes les partitions fusionnées possibles. Finalement, la partition fusionnée avec le meilleur coût est recommandée.

5.4.1.6 *Travaux de Gorla et al.*

Gorla et al. [30] adaptent l'algorithme Apriori pour proposer une approche de résolution du problème de la fragmentation verticale d'une table de données. L'approche fonctionne selon les étapes suivantes :

1. **Recherche des motifs fréquents :** En prenant en entrée une charge de requêtes $W = \{q_1^{f_1}, q_2^{f_2}, \ldots, q_n^{f_n}\}$ avec leurs fréquences respectives, un contexte d'extraction est formé à partir des attributs de la clause `SELECT` des requêtes. Pour un support minimum fixé σ, les motifs fréquents sont extraits à l'aide de l'algorithme Apriori. Le résultat de cette étape est ainsi un ensemble $\{M_1, M_2, \ldots, M_k\}$ de motifs fréquents.
2. **Génération des schémas de fragmentation candidats :** Dans cette étape les schémas de fragmentation candidats sont générés. L'approche commence par considérer le motif le plus large M_k qui désigne le premier fragment. Les motifs restants sont ensuite traités selon l'ordre décroissant de leur taille : $M_{k-1}, M_{k-2}, \ldots, M_1$ pour former d'éventuels fragments disjoints. Ce procédé conduit à un schéma de fragmentation F_1. Un traitement itératif similaire est appliqué à tous les motifs de taille k. Le résultat de cette étape est un ensemble de schémas de fragmentation candidats $\{F_1, F_2, \ldots, F_l\}$.
3. **Sélection du schéma optimal :** Un modèle de coût est utilisé pour évaluer les gains éventuels pour les différents schéma générés dans l'étape précédente. Le schéma présentant le meilleur gain est ainsi recommandé.

5.4.2 *Fragmentation verticale dans les entrepôts de données*

5.4.2.1 *Travaux de Datta et al.*

Datta et al. [160] proposent une technique de stockage et d'accès aux données pour améliorer les performances d'un entrepôt de données modélisé par un schéma en étoile sans créer des index. Ils exploitent la fragmentation verticale pour construire un index nommé `Cuio`. La technique proposée matérialise des fragments au lieu des index ce qui permet, selon les auteurs, un gain énorme en espace de stockage.

5.4.2.2 *Travaux de Golfarelli et al.*

Golfarelli et al. [147] étudient la fragmentation des vues matérialisées définies sur un entrepôt de données et non des tables elles-mêmes. La fragmentation proposée est basée sur une charge de requêtes et un modèle de coût. Les auteurs proposent deux fonctionnalités : d'une part la fragmentation d'une vue en plusieurs fragments et, d'autre part, l'opération inverse, à savoir l'unification en une seule vue de deux ou plusieurs vues ayant une clé commune. L'unification vise à réduire la redondance des vues. Les auteurs supposent que leur approche peut être bénéfique pour la distribution de l'entrepôt sur une architecture parallèle et proposent de combiner leur algorithme de fragmentation avec un algorithme d'allocation des fragments [123].

5.4.2.3 *Travaux de Ziyati et al.*

Ziyati et al. [145][161][162] traitent l'apport des algorithmes génétiques au problème de la fragmentation dans les entrepôts de données modélisés par un schéma en étoile. Le travail réalisé étudie la fragmentation verticale des tables proprement dites de l'entrepôt. L'approche proposée consiste à fragmenter verticalement les dimensions de l'entrepôt. Le problème est formalisé comme un problème d'optimisation avec contrainte (nombre de fragments). Pour le résoudre, les auteurs proposent un algorithme génétique qui utilise le même codage que l'algorithme proposé dans [163]. La méthode de la roulette est utilisée pour sélectionner les individus parents. Le type de croisement utilisé est celui d'un seul point pour la génération de la population suivante. Le processus de fragmentation est initialisé par une population diversifiée et aléatoire qui ne dépend pas des informations de l'entrepôt de données et ne prend aucune information qui lui permet d'avoir une idée sur la solution initiale. Les auteurs affirment que ceci offre plus de chances de bien cerner la recherche et de mieux s'approcher de la solution optimale. Cependant, ils indiquent dans la conclusion que l'une des manière pour améliorer leur approche est de proposer une population initiale mieux dépendante de la solution recherchée. Pour réaliser la fragmentation, les auteurs proposent à l'administrateur d'identifier lui même les paramètres relatifs d'une part à l'algorithme génétique (nombre de générations, taux de croisement, taux de mutation) et, d'autre part, les attributs de dimension participant à ce processus (appelés attributs de fragmentation à partir de la clause `Select` des requêtes). Les auteurs note également qu'il n'existe pas de paramètres qui soient adaptés à la résolution de tous les problèmes qui peuvent être posés à un algorithme génétique et que certaines valeurs qui sont souvent utilisées peuvent être de bons points de départ pour démarrer une recherche de solution. Les valeurs indiquées sont :

- Taille de la population : entre 30 et 50 individus

- Taux de crossover : entre 70% et 95%
- Taux de mutation : entre 0,5% et 1%.

5.5 SYNTHÈSE DU CHAPITRE

La fragmentation verticale est une technique d'optimisation de la conception physique d'une base de données. Elle permet d'optimiser les requêtes et de faciliter la gestion des données. Cependant, sélectionner un schéma de fragmentation optimisé n'est pas une tâche facile. Plusieurs approches de sélection ont été proposées. Nous avons passé en revue les différentes approches de résolution. Nous dressons ici une synthèse des travaux que nous avons présenté dans ce chapitre. Le tableau 19 recense ces travaux classés en fonction du contexte d'utilisation, de la technique de résolution utilisée, et de l'utilisation ou non d'un modèle de coût pour évaluer les fragments produits.

Dans le contexte des bases de données, la mesure de l'affinité a été largement utilisée pour résoudre le problème de la fragmentation verticale. Malgré la simplicité des approches basées sur la mesure d'affinité entre les attributs, elles présentent plusieurs limites. En effet, il est difficile de déterminer le nombre de fragments et les attributs liés à chacun d'eux. L'interprétation est subjective et est laissée à l'utilisateur final. Les améliorations apportées ne permettent, dans certains cas, de générer que deux fragments. Ce résultat est loin d'être réaliste. En pratique, et par souci de performance, les attributs peuvent être regroupés dans plus de deux fragments en particulier pour une table ayant beaucoup d'attributs et une charge composée de beaucoup de requêtes. Par ailleurs, aucune estimation du gain apporté par l'approche n'est proposée.

La mesure d'affinité utilisée est calculée par pair d'attributs. Elle ne permet pas donc d'estimer la co-utilisation d'un ensemble de k attributs ($k \geqslant 3$). Ceci complique le processus de comparaison entre les valeurs d'affinité de plus de deux attributs. La façon naturelle d'exprimer la co-utilisation de k attributs est de mesurer la fréquence d'utilisation des ensembles d'attributs de différents tailles $s (1 \leqslant s \leqslant k)$. Cette connaissance peut être extraite par le biais de la recherche des motifs fréquents (Cf. 2). C'est ainsi qu'une approche plus récente [30] a exploité cette technique pour fragmenter une table de données. Cependant le type des motifs utilisés est un handicap majeur. En effet, la dite approche génère un nombre important de fragments redondants. Ceci nécessite un traitement d'élagage complexe pour construire les fragments pertinents finaux. Ce constat a motivé la proposition d'une amélioration de cette approche qui sera présentée au chapitre suivant.

Travaux	Contexte		Approche			Modèle de coût
	BDD	EDD	Affinité	Métaheuristique	FDD	
Hoffer et Severance [153]	x		x			
Navathe et al. [156]	x		x			
Navathe et Ra [157]	x		x			
Wesley et al. [158]	x		x			x
Agrawal et al. [159]	x		x			x
Gorla et al. [30]	x				x	x
Datta et al. [160]		x	x			
Golfarelli et al. [147]		x	x			x
Ziyati et al. [145]		x		x		x

TABLE 19: Synthèse des travaux sur la fragmentation verticale

À notre connaissance, il n'y a que peu de travaux qui ont abordé le problème de la fragmentation verticale dans le contexte des entrepôts de données. Certains proposent d'appliquer une fragmentation pour créer un index et non un schéma de fragmentation optimisé. D'autres proposent de fragmenter des vues matérialisées pour améliorer leur pertinence. Le travail qui a étudié la fragmentation verticale des tables proprement dites est basée sur une approche évolutionnaire. Malgré le succès théorique des algorithmes génétiques, ils souffrent, comme indiqué par les auteurs, du problème de calibrage des paramètres, étape importante avant leur utilisation. Ce manque dans les approches proposées dans ce contexte nous a motivé pour proposer une nouvelle approche basée sur la classification automatique qui sera présenter dans le chapitre suivant.

6

FRAGMENTATION VERTICALE GUIDÉE PAR LA FOUILLE DE DONNÉES

Sommaire

6.1 INTRODUCTION

Dans ce chapitre, nous proposons des solutions guidées par la fouille de données pour le problème de la fragmentation verticale des données. Les approches proposées consistent à analyser, à l'aide de méthodes de fouille de données, un ensemble

de requêtes, afin d'en extraire des relations significatives aidant à la résolution de la problématique abordée. Nous allons étudier les deux cas de figures du problème : avec et sans spécification préalable du nombre maximal de fragments. Afin de montrer la forte adéquation de la fouille de données pour la résolution de ce problème, nous allons proposer, pour chacun des cas, une solution guidée par une technique différente de fouille de données.

Nous commençons ce chapitre par présenter les fondements de notre approche MaxPart pour la fragmentation verticale d'une table de données. Nous mettons l'accent en particulier sur l'intérêt de l'exploitation de la représentation condensée des motifs fréquents pour la résolution de ce problème. Nous enchaînons par présenter notre modèle de coût pour l'évaluation des schémas de fragmentation générés. Le reste de cette première partie décrit l'étude expérimentale réalisée pour la validation de nos propositions.

Dans la deuxième partie du chapitre, nous discutons le problème de la fragmentation verticale dans le contexte des entrepôts de données relationnels. Nous nous penchons en particulier sur la question de la sélection des tables à fragmenter et nous argumentons le choix retenu. Après une description des différents modes de fragmentation envisageables, notre intérêt se porte sur la classification automatique, qui constitue un concept pour la mise en place d'une méthodologie de résolution du problème étudié. L'approche de classification des requêtes que nous préconisons, vise la pertinence du processus de fragmentation à travers une exploration de la ressemblance des requêtes. A cet effet, nous mettons en œuvre une classification k-means sous Weka. Une étude expérimentale pour l'évaluation de l'approche proposée est présentée avant la synthèse du chapitre.

6.2 MAXPART : RETOUR DES MOTIFS FRÉQUENTS MAXIMAUX

6.2.1 *Motivation*

Intuitivement, l'efficacité d'une approche de fragmentation dépend essentiellement de la connaissance des vrais corrélations de l'usage des requêtes dans une charge de requêtes donnée. Ceci est le facteur clé pour la pertinence des fragments générés. De telles corrélations peuvent être identifiées par le biais de l'extraction des motifs fréquents (Cf. Chapitre 2). C'est ainsi qu'une proposition récente [30] mettant en œuvre de tels motifs, par adaptation de l'algorithme Apriori, a été proposée afin de résoudre le problème de la fragmentation verticale d'une table de données.

La technique utilisée se révèle particulièrement adaptée à résoudre ce problème. Cependant, elle nécessite des opérations coûteuses, et est par conséquent remise en

cause avec des charges conséquentes. D'autant que certaines représentations condensées des motifs fréquents peuvent, comme nous l'avons déjà mentionné dans le chapitre 4, apporter une performance prévisible, présentant un avantage intéressant en simplifiant le processus de génération des schémas de fragmentation. En d'autres termes, les contextes d'utilisation de la fouille de données étant variés, il existe des situations où l'extraction de tous les motifs fréquents est tout simplement inadaptée. Afin d'illustrer cette remarque considérons l'exemple suivant qui est extrait de [30].

Exemple 13. *Soit une table composée des attributs* a, b, c, d, e, f. *Considérons une charge formée de cinq requêtes* $q_1, \ldots, q_5$ *et leurs fréquences correspondantes(Table 20(a)). Pour un support minimum de 40%, les fragments candidats générés en utilisant respectivement tous les motifs fréquents (AllPart), et les motifs fréquents maximaux (MaxPart) sont illustrés dans la table 20(b). AllPart génère* 13 *fragments candidats alors que MaxPart ne génère que* 2 *candidats. En reprenant l'approche proposée dans [30], le processus de génération des schémas de fragmentation est réalisé comme suit : Allpart commence par considérer le fragment de longueur maximale* $\{a, d, f\}$ *comme étant le premier fragment. Elle parcourt ensuite l'ensemble des 2-Fragments et trouve que* $\{b, e\}$ *ne chevauche pas avec la partition courante. Le deuxième fragment est ainsi généré. L'attribut restant* $\{c\}$*, qui est non fréquent, forme le troisième fragment. Le déroulement de cette itération conduit au schéma de fragmentation* $[\{a, d, f\}, \{b, e\}, \{c\}]$. *D'une façon similaire, le deuxième 3-Fragment* $\{a, b, e\}$*, conduit au schéma de fragmentation* $[\{a, b, e\}, \{d, f\}, \{c\}]$. *L'approche MaxPart, de l'autre coté, nécessite seulement deux comparaisons entre les 3-Fragment* $\{a, d, f\}$ *et* $\{a, b, e\}$ *pour générer les mêmes schémas de fragmentation.*

	Requêtes	Fréq.
q_1	Select a,b,e	1
q_2	Select b,e	3
q_3	Select a,d,f	3
q_4	Insert	2
q_5	Insert	1

(a)

$\Longrightarrow$

	AllPart	MaxPart
1-Fragment	a, b, d, e, f	-
		-
2-Fragment	ab, ad, ae	-
	df, af, be	-
3-Fragment	adf, abe	adf, abe

(b)

TABLE 20: Exemple d'un ensemble de requêtes (a) et des fragments candidats (b)

L'exemple précédent montre à quel point le choix de la technique utilisée peut influer l'efficacité de l'approche de résolution. Il est clair que l'approche MaxPart

élague d'une manière significative l'espace des fragments candidats de façon à pouvoir réduire le surcoût de génération des schémas de fragmentation induit par l'approche Allpart. En effet, lors du traitement des fragments F_k, de taille maximale, l'approche AllPart nécessite plusieurs parcours pour les comparer avec les fragments $F_{k-1}, F_{k-2}, \ldots, F_1$. Cependant, ces derniers peuvent être tout simplement ignorés du fait qu'ils sont inclus dans les fragments de taille k. Evidement, l'éfficacité de MaxPart devient plus significative en utilisant des charges de travail de grandes tailles et comportant un nombre important d'attributs. D'un point de vue de faisabilité, nous pouvons remarquer qu'utiliser l'approche AllPart revient à considérer le pire des cas. Ce problème récurrent dans le domaine de l'application des motifs fréquents pose un réel challenge. Il faut arriver à identifier les corrélations intéressantes sans pour autant trop complexifier le processus de génération de la solution souhaitée. Si la qualité de la solution repose pour beaucoup sur les corrélations pertinentes entre les attributs, elle dépend également de la façon dont ces dernières sont identifiées.

6.2.2 *Évaluation des schémas de fragmentation : Un modèle de coût*

Afin de valider nos approches, nous sommes amenés à estimer les coûts de traitement des requêtes sur les différents schémas de fragmentation générés. Il ne s'agit pas toutefois de déterminer le meilleur plan d'exécution des requêtes. Cet objectif est trop ambitieux pour notre travail. En effet, l'optimisation des plans d'exécution des requêtes est un thème de recherche à part entière caractérisé par trois dimensions [123][164] (i) l'espace de recherche qui constitue l'ensemble des possibilités d'exécution d'une requête. (ii) la stratégie de recherche qui permet d'énumérer les plans d'exécution et choisir le plan optimal, (iii) le modèle de coût qui permet d'estimer le coût d'une exécution.

Dans notre travail, il s'agit plus simplement de pouvoir exprimer les coûts de traitement des requêtes par des estimations aussi simple que possible et de faciliter, par la suite, la validation expérimentale des solutions proposées.

Dans les SGBD, il est bien connu que le coût de traitement des requêtes est dominé par celui des jointures [165][166]. En effet, les autres opérations classiques telles que les sélections et les projections sont en général moins coûteuses [165]. Par conséquent, dans nos modèles nous nous concentrons sur les coûts associés principalement aux différentes jointures impliquées par les requêtes. Notons également que l'ordre des jointures n'est pas pris en compte. L'ordre des jointure, connu dans la littérature sous le nom de *Join Ordering Problem*, est un problème NP-Complet [167][168][169][170] et n'est pas traité dans le cadre de cette thèse.

Rappelons que le nombre des accès disque (les opérations d'E/S), et donc la quantité des données transférées, sont les paramètres les plus couramment utilisés pour évaluer le coût de traitement des requêtes sur une base de données [29][132][139][30].

Étant donnés une requête q et un schéma de fragmentation composé de k fragments $\mathcal{F} = \{F_1, F_2, \ldots, F_k\}$, nous proposons un modèle de coût théorique pour estimer le coût d'exécution de q sur $\mathcal{F}$. Pour faciliter les calculs, nous supposons que le coût de traitement de q sur $\mathcal{F}$ est la somme :

1. du coût des jointures entre les fragments nécessaires pour répondre à la requête q. Le cas idéal est obtenu si q accède à un seul fragment contenant exactement les attributs qu'elle référence évitant ainsi des jointures supplémentaires.
2. du coût de traitement de q sur le fragment résultant des éventuelles jointures réalisées dans l'étape 1.

Notre modèle de coût est basé sur des formules exprimant, d'une part, le coût de la jointure de deux tables X et Y [171] et, d'autre part, le nombre de bloc nécessaire pour traiter une requête q [172]. Le Tableau 21 résume les notations utilisées par notre modèle.

Notation	Désignation
$\|\|X\|\|$	Le nombre total des tuples dans la table X
$\|X\|$	Taille, en octets, des attributs de la table (ou du fragment) X
DBS	La taille d'un bloc en octets
B_X	Nombre de blocs nécessaire pour stocker la table (ou le fragment) X
B_q^F	Nombre de blocs accédés par la requête q dans le fragment F
t_q	Le nombre des tuples satisfaisant la requête q

TABLE 21: Notation utilisées dans le modèle de coût

Le nombre des E/S nécessaires pour réaliser la jointure entre deux tables X et Y est donnée par [171] :

$$C_{\bowtie} = 3 \times (B_X + B_Y) \tag{6.2.1}$$

où B_X et B_Y désignent les nombres de bloc [1] nécessaire pour stocker les tables X et Y respectivement. Pour un schéma de fragmentation donné, nous appliquons cette formule pour estimer les éventuelles jointures entre les fragments nécessaires pour répondre à une requête q.

Une fois les jointures réalisées, le coût de la seconde étape de traitement de la requête q est estimé par le nombre de blocs accédés. Soient T une table de n tuples et m le nombre de blocs nécessaires pour stocker T. Supposons que k tuples satisfont la requête q. Alors le nombre de bloc accédés pour traiter q est donné par [172] :

$$B_q^T = m \times \left(1 - \left(1 - \frac{1}{m}\right)^k\right) \tag{6.2.2}$$

Nous appliquons cette formule sur la portion de données obtenue par la jointure des fragments nécessaires pour répondre à une requête donnée q. Ainsi, le coût de traitement d'une requête sur un schéma de fragmentation est estimé comme suit :

1. **Évaluation du coût de jointure des fragments requis par q :** Soit T la table d'origine. Supposons que $F_q = \{F_q^1, F_q^2, \ldots, F_q^N\}$ est l'ensemble des fragments requis par q. La portion P_q^{Final} nécessaire pour répondre à q est obtenue par :

   $$P_q^{Final} = \bowtie_{i=2}^{N} \left(F_q^i, P_q^{i-1}\right)$$

   où $P_q^1 = F_q^1$ et P_q^i désigne la portion intermédiaire obtenue après la $i^{ème}$ jointure.

   En utilisant l'équation (6.2.1), le coût des jointures est alors :

   $$Cost_{\bowtie} = \sum_{i=2}^{N-1} 3 \times \left(B_{F_q^i} + B_{P_q^{i-1}}\right) \tag{6.2.4}$$

   où

   $$B_X = \frac{||T|| \times |X|}{DBS}$$

2. **Éstimation du nombre des bloc nécessaire dans le fragment résultant :** Soit $B_q^{P_q^{Final}}$ le nombre des bloc nécessaires pour répondre à q en utilisant le fragment final P_q^{Final}. En utilisant l'équation (6.2.2) nous avons :

   $$B_q^{P_q^{Final}} = \left[B_{P_q^{Final}} \left(1 - \left(1 - \frac{1}{B_{P_q^{Final}}}\right)^{t_q}\right)\right] \tag{6.2.5}$$

   où

   $$B_{P_q^{Final}} = \frac{||T|| \times |P_q^{Final}|}{DBS}$$

1. Un bloc (ou une page) est l'unité de stockage dans les SGBD. Elle désigne la quantité de données que le disque dur transfert de manière atomique en mémoire. Nous avons utilisé ici le terme bloc pour garder la même appellation que dans [30]

Par conséquent, le coût de traitement d'une requête q est :

$$Cost(q) = \left[\sum_{i=2}^{N-1} 3 \times \left(B_{F_q^i} + B_{P_q^{i-1}}\right)\right] + B_q^{P_q^{Final}} \tag{6.2.6}$$

D'oú :

$$\begin{aligned} Cost(q) &= \left[\sum_{i=2}^{N-1} 3 \times \left(\frac{||T|| \times \left(|F_q^i| + |P_q^{i-1}|\right)}{DBS}\right)\right] \\ &+ \left[\frac{||T|| \times |P_q^{Final}|}{DBS} \times \left(1 - \left(1 - \frac{1}{\frac{||T|| \times |P_q^{Final}|}{DBS}}\right)^{t_q}\right)\right] \end{aligned} \tag{6.2.7}$$

$$\begin{aligned} Cost(q) &= \frac{||T||}{DBS}\left[\left(\sum_{i=2}^{N-1} 3 \times \left(|F_q^i| + |P_q^{i-1}|\right)\right)\right. \\ &+ \left.\left(|P_q^{Final}| \times \left(1 - \left(1 - \frac{1}{\frac{||T|| \times |P_q^{Final}|}{DBS}}\right)^{t_q}\right)\right)\right] \end{aligned} \tag{6.2.8}$$

Finalement, le coût de traitement de toute la charge $\mathcal{W}$ est estimé par :

$$Cost(\mathcal{W}) = \sum_{q_i \in \mathcal{W}} (Cost(q_i) \times f_i)$$

6.2.3 *MaxtPart : Vue générale*

Cette section décrit l'approche MaxPart illustrée par l'algorithme 4. MaxPart prend en entrée (i) une charge de requêtes $\mathcal{W}$ et (ii) une valeur de seuil prédéfinie σ. Elle renvoie un schéma de partitionnement optimisé $\mathcal{F}_{opt}$ qui minimise le coût de la charge de travail $\mathcal{W}$. Les principales étapes de l'approche MaxPart sont :

1. **Construction du contexte d'extraction :** Étant donnée une charge de requêtes $\mathcal{W} = \{q_1^{f_1}, \ldots, q_n^{f_n}\}$, cette première phase consiste à construire le contexte d'extraction relatif à $\mathcal{W}$. Il est représenté par un fichier texte. Chaque ligne i correspondant à la requête q_i est dupliquée f_i fois. Le nombre de colonnes dans chaque ligne dépend du nombre d'attributs référencés par la requête en question. Le contexte ainsi construit est formé donc de $\sum_{i=1}^{n} f_i$ lignes. Au même titre que dans [30], les attributs candidats sont les attributs non clés contenus dans la clause `SELECT`. Pour les opérations `INSERT/DELETE`, tous les attributs non clés, de la table sont utilisés.

2. **Génération des fragments candidats :** Pour une valeur donnée du support minimum, nous générons l'ensemble des motifs fréquents maximaux. Chaque motif désigne un fragment candidat. Les fragments ainsi générés sont classés en ordre décroissant selon leurs tailles (nombre d'attributs) et leurs supports.
3. **Generation des différents schémas de fragmentation possibles :** Soit $\mathbb{F} = \{\mathbb{F}_k, \ldots, \mathbb{F}_1\}$ l'ensemble des fragments candidats générés à l'issue de l'étape précédente. Chaque classe $\mathbb{F}_i$ correspond aux fragments de taille i. Pour tous les $\mathcal{F}_j \in \mathbb{F}_k (1 \leqslant j \leqslant size(\mathbb{F}_k))$, nous construisons un schéma de fragmentation possible comme suit :
   a) Le motif $\mathcal{F}_j$ désigne le premier fragment.
   b) Les ensembles $\mathbb{F}_k, \mathbb{F}_{k-1}, \ldots, \mathbb{F}_1$ sont examinés successivement pour former d'éventuels fragments. Évidement, la priorité est donnée aux fragments de taille maximale et présentant la plus grande valeur du support. Si plusieurs cas se présentent, le choix est alors aléatoire.
   c) Finalement, les attributs non fréquents définissent le dernier fragment. Le résultat de cette étape est l'ensemble $\{P_1, P_2, \ldots, P_p\}$ des schémas de fragmentation possibles.
4. **Évaluation des schémas de fragmentation générés :** Les schémas de fragmentation générés à l'issue de l'étape précédente visent à associer les fragments pour les besoins des requêtes selon l'utilisation réelle des attributs. Afin d'évaluer la pertinence de cette démarche de fragmentation, nous utilisons le modèle de coût proposer dans la section 6.2.2. Le schéma de fragmentation présentant le coût minimal est recommandé.

6.2.4 *Exemple illustratif*

Nous présentons, dans cette section un exemple illustratif du fonctionnement de l'approche MaxPart. Reprenons la charge de l'exemple 13 et considérons les caractéristiques résumées dans la table 22. Supposons de plus que le nombre de tuples de la table T est $||T|| = 150$ et que la taille d'un bloc de données est de 100 octets.

1. **Construction du contexte d'extraction :** La Figure 28 illuste le contexte d'extraction pour notre exemple.
2. **Génération des fragments candidats :** Considérons, par exemple, une valeur de 40% pour le support minimum. Les fragments candidats générés sont alors $\{a, d, f\}(6)$ et $\{a, b, e\}(4)$. La valeur entre parenthèse représente la fréquence du fragment.

Algorithme 4 : MaxPart

Données :

– Une charge de requêtes $\mathbb{W} = \{q_1^{f_1}, q_2^{f_2}, ..., q_n^{f_n}\}$;

– Un support minimum σ ;

Résultat :

– $\mathcal{F}_{opt} = \{F_1, F_2, \ldots, F_k\}$: Un schéma de fragmentation optimisé ;

début

// Construction du contexte d'extraction $\mathbb{EC}$

$\mathbb{EC} \longleftarrow \emptyset$;

Row $\longleftarrow$ "";

pour *(Chaque* $q_i \in \mathbb{W}$*)* **faire**

Row $\longleftarrow$ Attributs_Fragmentation(q_i);

pour $j = 1 \rightarrow f_i$ **faire**

Writeln($\mathbb{EC}$, Row);

fin

fin

// Identification des fragments maximaux

$\mathbb{F}_{\mathcal{W}}[..] \leftarrow \text{FPMAX}(\mathcal{EC}, \sigma)$; // $\mathbb{F}_{\mathcal{W}}$: Ensemble des fragments générées

$\mathbb{F}_{\mathcal{W}}[..] \leftarrow \text{Sort}(\mathbb{F}_{\mathcal{W}}[..], length, frequency)$; // Tri des fragments dans $\mathbb{F}_{\mathcal{W}}$

// Génération des schémas possibles à partir de $\mathbb{F}_{\mathcal{W}}$

$\mathbb{PPS} \leftarrow \emptyset$; // $\mathbb{PPS}$ Ensemble des schémas possibles

pour $(i \leftarrow 1, size(\mathbb{F}_k))$ **faire**

$P \leftarrow \mathbb{F}_k[i]$;

$P_{temp}[..] \leftarrow \emptyset$;

pour $(p \neq P, p \in \mathbb{F}_k, \mathbb{F}_{k-1}, \ldots, \mathbb{F}_1)$ **faire**

$P_{temp}[..] \leftarrow p - (P \cap p)$;

// Construction des fragments non chauvauchants

fin

while $P_{temp}[..] \neq \emptyset$ **do**

$P_1 \leftarrow$ Fragment $\in P_{temp}[..]$ avec taille et support maximaux;

$P \leftarrow P \cup P_1$;

$P_{temp}[..] \leftarrow P_{temp}[..] - P_1$;

end

$\mathbb{PPS} \leftarrow \mathbb{PPS} \cup P$;

fin

// Identifier le meilleur schéma de fragmentation

$\mathcal{F}_{opt} \leftarrow \mathbb{PPS}[0]$;

pour $(i \leftarrow 1, size(\mathbb{PPS}))$ **faire**

si $\text{Cost}(\mathcal{W}, \mathcal{F}_{opt}) > \text{Cost}(\mathcal{W}, \mathbb{PPS}[i])$ **alors**

$\mathcal{F}_{opt} \leftarrow \mathbb{PPS}[i]$;

fin

fin

Retourner $\mathcal{F}_{opt}$;

Fin.;

Requêtes	Fréquence	# de tuples		Attributs	Taille (Octets)
q_1	1	2		a	1
q_2	3	60		b	4
q_3	3	20		c	8
q_4	2	10		d	2
q_5	1	8		e	1
				f	2

TABLE 22: Exemple de caractéristiques des requêtes et des attributs

a	b	e			
b	e				
b	e				
b	e				
a	d	f			
a	d	f			
a	d	f			
a	b	c	d	e	f
a	b	c	d	e	f
a	b	c	d	e	f

FIGURE 28: Contexte d'extraction de l'exemple 13

3. **Génération des schémas de fragmentation possibles :** Le motif maximal $\{a, d, f\}$, ayant la plus haute fréquence, est considéré comme le premier fragment. La vérification du motif $\{a, b, e\}$ permet de construire le fragment $\{b, e\}$ qui ne chevauche pas avec la partition existante. L'attribut restant $\{c\}$, qui est non fréquent, représente le dernier fragment. Le premier schéma de fragmentation est ainsi $\{[a, d, f], [b, e], [c]\}$. De la même façon, le second motif maximal $\{a, b, e\}$ conduit au schéma de fragmentation $\{[a, b, e], [d, f], [c]\}$.
4. **Évaluation des schémas de fragmentation :** Nous appliquons maintenant notre modèle de coût pour évaluer les deux schémas générés dans l'étape précédente.
   – **Schéma de fragmentation** $\{[a, d, f], [b, e], [c]\}$ **:** La requête q_1 référence les attributs a, b et e. La réponse à q_1 nécessite une jointure entre les frag-

ments $[a, d, f]$ et $[b, e]$ donnant lieu à la portion finale $[a, b, d, e, f]$. Nous avons $|abdef| = 10$. En utilisant l'équation (6.2.8), nous obtenons :

$$Cost(q_1) = \frac{150}{100}\left[(3 \times (5+5)) + \left(10 \times \left(1 - \left(1 - \frac{1}{\frac{150\times10}{100}}\right)^2\right)\right)\right]$$

$$Cost(q_1) = 46,95$$

La requête q_3, qui référence les attributs a, d et f, est traitée en utilisant le fragment $[a, d, f]$ seulement. Nous avons :

$$Cost(q_3) = \frac{150}{100}\left[\left(5 \times \left(1 - \left(1 - \frac{1}{\frac{150\times5}{100}}\right)^{20}\right)\right)\right] = 7,50$$

La requête q_2, qui ne nécessite pas de jointure, est traitée d'une manière similaire à q_3. Les requêtes restantes q_4 et q_5 nécessitant des jointures supplémentaires et sont évaluées de la même façon que q_1. Les différentes évaluations sont résumées dans le tableau 23.

Requêtes	Fréquence	Coût	Coût * Fréquence
q_1	1	46.95	46.95
q_2	3	00.46	01.38
q_3	3	07.50	22.50
q_4	2	134.37	268.74
q_5	1	133.02	133.02
Coût de la charge			472.59

TABLE 23: Évaluation du premier schéma de fragmentation

– **Schéma de fragmentation** $\{[a, b, e], [d, f], [c]\}$ **:** De la même manière, nous résumons les coûts obtenus dans la tableau 24. Le premier schéma de fragmentation est ainsi recommandé.

6.3 EVALUATION EXPÉRIMENTALE

6.3.1 *Tables et requêtes*

Notre approche constitue une amélioration du travail de Gorla et al. présenté dans [30]. Nous tirons profit des avantages des représentations condensées des mo-

Requêtes	Fréquence	Coût	Coût * Fréquence
q_1	1	01.89	01.89
q_2	3	08.99	26.97
q_3	3	56.25	168.75
q_4	2	134.37	268.74
q_5	1	133.02	133.02
Coût de la charge			599.37

TABLE 24: Évaluation du second schéma de fragmentation

tifs fréquents, en montrant les aspects liés à la complexité de la procédure de génération d'un schéma de fragmentation. Afin de montrer l'intérêt et l'efficacité de nos propositions, nous avons naturellement réalisé une étude expérimentale dans les mêmes conditions que dans [30]. Nous avons ainsi testé notre approche sur les mêmes tables et les mêmes charges. Les tables utilisées sont Teaching Assistant Evaluation (TAE) et ADULT disponibles sur le site de UCI [2] (Machine Learning Repository) [173]. Le tableau 25 présente les caractéristiques des deux tables.

Les charges de requêtes utilisées [30] sont formées de 12 et 20 requêtes pour les tables TAE et ADULT respectivement (voir annexe BB). Dans les deux cas, nous générons les schémas de fragmentations en utilisant les mêmes valeurs pour le support minimum à savoir 20%, 30%, 40%, 50% et 60%. Pour la génération des fragments maximaux, nous avons utilisé notre propre implémentation de l'algorithme Fpmax [114]. En exploitant la propriété de l'anti-monotonicité, nous en avons dérivé l'ensemble des fragments. Les expérimentations ont été réalisées sur une machine Intel(R) Xenon(TM) de 3.4 Ghz et 1024 Mb de mémoire sous Linux Ubuntu 10.04.

L'objectif principal de notre évaluation expérimentale est de montrer l'intérêt d'utiliser les motifs fréquents maximaux pour générer les schémas de fragmentation. Ainsi, notre étude comparative est quantifiée selon deux grandeurs : (1) le nombre des fragments candidats et (2) le nombre de combinaisons nécessaires pour générer un schéma de fragmentation possible. Nous étudions également la performance des schémas de fragmentation recommandés. Nous voulons démontrer l'intérêt de notre approche en ce qui concerne à la fois l'efficacité (qualité des schémas de fragmentation recommandés) et la complexité de traitement (nombre d'opérations nécessaires pour générer les schémas de fragmentation).

2. http ://archive.ics.uci.edu/ml/

Table	# des attribut	# de tuples	Attributs	
			Attribut	Taille (Octets)
TAE	6	161	a : Speaker	19
			b : Cours_instructor	2
			c : Course	2
			d : Semester	7
			e : Class_size	2
			f : Class_attribute	6
ADULT	15	30162	a : Age	1
			b : WorkClass	16
			c : Final_weight	4
			d : Education	12
			e : Education_num	1
			f : Marital_status	21
			g : Occupation	17
			h : Relationship	14
			i : Race	18
			j : Sex	6
			k : Capital_gain	3
			l : Capital_loss	2
			m : Hours_per_week	1
			n : Native_country	26
			q : Class	2

TABLE 25: Caractéristiques des tables TAE et ADULT

6.3.2 *Résultats et discussion*

Le tableau 26 résume les résultats relatifs au nombre des fragments candidats générés par les deux approches. Les résultats obtenus montrent que l'approche AllPart génère beaucoup trop de fragments candidats. Dans le cas de la table TAE, AllPart génère un nombre de fragments allant de 1.5 à 17 fois celui généré par MaxPart.

L'élagage, par notre approche, des fragments candidats est plus important pour la table ADULT. Les fragments générés par AllPart présentent un facteur allant de 3 à 1638 fois ceux générés par MaxPart.

TAE				ADULT		
Support(%)	AllPart	MaxPart	Réduction	AllPart	MaxPart	Réduction
20	51	3	94.11%	32767	20	99.94%
30	29	4	86.20%	269	17	93.68%
40	16	5	68.75%	21	7	66.66%
50	7	5	28.57%	5	5	00.00%
60	2	2	00.00%	3	3	00.00%

TABLE 26: Nombre des fragments candidats

Le tableau 27 illustre le nombre de combinaisons réalisées par les deux approches afin de produire les schémas de fragmentation possibles. Nous pouvons noter que l'approche AllPart génère des solutions trop coûteuses. Les résultats obtenus montrent que notre approche permet une réduction très significative du coût de traitement nécessaire pour la génération des différents schémas de fragmentation. Pour la table TAE, on dénombre par exemple pour un support de 20%, 100 combinaisons nécessaires pour AllPart, contre seulement 4 combinaisons nécessaires pour MaxPart, représentant ainsi une réduction d'environ 96.00%. Pour la table ADULT, la réduction est encore plus importante. Pour la même valeur du support, on dénombre 622554 combinaisons nécessaires pour AllPart, contre seulement 19 combinaisons nécessaires pour MaxPart, soit une réduction d'environ 99.99%. Ces résultats sont, évidement, une conséquence directe des stratégies adoptées par les deux approches pour la génération des fragments candidats. En effet, plus le nombre de fragments candidats est élevé plus le traitement des schémas de fragmentation est coûteux.

Comme nous l'avons démontré dans les sections précédentes, notre approche explore un espace de recherche très réduit pour proposer les mêmes schémas de fragmentation. Nous allons maintenant évaluer la performance des différents schémas générés par notre approche. Les coûts des charges de requêtes sont évalués en utilisant notre modèle de coût proposé dans la section 6.2.2. Les résultats obtenus sont présentés dans les tableaux 28 et 29 respectivement.

TAE				ADULT		
Suppor (%)	AllPar	MaxPar	Réduction	AllPart	MaxPart	Réduction
20	100	4	96.00%	622554	19	99.99%
30	56	6	89.29%	268	16	94.03%
40	30	8	73.33%	40	12	70.00%
50	6	4	33.33%	4	4	00.00%
60	2	2	00.00%	2	2	00.00%

TABLE 27: Nombre de combinaisons pour générer les schémas de fragmentation possibles

Support(%)	Schéma de fragmentation	Coût	Réduction
20	abdef - c	2448	00.32%
30	acdf - be	2171	11.60%
40	acf - be - d	2420	01.46%
50	af - b - c - d - e	2600	-05.86%
60	a - b - c - d - e - f	2720	-10.74%
Sans fragmentation	abcdef	2456	

TABLE 28: Évaluation des schémas de fragmentation (TAE)

Pour la table TAE, nous pouvons noter que le meilleur schéma de fragmentation est obtenu pour la valeur 30% du support. Pour cette valeur le coût de traitement de la charge est réduit d'environ 11.60%. Le meilleur schéma de fragmentation pour la table ADULT est obtenu pour la valeur 40% du support conduisant à un gain de coût d'environ 36.05%. Nous pouvons également remarquer que dans les deux cas, l'augmentation de la valeur du support génère un nombre plus important de petits fragments. Nous expliquons ce résultat par le fait que, pour de telles valeurs, il existe peu de fragments larges (peu de motifs fréquents maximaux). Dans un tel cas, les requêtes nécessitent des jointures supplémentaires conduisant à un coût élevé de la charge de travail.

Support (%)	Schéma de fragmentation	Coût	Réduction
20	abcdefghijklmno	454236	00.00%
30	abceikm - fo - dh - g - j - l - n	331520	27.01%
40	abo - efk - m - g - d - n - c - h - i - j - l	290451	36.05%
50	a - b - c - d - e - f - g - h - i - j - k - l - m - n - o	501283	-10.35%
60	a - b - c - d - e - f - g - h - i - j - k - l - m - n - o	501283	-10.35%
Sans frag.	abcdefghijklmno	454236	

TABLE 29: Évaluation des schémas de fragmentation (ADULT)

6.4 CLASSIFICATION AUTOMATIQUE : UNE SOLUTION POUR LA FRAGMENTATION VERTICALE DANS LES ENTREPÔTS DE DONNÉES

6.4.1 *Introduction*

Rappelons que les requêtes sur un entrepôt de données relationnel modélisé par un schéma en étoile, dites requêtes de jointures en étoile, sont caractérisées par des restrictions de sélection sur une ou plusieurs tables de dimensions suivies par des jointures avec la table des faits. Les restrictions sur les valeurs des dimensions sont utilisées pour la sélection de certaines mesures dans la table des faits. Ces dernières sont ensuite groupées et agrégées selon les demandes des utilisateurs. Le goulot d'étranglement majeur dans l'évaluation de ces requêtes est la jointure de la table des faits centrale (qui est généralement très volumineuse) avec les tables de dimension [174]. Aucune jointure n'est réalisée directement entre les tables de dimensions, qui sont généralement, de tailles plus petites [17], ce qui nécessite un nombre important d'opérations d'E/S et complique, par conséquence, l'évaluation des jointures. Cherchant à répondre à ce problème crucial et de remédier aux coûts prohibitifs imposés par les opérations de jointures, la fragmentation verticale semble, intuitivement, une solution bien adaptée. Elle permet, en effet, un bon regroupement physique des données afin de diminuer le volume des données manipulées et par conséquence le nombre des opérations d'E/S nécessaires à l'évaluation des jointures.

6.4.2 *Méthodes de fragmentation envisageables*

L'intuition principale derrière notre proposition est qu'une fragmentation optimisée permet de réduire considérablement la taille de tuples participant à une séquence de jointures entre les tables. Réaliser les jointures entre des données de tailles inférieures par rapport aux données d'origine permet certainement des gains significatifs sur les coûts totaux de traitement des requêtes. Cependant, l'adoption de la technique de fragmentation induit une question évidente mais importante concernant le choix du type de table à fragmenter : la table des faits, les tables de dimension ou les deux à la fois ?. Trois cas de figure se distinguent :

1. **Fragmentation des tables de dimension :** La fragmentation des tables de dimensions permet d'optimiser les opérations de sélection définies sur ces tables. En revanche, elle ne permet pas d'optimiser les opérations de jointures entre la table des faits et les tables de dimension. Étant donné que la table des faits est très volumineuse, ne pas la fragmenter ne réduit pas significativement le coût d'exécution des jointures qui représentent les opérations les plus coûteuses [175]. Par conséquence, toute fragmentation ne prenant pas en considération la table des faits est à exclure.
2. **Fragmentation de la table des faits :** Dans ce cas de figure, la table des faits est fragmentée verticalement. Lors du traitement des requêtes, l'opération de jointure avec les tables de dimensions n'implique que le fragment concerné. Ceci va réduire considérablement le taille des tuples de la table de faits participant à une séquence de jointures. Étant donné que la table des faits est très volumineuse, la fragmenter convenablement peut réduire le volume des données manipulées, en particulier lors des opérations de jointure, et par conséquence le nombre des opérations d'E/S.
3. **Fragmentation des toutes les tables :** Fragmenter l'ensemble des tables de l'entrepôt risque de faire exploser le nombre de fragments ce qui complique la tâche de la gestion et la mise en œuvre du schéma de fragmentation résultat. De plus, la table des faits et les tables de dimension sont fragmentées de manière indépendante car il ne s'agit pas ici d'une fragmentation dérivée comme dans le cas de la fragmentation horizontale. Ceci risque même d'augmenter le nombre des opérations de jointures et de pénaliser, par conséquence, le traitement des requêtes. En effet, ce schéma de fragmentation nécessite de définir des jointures entre les différentes fragments pour interroger les données des dimensions. Il nécessite également des jointures entre les fragments de la table des faits et finalement des jointures entre les portions à la fois des tables de dimensions et la table des faits impliquées par les requêtes. Par conséquence,

cette manière de procéder ne peut être envisageable et ne garantit pas l'amélioration des performances des requêtes de jointure en étoile. En effet, ce type de schéma augmente le nombre de jointures à réaliser dans l'exécution d'une requête

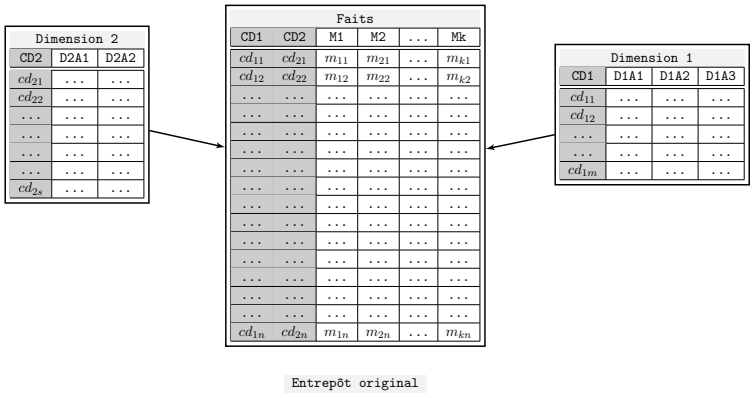
En résumé, nous pouvons dire que fragmenter seulement les tables de dimensions n'est pas un choix adapté pour optimiser les requêtes de jointures en étoile parce que, d'une part, les tailles des tables de dimensions sont généralement petites, et par conséquence le gain escompté de leur fragmentation ne sera pas significatif. D'autre part, elle ne permet pas d'optimiser les jointures, les opérations les plus coûteuses, avec la table des faits qui constitue le vrai goulot d'étranglement pour les requêtes. Fragmenter l'ensemble des tables de l'entrepôt risque de faire augmenter le nombre de jointures nécessaires pour le traitement des requêtes et rend ainsi la résolution du problème particulièrement complexe. Le deuxième cas convient plus à notre contexte. Il permet d'opter pour une solution cohérente d'un point de vue à la fois de la complexité de la fragmentation et l'efficacité du traitement des requêtes. La figure 29 schématise le mode de fragmentation adopté.

6.4.3 *Notre approche*

6.4.4 *Motivation*

Rappelons que la fragmentation verticale consiste à diviser une table en deux ou plusieurs fragments dont chacun est composé d'un sous-ensemble disjoint des attributs de la table d'origine (sauf pour les attributs clés pour d'éventuelles jointures). En considérant une charge de requêtes, l'objectif consiste à choisir un schéma de fragmentation minimisant le coût de traitement de la charge en assurant le minimum d'accès possibles aux fragments construits. Cependant, la sélection d'un schéma de fragmentation optimal est un problème NP-complet [28][29] ce qui motive l'utilisation de méthodes approchées.

Généralement, une charge de requêtes présente de fortes dépendances entre les requêtes [176]. Il est évident que la découverte de ces dépendances est importante dans le sens où ces dernière deviennent des hypothèses permettant une meilleure estimation d'un schéma de fragmentation optimisé. L'administrateur est souvent confronté à un manque de connaissances sur ce sujet. Pour lui, un choix optimisé constitue donc un processus difficile et fastidieux puisqu'il s'agit à la fois d'analyser les requêtes de la charge et d'identifier d'éventuelles correspondances pour enfin déduire une solution adéquate.



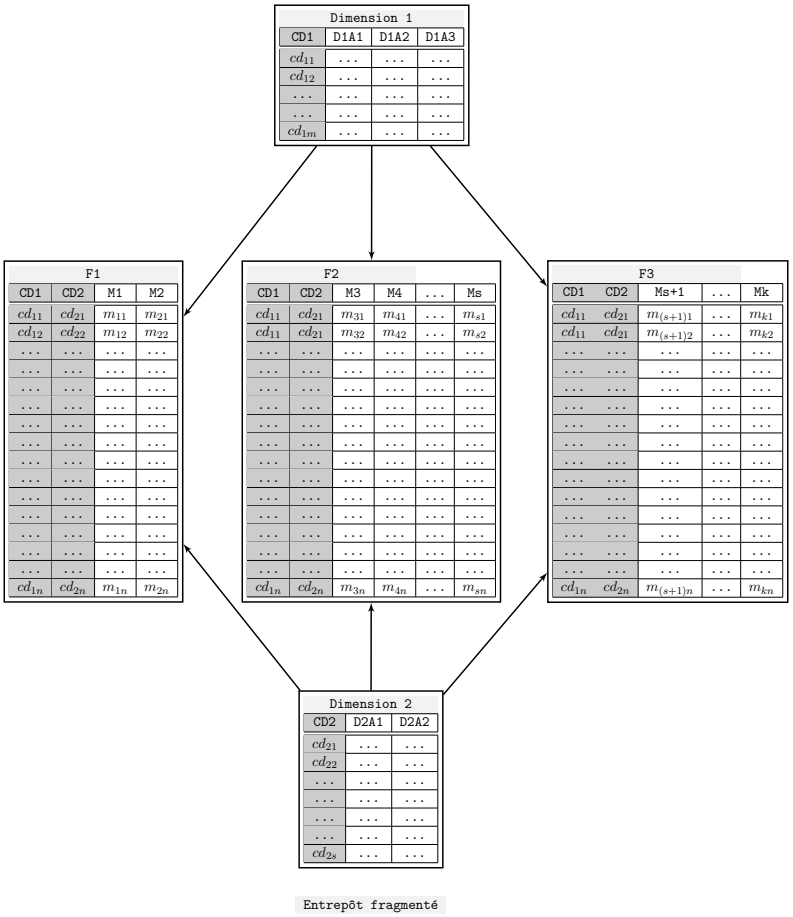


FIGURE 29: Mode de fragmentation adopté

Nous pouvons dire qu'une solution efficace au problème de la fragmentation doit organiser l'ensemble de toutes les requêtes en fonction de leurs caractéristiques pour en déduire des groupes de requêtes fortement corrélées. Une telle organisation permet certainement de générer les fragments qui correspondent le mieux aux attentes des requêtes. En effet, si un fragment est pertinent par rapport à une requête, alors il a plus de chance d'être également pertinent pour l'ensemble des requêtes similaires à la première. Sous l'angle de cette vision, le degré de pertinence d'un fragment relativement à un ensemble de requêtes est perçu comme le degré de corrélation entre les requêtes en question. Nous visons à proposer une solution adaptative qui prend en compte les caractéristiques communes aux requêtes dans le but d'améliorer l'efficacité du processus de fragmentation. Le but à atteindre étant d'obtenir des fragments (idéalement un fragment) conformes aux besoins d'un groupe donné de requêtes.

Ainsi, nous pouvons noter la similitude entre le problème de la fragmentation verticale et celui de la classification automatique. Cette dernière est, en effet, utilisée pour identifier, parmi un ensemble d'objets, des groupes ayant des propriétés similaires [177]. Inspiré par cette remarque, nous souhaitons résoudre la problématique étudiée par la classification automatique des requêtes. Ces dernières expriment les attentes des utilisateurs et contiennent, par conséquence, les informations nécessaires à leur compréhension. L'objectif étant de de fournir à l'administrateur une approche permettant l'analyse et l'extraction des correspondances, implicites et inconnues à priori, parmi les requêtes de la charge originale. La classification envisagée porte sur l'identification automatique d'éventuels groupes de requêtes présentant le plus grand taux de similitude possible.

Nous sommes convaincus que la classification automatique des requêtes constitue un cadre pratique particulièrement bien adapté pour modéliser le processus de fragmentation selon les attentes exprimées par les requêtes de la charge. Notons enfin que notre approche permet de contrôler le nombre de fragments générés. Le nombre des groupes identifiés correspond justement au nombre maximal des fragments souhaité par l'administrateur.

Afin d'apprécier la similitude des requêtes et d'avoir, en conséquence, une classification de qualité, il est important que la représentation des requêtes de la charge reflète au mieux leurs caractéristiques. Dans notre cas, il est nécessaire d'utiliser des descripteurs qui soient non seulement capables de caractériser une requête, mais qui tiennent compte également de l'importance de la requête vis à vis de la charge. Nous décrivons les requêtes par les deux principales caractéristiques suivantes :

1. **Les attributs référencés par la requête :** L'analyse de la requête permet de l'exprimer en termes des attributs référencés. On distingue ainsi deux catégo-

ries d'attributs : ceux qui sont référencés par la requête et ceux qui ne le sont pas. Dans une première étape, nous proposons d'associer à chaque requête ses différents attributs référencés en émettant l'hypothèse que si deux requêtes référencent en commun un maximum d'attributs, la cas échéant, tous les attributs, alors elles sont similaires. Intuitivement, deux requêtes sont dissimilaires si elles ne partagent aucun attribut. Plus les requêtes ont des attributs en communs plus leurs degrés de similitudes sont élevés. Les attributs utilisés pour représenter une requête dépendent évidemment de l'opération envisagée. A titre d'exemple, il s'agit des attributs non clés des tables de dimensions de la clause `Where` pour la sélection des index de jointure binaire [17][20][63](Cf. Chapitre 3), des attributs non clés de la clause `Select` [30][178] pour la fragmentation verticale et des prédicats de la clause `Where` pour la fragmentation horizontale [17][163].

2. **La fréquence de la requête :** Rappelons que la charge de travail est un ensemble de requêtes associées à leurs fréquences. Nous pouvons dire que la fréquence d'une requête correspond à une sorte de pondération exprimant l'importance de la requête vis à vis de la charge des requêtes. La prise en considération de ces deux paramètres n'est pas fortuite. En effet, elle combine les deux critères qui sont, d'une part, l'importance d'une requête pour une charge donnée (fréquence) et, d'autre part, le pouvoir de discrimination de cette même requête (par ses attributs). En considérant ces paramètres, notre approche prend donc en compte non seulement la structure d'une requête mais également sa fréquence d'exécution. Ainsi, les requêtes ayant une importance élevée et référençant un maximum d'attributs en commun sont considérées comme fortement corrélées (similaires).

6.4.5 *Description générale de l'approche*

Nous décrivons dans ce qui suit le mode de fonctionnement global de notre approche. Trois étapes sont essentielles à notre processus de fragmentation (Algorithme 5) : (i) la caractérisation des requêtes, (ii) la classification des requêtes et (iii) la génération d'un schéma de fragmentation optimisé.

1. **La caractérisation des requêtes :** Cette étape est cruciale pour le processus de fragmentation envisagé. Elle peut être vue comme une phase de préparation pendant laquelle les requêtes sont analysées afin d'en extraire les informations les plus discriminantes. Avant d'appliquer une méthode de classification sur les requêtes, il faut choisir un espace de représentation pour celles-ci. Dans un processus de classification automatique, le modèle de représentation le plus

utilisé est de décrire une population d'individus dans un espace $\mathbb{R}^d$ [179]. Autrement dit, chaque requête est définie par d variables, chacune de celles-ci prenant ses valeurs dans $\mathbb{R}$. A chaque requête, nous associons un vecteur de $d+1$ éléments où d est le nombre des attributs. Les cases de 1 à d sont mis à 1 ou 0 selon que la requête référence l'attribut en question ou non. Le contenu de la case $(d+1)$ corréspond à la fréquence de la requête. L'objectif est de modéliser les requêtes en fonction de leurs structures (référencement des attributs) et leurs poids (fréquences) par rapport à la charge de travail. Il convient de souligner que cette représentation des requêtes permet de transformer les données en entrée en une représentation numérique directement adaptée pour une utilisation avec des algorithmes de classification. La comparaison des requêtes est ainsi effectuée en comparant leurs vecteurs respectifs.

2. **La classification automatique des requêtes :** En considérant la caractérisation réalisée dans l'étape précédente, nous disposons des informations nécessaires pour permettre une classification de sorte que les groupes générés reflètent les éventuelles correspondances, implicites et inconnues à priori, parmi les requêtes de la charge originale. Le principe général des techniques de partitionnement est de minimiser la distance à l'intérieur des groupes tout en maximisant la distance entre les groupes identifiés. Les requêtes ne sont plus alors considérées de manière isolée, mais sont vues comme des groupes possédant des propriétés similaires. Afin de discriminer les requêtes de la charge, nous exploitons l'algorithme de classification le plus connue et le plus utilisé dans la littérature à savoir l'algorithme K-means. K-means est la méthode la plus classique et reste l'outil de classiffication le plus utilisé dans les applications scientifiques et industrielles [59][180][181]. Dans notre contexte, le nombre k de classes nécessaires pour le fonctionnement de K-means corréspond au nombre maximal de fragments souhaité par l'administrateur.

3. **Génération d'un schéma de fragmentation optimisé :** A l'issue de l'étape précédente, un ensemble de groupes de requêtes cohérentes est identifié. L'hypothèse sous-jacente que nous faisons est que les fragments, idéalement un seul, générés pour chacun des groupes sont plus adaptés aux requêtes de ce dernier. A cet effet, nous traitons les groupes identifiés de manière à favoriser les requêtes les plus fréquentes. Les groupes sont ainsi triés et traités selon l'ordre décroissant de la somme des fréquences des requêtes qu'ils contiennent. Nous générons un schéma de fragmentation optimisé comme suit :

    a) Trier les groupes en ordre décroissant des fréquences des requêtes qui les constituent. Si plusieurs cas se présentent, on choisit celui dont les re-

quêtes référencent le maximum d'attributs. Si plusieurs cas se présentent, le choix est alors aléatoire.

b) Un fragment est formé des attributs référencés par les requêtes du groupe en question. Initialisé par les attributs de la requête la plus fréquente dans le groupe, le fragment est ensuite enrichie par les attributs du reste des requêtes dans le groupe traitées selon l'ordre décroissant de leurs fréquences.

c) S'il reste encore des attributs non affectés aller à `(d)` sinon `FIN`

d) Traiter le groupe suivant et former un nouveau fragment de la même manière qu'en `(b)`. Aller à `(c)`.

Il est clair que notre approche génère ainsi un nombre de fragments inférieur ou égal à K. Il est possible, en effet, que tous les attributs soient affectés alors qu'on n'a pas encore traité tous les groupes générés par l'étape de classification.

6.4.6 *Exemple illustratif*

Les tableaux 30 et 31 illustrent respectivement un exemple des caractéristiques des requêtes et leurs représentations en vue du processus de classification.

Requêtes	Attributs	Fréquence
q_1	a, b, e	6
q_2	b, e	3
q_3	a, c, d, f	2
q_4	b, d, e	3
q_5	b, d, e, f	4

TABLE 30: Exemple des caractéristiques de requêtes

supposons que, pour une classification en 3 classes, les groupes générés sont $G_1 = \{q_1, q_2\}$, $G_2 = \{q_4, q_5\}$ et $G_3 = \{q_3\}$. Le schéma de fragmentation optimisé est généré comme suit :

1. On commence par G_1 car la somme des fréquences de ses requêtes est la plus élevée. Dans ce groupe, la requête q_1 est la plus fréquente. Par conséquence

	a	b	c	d	e	f	Fréq.
q_1	1	1	0	0	1	0	6
q_2	0	1	0	0	1	0	3
q_3	1	0	0	1	0	1	2
q_4	0	1	1	1	1	0	3
q_5	0	1	1	1	1	1	4

TABLE 31: Préparation de la classification des requêtes

$\{a, b, e\}$ initialise le premier fragment. En considérant la requête q_2, le fragment reste inchangé car $\{a, e\}$ est inclus dans $\{a, b, e\}$.

2. Nous considérons maintenant le groupe $G_2 = \{q_4, q_5\}$. La requête la plus fréquente q_5 conduit à la formation du fragment $\{d, f\}$ (les attributs b et e étant déjà affectés au premier fragment). La requête q_4, ne génère aucun changement car tous ses attributs sont déjà utilisés.
3. Le dernier groupe $G_3 = \{q_3\}$ conduit à la formation du fragment $\{c\}$. Le schéma de fragmentation recommandé est ainsi $\{a, b, e\}, \{d, f\}, \{c\}$

.

6.5 ÉTUDE EXPÉRIMENTALE

Afin d'évaluer la pertinence de notre approche, nous avons procédé à un ensemble d'expérimentations dont l'objectif est d'évaluer :

1. L'apport de notre approche pour l'amélioration des performances du système par un processus de fragmentation verticale de la table des faits. Cette évaluation est effectuée en utilisant des estimations comparatives des coûts cumulés des requêtes sans et avec le schéma de fragmentation recommandé par notre approche.
2. L'impact du paramètre du nombre de fragments sur les résultats obtenus.

6.5.1 *Benchmark et requêtes*

L'emploi de benchmark (bancs d'essais) est profitable pour la communauté scientifique afin d'évaluer l'impact de différents choix techniques (indexation, fragmentation, matérialisation de vues...) [182]. Nous avons utilisé l'entrepôt de données issu

Algorithme 5 : AutoFrag

Données :

– Une charge de requêtes $\mathbb{W} = \{q_1^{f_1}, q_2^{f_2}, ..., q_n^{f_n}\}$;

– Nombre k maximal de fragments ;

Résultat :

– $\mathbb{F}$: Un schéma de fragmentation optimisé ;

```
1  début
       // Caractérisation des requêtes
2      M[][] ⟵ Attributs_Référencés(q_i) + Fréquence(q_i);
       // Classification Kmeans des requêtes
3      G_Req.[..] ⟵ ∅ ;// Ensemble des groupes générés
4      G_Req.[..] ⟵ Kmeans(M[][], k);
       // Génération d'un schéma de fragmentation optimisé
5      Trier G_Req.[..];
6      F ← ∅;
7      pour (i ← 1, size(G_Req.[..]), i++) faire
8          si Encore des attributs à traiter alors
9              f ⟵ Fragment généré à partir de la requête la plus féquente;
10             F ⟵ F ∪ f;
11         fin
12         sinon
13             Exit ;
14         fin
15     fin
16     Retourner F;
17 fin
```

du banc d'essais TPC-H édité et mis à la disposition de la communauté scientifique par le TPC (Transaction Processing Council) [183]. L'entrepôt considéré est constitué d'une table des faits `Lineitem` et de sept tables de dimension : `Part`, `Supplier`, `Partsupp`, `Customer`, `Orders`, `Nation` et `Region`. La table 32 illustre les caractéristiques des tables de l'entrepôt. La charge de requêtes considérée est composée de 21 requêtes (voir annexe B). Chaque requête est caractérisée par sa fréquence d'utilisation ainsi que l'ensemble des attributs qu'elle référence. L'ensemble des attributs concernés par le processus de fragmentation est décrit dans la table 33.

Table	Nombre d'enregistrements
Lineitem	6 000 000
Part	200 000
Supplier	10 000
Partsupp	800 000
Customer	150 000
Orders	1 500 000
Nation	25
Region	5

TABLE 32: Caractéristiques des tables de l'entrepôt TPC-H

Code	Attribut	Type	taille (octets)
a	LineNumber	Entier	4
b	Quantity	Décimal	8
c	ExtendedPrice	Décimal	8
d	Discount	Décimal	8
e	Tax	Décimal	8
f	ReturnFlag	Fixed text	1
g	LineStatus	Fixed text	1
h	ShipDate	Date	7
i	CommiDate	Date	7
j	ReceipDate	Date	7
k	ShipInStruct	Fixed text	25
l	ShipMode	Fixed text	10
m	Comment	Variable text	44

TABLE 33: Caractéristiques des attributs de fragmentation

6.5.2 *Méthodologie d'évaluation*

Afin d'évaluer la pertinence des schémas de fragmentation recommandés, nous adaptons le modèle de coût proposé dans la section 6.2.2 à notre contexte d'étude. Étant donnés une requête q et un schéma de fragmentation de la table des faits composé de n fragments $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$, nous déterminons :

1. L'ensemble des fragments de faits, idéalement un seul fragment, concernés par la requête q. Un fragment $F_i (1 \leqslant i \leqslant n)$ est impliqué dans l'exécution de q, si et seulement si, q accède à au moins une colonne (un attribut) du fragment F_i.
2. Les différentes tables de dimensions impliquées par la requête en question.

Nous définissons le coût de traitement de q sur $\mathcal{F}$ comme étant la somme des deux coûts suivants :

1. `Coût 1:` Coût des éventuelles jointures entre les fragments de la table des faits nécessaires pour répondre à la requête q. Le cas idéal est obtenu si chaque requête accède à un seul fragment contenant exactement les attributs qu'elle référence évitant ainsi des jointures supplémentaires.
2. `Coût 2:` Coût des jointures entre le fragment de la table des faits résultant de l'étape précédente et chacune des tables de dimensions impliquées par la requête en question.

Comme nous l'avons précisé dans la section 6.2.2, notre objectif n'est pas de chercher le meilleur plan d'exécution des requêtes mais plutôt de proposer un mécanisme pour estimer le coût de traitement des requêtes afin d'évaluer nos propositions.

Rappelons que le nombre des E/S nécessaire pour réaliser la jointure entre deux tables X et Y est donnée par [171] :

$$C_{\bowtie} = 3 \times (B_X + B_Y) \tag{6.5.1}$$

où B_X et B_Y désignent les nombres de bloc nécessaire pour stocker les tables X et Y respectivement. Pour un schéma de fragmentation donné, nous appliquons cette formule pour estimer les différentes jointures nécessaires pour répondre à une requête q.

1. `Coût 1:` Ce coût est estimé de la même manière que celui de la section 6.2.2. Soit F la table des faits. Supposons que $F_q = \{F_q^1, F_q^2, \ldots, F_q^N\}$ est l'ensemble des fragments requis par q. La portion F_q^{Final} nécessaire pour répondre à q est obtenue par :

   $$F_q^{Final} = \bowtie_{i=2}^{N} \left(F_q^i, P_q^{i-1}\right)$$

   où $P_q^1 = F_q^1$ et P_q^i désigne la portion intermédiaire obtenue après la $i^{ème}$ jointure.

   En utilisant l'équation (6.5.1), le coût des jointures est alors :

   $$Cost_{\bowtie}^1 = \sum_{i=2}^{N} 3 \times \left(B_{F_q^i} + B_{P_q^{i-1}}\right) \tag{6.5.2}$$

où

$$B_X = \frac{||X|| \times |X|}{DBS}$$

Comme il s'agit de la fragmentation de la table des faits, nous avons :

$$B_X = \frac{||F|| \times |X|}{DBS}$$

$||F||$ étant le nombre de tuples dans F. $|X|$ désigne la longueur (la taille) de chaque tuple du fragment X et DBS désigne la taille d'un bloc.

$$Cost^1_{\bowtie} = \left[\sum_{i=2}^{N} 3 \times \left(\frac{||F|| \times \left(|F^i_q| + |P^{i-1}_q|\right)}{DBS}\right)\right] \tag{6.5.3}$$

et donc :

$$Cost^1_{\bowtie} = \frac{3 \times ||F||}{DBS}\left[\sum_{i=2}^{N}\left(|F^i_q| + |P^{i-1}_q|\right)\right] \tag{6.5.4}$$

2. `Coût 2:` Soit F^{Final}_q le fragment nécessaire pour répondre à q et $\{T_1, T_2, \ldots, T_d\}$ l'ensemble des tables de dimension impliquées par q.

$$Cost^2_{\bowtie} = 3 \times \left(B_{F^{Final}_q} + B_{T_1}\right) + \cdots + 3 \times \left(B_{F^{Final}_q} + B_{T_d}\right) \tag{6.5.4}$$

$$Cost^2_{\bowtie} = 3 \times \left(d \times B_{F^{Final}_q} + B_{T_1} + B_{T_2} + \cdots + B_{T_d}\right) \tag{6.5.5}$$

$$Cost^2_{\bowtie} = 3 \times \left(\frac{d \times ||F|| \times |F^{Final}_q|}{DBS} + \frac{||T_1|| \times |T_1|}{DBS} + \cdots + \frac{||T_d|| \times |T_d|}{DBS}\right) \tag{6.5.6}$$

$$Cost^2_{\bowtie} = \frac{3}{DBS} \times \left(d \times ||F|| \times |F^{Final}_q| + \sum_{i=1}^{d}\left(||T_i|| \times |T_i|\right)\right) \tag{6.5.7}$$

Finalement :

$$Coût(q) = Cost^1_{\bowtie} + Cost^2_{\bowtie} \tag{6.5.8}$$

$$Coût(q) = \frac{3 \times ||F||}{DBS}\left[\sum_{i=2}^{N}\left(|F^i_q| + |P^{i-1}_q|\right)\right] + \frac{3}{DBS} \times \left(d \times ||F|| \times |F^{Final}_q| + \sum_{i=1}^{d}\left(||T_i|| \times |T_i|\right)\right)$$

(6.5.9)

$$\text{Coût}(q) = \frac{3}{\text{DBS}} \left[||F|| \times \sum_{i=2}^{N} \left(|F_q^i| + |P_q^{i-1}| \right) + d \times ||F|| \times |F_q^{Final}| + \sum_{i=1}^{d} (||T_i|| \times |T_i|) \right] \tag{6.5.10}$$

$$\text{Coût}(q) = \frac{3}{\text{DBS}} \left[||F|| \times \left(\sum_{i=2}^{N} \left(|F_q^i| + |P_q^{i-1}| \right) + d \times |F_q^{Final}| \right) + \sum_{i=1}^{d} (||T_i|| \times |T_i|) \right] \tag{6.5.11}$$

Le coût d'une charge de n requêtes $\mathbb{W} = \{q_1^{f_1}, q_2^{f_2}, ..., q_n^{f_n}\}$ est ainsi estimé par :

$$\text{Coût}(\mathbb{W}) = \sum_{i=1}^{n} (\text{Coût}(q_i) \times f_i) \tag{6.5.12}$$

6.5.3 *Résultats et discussions*

Nous considérons que les valeurs du paramètre du nombre de fragments dépend d'avantage du nombre d'attributs concernés par le processus de fragmentation. En tenant compte de cette remarque, nous avons appliqué notre approche pour trois valeurs du nombre maximal k des fragments souhaités. Les valeurs retenues sont 2, 3 et 5. Pour chacune des valeurs considérées, nous avons réalisé une classification k-means avec l'outil weka[3]. Nous avons, ensuite, estimé le coût (pondéré par la fréquence) de traitement de chacune des requêtes de la charge ainsi que le coût total cumulé en utilisant le schéma de fragmentation recommandé.

1. **Pour k=2 :** Le schéma de fragmentation généré est `bcd - aefghijklm`. Les coûts relatifs à ce cas de figure sont résumés dans le tableau 34. On note que le coût total cumulé en utilisant le schéma de fragmentation recommandé est inférieur à celui obtenu sans aucune fragmentation. Plus précisément, on enregistre un gain d'environ 39, 59% sur le coût total de la charge. Ceci est du au nombre de requêtes optimisées qui est de 15 sur 21, soit un pourcentage de 71, 43%.
2. **Pour k=3 ;** Le schéma de fragmentation généré est `bcd - fg - aehijklm`. Les coûts relatifs à ce cas de figure sont résumés dans le tableau 35. Les résultats

3. http ://www.cs.waikato.ac.nz/ml/weka

K=2			
Schéma de Fragmentation généré: bcd - aefghijklm			
	Coût		
Requêtes	Sans Fragmentation	Avec Fragmentation	gain(%)
q_1	6300660,25	12365114,41	-96,25
q_2	24555328,27	5360012,49	+78,17
q_3	20690005,97	7242738,05	+64,99
q_4	2247803,11	4067139,36	-80,93
q_5	38400697,75	9607724,09	+74,98
q_6	6300660,25	2345581,45	+62,77
q_7	29807451,37	38904181,23	-30,51
q_8	34469469,59	8154976,97	+76,34
q_9	32122376,75	8128232,039	+74,69
q_{10}	3971564,77	1176642,42	+70,37
q_{11}	6670935,22	13341834,80	-99,99
q_{12}	8991212,47	7620117,18	+15,24
q_{13}	4851589,25	1687526,21	+65,21
q_{14}	6298829,2	2343750,40	+62,79
q_{15}	5890638,16	7709974,41	+64,99
q_{16}	25697901,40	30549464,73	-18,87
q_{17}	7558595,04	2812500,48	+62,79
q_{18}	8276002,38	2897095,22	+64,99
q_{19}	6298829,20	2343750,40	+62,79
q_{20}	7285670,10	1906762,93	+73,82
q_{21}	1215659,38	2428550,21	-99,77
Total	286149243,01	172849442,32	+39,59

TABLE 34: Coûts de la charge :K=2

obtenus montrent que la classification réalisée génère une solution significa-

tivement moins coûteuse avec un gain d'environ 45, 73%. Le nombre des requêtes optimisées est meilleur avec un taux de 17 sur 21 soit un pourcentage de 80.94%.

3. **Pour k=5**, Le shéma de fragmentation généré est `bcd - fg - aehijklm`. Nous retrouvons le même résultat que celui généré pour $k = 3$. Ceci est expliqué par le fait qu'au boût de traitement du troisième groupe, tous les attributs étaient affectés. Aucun fragment supplémentaire ne peut être généré.
4. Nous avons également analysé l'impact des nombres des fragments de la table des faits, noté `f`, sur l'optimisation des requêtes (Tableau 36). On peut observer que les requêtes qui nécessitent un seul fragment sont optimisées quelque soit le nombre des tables de dimensions, noté `d`, impliquées. D'un autre coté, les requêtes qui nécessitent l'ensemble des fragments générés sont non optimisées et leur évaluation devient plus coûteuse. Ceci est prévisible. En effet, dans cette situation la construction du fragment nécessaire au traitement d'une requête `q` revient tout simplement à la reconstruction de la table des faits originale par jointure de l'ensemble des fragments. C'est ce traitement qui induit un coût supplémentaire car l'évaluation de `q` sur un entrepôt non fragmenté revient à l'estimation des jointures entre la table des faits et les tables de dimension impliquées par `q`. Nous pouvons ainsi noter que l'optimisation du coût de la charge est fortement liée au nombre des fragments de la table des faits. Les résultats obtenus montrent qu'avec un choix judicieux du nombre de fragments, il est possible de garantir une bonne fragmentation.

6.6 SYNTHÈSE DU CHAPITRE

La fragmentation verticale est une technique d'optimisation permettant de réduire le coût de traitement des reqûetes en minimisant le volume des données manipulées. Cependant, la recherche d'un schéma de fragmentation optimal est un problème NP-complet ce qui motive le recours à l'utilisation des méthodes approchées. Dans ce chapitre, nous avons étudié les deux cas de figures du problème : avec et sans spécification préalable du nombre maximal de fragments. Afin de montrer la forte adéquation de la fouille de données pour la résolution de ce problème, nous avons proposé, pour chacun des cas, une solution guidée par une technique différente de fouille de données. Les techniques de fouille de données utilisées permettent, à partir des données du problème, d'obtenir les informations, inconnues à priori, utiles pour sa résolution.

Dans le premier cas de figure, nos propositions constituent une amélioration du travail, noté AllPart, présenté dans [30]. La technique d'extraction des motifs fré-

K=3 Schéma de Fragmentation généré: bcd - fg - aehijklm			
	Coût		
Requêtes	Sans Fragmentation	Avec Fragmentation	gain(%)
q_1	6300660,25	3713379,54	+41,06
q_2	24555328,27	5360012,49	+78,17
q_3	20690005,97	7242738,05	+64,99
q_4	2247803,11	4014404,98	-78,59
q_5	38400697,75	9607724,09	+74,98
q_6	6300660,25	2345581,45	+62,77
q_7	29807451,37	38376837,39	-28,74
q_8	34469469,59	8154976,97	+76,34
q_9	32122376,75	8128232,039	+74,69
q_{10}	3971564,77	1176642,42	+70,37
q_{11}	6670935,22	2664931,12	+60,05
q_{12}	8991212,47	7725585,84	+14,07
q_{13}	4851589,25	1687526,21	+65,21
q_{14}	6298829,2	2343750,40	+62,79
q_{15}	5890638,16	30549464,73	-18,87
q_{16}	25697901,40	23922484,55	+06,90
q_{17}	7558595,04	2812500,48	+62,79
q_{18}	8276002,38	2897095,22	+64,99
q_{19}	6298829,20	2343750,40	+62,79
q_{20}	7285670,10	1906762,93	+73,82
q_{21}	1215659,38	2758667,46	-126,92
Total	286149243,01	155279434,67	+45,73

TABLE 35: Coûts de la charge :K=3

quents utilisée dans ce travail est particulièrement trop coûteuse en générant un

K=2				K=3			
f	Requêtes	d	Gain(%)	f	Requêtes	d	Gain(%)
1	q_2	5	+78,17	1	q_2	5	+78,17
	q_3	2	+64,99		q_3	2	+64,99
	q_5	5	+74,98		q_5	5	+74,98
	q_6	1	+62,77		q_6	1	+72,77
	q_8	6	+76,34		q_8	6	+76,34
	q_9	5	+74,69		q_9	5	+74,69
	q_{10}	3	+70,37		q_{10}	3	+70,37
	q_{12}	1	+15,24		q_{12}	1	+14,07
	q_{13}	1	+65,21		q_{13}	1	+65,21
	q_{14}	1	+62,79		q_{14}	1	+62,79
	q_{15}	2	+64,99		q_{16}	5	+06,90
	q_{17}	1	+62,79		q_{17}	1	+62,79
	q_{18}	2	+64,99		q_{18}	2	+64,99
	q_{19}	1	+62,79		q_{19}	1	+62,79
	q_{20}	2	+73,82		q_{20}	2	+73,82
2	q_1	1	-96,25	2	q_1	1	+41,06
	q_4	1	-80,93		q_4	1	-78,59
	q_7	3	-30,51		q_7	3	-28,74
	q_{11}	1	-99,99		q_9	5	+74,69
	q_{16}	5	-18,87		q_{11}	1	+60,05
	q_{21}	1	-99,77				
				3	q_{15}	2	-18,87
					q_{21}	1	-126,92

TABLE 36: Impact du nombre de fragments de la table des faits

nombre important de fragments candidats équivalents (formés des mêmes attributs). D'autant que certaines représentations condensées des motifs fréquents peuvent, comme nous l'avons déjà mentionné dans le chapitre 4, apporter une performance prévisible, présentant un avantage intéressant en simplifiant le processus de génération des schémas de fragmentation. Inspiré par cette remarque, nous avons proposé une approche, dite MaxPart, guidée plutôt par l'extraction des motifs fréquents

maximaux. A travers un exemple pratique, nous avons motivé notre orientation de résolution. Nous avons plus particulièrement démontré que la complexité du processus de génération des schémas de fragmentation candidats sera largement simplifiée, tout en assurant une qualité égale de performance. En d'autres termes, nous avons démontré que notre approche explore un espace de recherche très réduit pour proposer les mêmes schémas de fragmentation. Nous avons également proposé un modèle de coût pour estimer la pertinence des schémas de fragmentation générés.

Afin de montrer l'intérêt de notre approche, en ce qui concerne à la fois l'efficacité (qualité des schémas de fragmentation recommandés) et la complexité de traitement (nombre d'opérations nécessaires pour générer les schémas de fragmentation), nous avons naturellement réalisé une étude expérimentale dans les mêmes conditions que dans le travail cité. Nous avons ainsi testé notre approche sur les mêmes tables et les mêmes charges de requêtes. Les tables utilisée sont Teaching Assistant Evaluation (TAE) et ADULT disponibles sur le site de UCI (Machine Learning Repository). L'étude expérimentale réalisée confirme nos propositions et montrent l'intérêt de l'approche proposée.

Dans le deuxième cas de figure, nous avons abordé la problématique étudiée dans le contexte des entrepôts de données. Nous avons noté en particulier l'analogie entre le problème de la fragmentation verticale et celui de la classification automatique. Cette dernière a l'avantage d'exprimer les appartenances des requêtes à des classes homogènes, ce qui permet de mieux connaître les références réelles des attributs et offre ainsi un éclairage intéressant pouvant aider au processus de la fragmentation. Dans notre contexte, la technique de classification est d'intérêt important pour deux raisons. Tout d'abord, elle est en mesure de mettre en correspondance les fragments générés et les besoins des requêtes utilisateurs afin de recommander les fragments en rapport avec ces besoins. Deuxièmement, elle permet de contrôler le nombre maximal de fragments générés.

Les résultats expérimentaux sur le benchmark TPC-H montrent que notre approche génère une solution significativement moins coûteuse. Nous avons noter en particulier que le coût de la charge dépend du nombre des fragments de la table des faits. Aussi, nous pouvons dire que la performance du processus de la fragmentation est fortement liée à la bonne sélection du nombre de fragments qui doit être préalablement fixé par l'administrateur.

,

7

BILAN ET PERSPECTIVES

7.1 BILAN

Les entrepôts de données tiennent actuellement une place prépondérante dans un monde où le volume des données stockées est en évolution croissante. Malgré la popularité et la prolifération des entrepôts de données, il reste un problème fondamental auquel ils sont confrontés : leur performance. En effet, les requêtes appliquées sur un entrepôt ont tendance à être complexes, ce qui nécessite souvent des opérations de calcul coûteuses tels que les jointures et les agrégations. Pour un schéma logique donné, les requêtes aboutissent toujours au même résultat quel que soit le schéma physique mis en place. Cependant, le coût d'une requête peut varier de plusieurs ordres de grandeur entre différentes implémentations physiques du même schéma logique. Par conséquence, le choix de la conception physique à mettre en place a un impact important sur les performances de l'entrepôt. La conception physique de l'entrepôt consiste principalement à la sélection des structures d'optimisation (index, fragments, vues matérialisées,... etc.) appropriées susceptibles d'améliorer les performances du système en minimisant les temps nécessaires à l'évaluation des requêtes.

La problématique à laquelle nous nous sommes intéressé dans cette thèse porte sur l'indexation et la fragmentation automatique des entrepôts de données relationnels modélisés par un schéma en étoile. Plus précisément nous avons apporté des solutions guidées par la fouille de données pour la résolution des problèmes abordés. Cette orientation de recherche n'est pas fortuite. La fouille de données offre dans un premier temps, un support mathématique solide pour la modélisation des problèmes posés. Ensuite, en termes de résolution, nous pouvons profiter des progrès dans ce domaine très actif de recherche.

Nous avons commencé par présenter les notions de base sur la fouille de données et plus particulièrement celles relatives à la recherche des motifs fréquents et à la classification automatique par partitionnement (Cf. Chapitre 2). Cette présentation a permis de montrer le caractère générique et l'intérêt pratique des techniques exposées, ce qui nous a motivé pour envisager leur utilisation aussi bien pour la modélisation que pour la résolution des problématiques étudiées.

Le premier axe de notre étude a porté sur la proposition d'une nouvelle approche pour le problème de sélection des index avec contrainte de stockage. Nous avons mis l'accent sur la complexité du problème étudié et la variété des approches de résolution proposées (Cf. Chapitre 3). Nous avons regroupé les approches proposées en fonction de la méthode de résolution adoptée. Tout d'abord les méthodes basées sur des heuristiques gloutonnes. Ensuite, les approches basées sur des métaheuristiques. Nous avons fini par nous intéresser, plus particulièrement, aux méthodes utilisant la fouille de données car nos propres propositions sont fondées sur les limites de ces dernières. Nous avons également souligné que la qualité d'une approche de résolution doit tenir compte de trois critères que nous considérons comme indicateurs de l'efficacité d'une approche de résolution du problème. La motivation principale derrière cette proposition est notre conviction que si la qualité d'une solution repose pour beaucoup sur le gain de coût réalisé, elle dépend également de la façon dont elle est identifiée. En effet, la simplicité de résolution d'un problème aussi complexe est un objectif à atteindre.

La solution proposée permet de construire un cadre de formulation du problème de sélection des index en exploitant le concept des motifs fréquents maximaux. Nous avons, en particulier, mis en évidence les limitations principales des approches basées sur l'extraction des motifs fréquents fermés. Soumises aux critères de performance et simplicité, les représentations condensées via les motifs fréquents maximaux sont plus intéressantes et moins coûteuses. Notre objectif était double : d'un coté minimiser le nombre des index générés et d'autre part conserver la pertinence des ces index.

Conceptuellement, une requête donnée de la charge est considérée comme une transaction alors que les attributs référencés par la dite requête définissent les items de la transaction. En terme de résolution, l'approche proposée permet un élagage judicieux de l'espace de recherche en évitant complètement la génération dupliquée des index. Elle permet de réduire autant que possible le nombre des index maintenus sans pour autant sacrifier leur qualité. La réduction du nombre d'index pertinents est une direction intéressante pour minimiser par la même voie l'espace de stockage requis. Nous avons montré formellement puis expérimentalement l'intérêt de notre approche. Nous avons en particulier prouvé que, quelque soit la limite de l'espace de stockage réservé aux index, notre approche nécessite un espace de stockage inférieur ou, au pire des cas, égal à celui exigé par les approches fondées sur l'extraction des motifs fréquents fermés. Ceci est particulièrement très intéressant quand l'administrateur ne dispose pas d'assez d'espace à allouer aux index.

Nous avons proposé également un contexte d'extraction étendu dans lequel la modélisation des requêtes a été révisée et enrichie en fonction des fréquences relatives

aux requêtes. Le nouveau contexte consiste à dupliquer la référence à une requête autant de fois que sa fréquence. Nous avons montré qu'on peut effectuer des recommandations avec une meilleure précision lorsque les fréquences des requêtes sont prises en compte. En effet, l'exploitation des fréquences de requêtes permet de promouvoir la sélection des attributs référencés par les requêtes les plus fréquentes.

Une autre caractéristique intéressante de notre approche concerne la mesure d'évaluation proposée. L'hypothèse faite est que l'utilisation seule du coût de la charge pour la recommandation d'une configuration finale est insuffisante. Elle ne permet pas forcément de trouver une solution satisfaisante pour l'administrateur. En considérant la configuration conduisant au coût minimal de la charge comme référence, notre point de vue est qu'il est possible qu'une autre configuration soit suffisamment proche de la première sans pour autant être exigeante en espace de stockage. D'un point de vue pratique, cette dernière configuration est plus intéressante pour l'administrateur. Ainsi, au lieu de se concentrer sur une seule configuration, nous suggérons l'idée de considérer plusieurs configurations pertinentes. Une mesure de profit exprimant le compromis entre le coût relatif à une configuration et l'espace de stockage induit par cette dernière permet de mieux appréhender l'intérêt des solutions retenues.

Nous avons implémenté un algorithme de recherche des motifs fréquents maximaux, en l'occurrence FPMAX. Dans le but de proposer un outil qui permet d'évaluer l'apport du cadre théorique proposé, nous avons également développé une application pour l'évaluation des solutions retenues. Une partie est dédiée à la manipulation des données. Une deuxième est dédiée à la visualisation des résultats obtenus. La partie fouille permet de mettre en œuvre le module de découverte des motifs à partir des contextes d'extraction fournis. Une interface utilisateur permet la sélection du contexte d'extraction et les paramètres relatifs à l'exploration de ce dernier.

Dans la deuxième partie de cette thèse, nous nous sommes intéressés au problème de la fragmentation verticale dans les bases de données (Cf. Chapitre 5). Le problème étudié consiste à diviser une table en un nombre de fragments verticaux de manière à réduire le coût de traitement des requêtes en minimisant le volume des données manipulées. Dans cette partie de la thèse, nous avons étudié les deux cas de figures du problème : avec et sans spécification préalable du nombre maximal de fragments. Afin de montrer la forte adéquation de la fouille de données pour la résolution de ce problème, nous avons proposé, pour chacun des cas, une solution guidée par une technique différente de fouille de données (Cf. Chapitre 6).

Dans un premier temps, nous avons proposé une amélioration du travail de Gorla et al. [30] pour la fragmentation verticale d'une table de données. A travers un exemple pratique, nous avons motivé notre orientation de résolution en mettant l'ac-

cent sur le fait que l'extraction de tous les motifs fréquents est tout simplement inadaptée à la résolution du problème étudié. Nous avons plus particulièrement démontré que la complexité du processus de génération des schémas de fragmentation candidats pourra être largement simplifiée, tout en assurant une qualité égale de performance. En d'autres termes, nous avons démontré que notre approche explore un espace de recherche très réduit pour proposer les mêmes schémas de fragmentation.

Ensuite, nous avons étudié le problème de la fragmentation verticale dans le contexte des entrepôts de données relationnels. Notre motivation est d'aborder un problème encore peu étudié dans la littérature. Nous nous sommes penchés en particulier sur la question de la sélection des tables à fragmenter : la table des faits, les tables de dimension ou les deux à la fois ?. Nous avons jugé que la fragmentation de la table des faits convient plus à notre contexte. Elle permet d'opter pour une solution cohérente d'un point de vue à la fois de la complexité du processus de la fragmentation et de l'efficacité du traitement des requêtes. En effet, d'une part, la fragmentation des seules tables de dimensions n'est pas un choix adapté pour optimiser les requêtes de jointures en étoile dont le goulot d'étranglement majeur est la jointure de la table des faits centrale (qui est généralement très volumineuse) avec les tables de dimension. D'autre part la fragmentation de l'ensemble des tables de l'entrepôt risque de faire exploser le nombre de jointures nécessaires pour le traitement des requêtes et rend ainsi la résolution du problème particulièrement complexe.

Nous avons ainsi proposé une approche qui permet la fragmentation automatique de la table des faits d'un entrepôts de données modélisé par un schéma en étoile. L'approche proposée vise une fragmentation adaptative aux besoins des requêtes utilisateurs en se basant sur une classification automatique de ces dernières. La classification envisagée fournit une typologie des requêtes en groupes homogènes et bien séparées. Les groupes de requêtes identifiés permettent de recommander un schéma de fragmentation pertinent.

Dans les deux cas de figures étudiés, nous avons proposé des modèles de coût pour évaluer la pertinence des schémas de fragmentation générés. Il s'agit de formules mathématiques exprimant les coûts de traitement des requêtes par des estimations aussi simple que possible afin de faciliter, par la suite, la validation expérimentale des solutions proposées.

7.2 PERSPECTIVES

Il ressort principalement du travail présenté dans ce manuscrit, que la fouille de données trouvent un champ d'application adéquat dans le domaine de l'automatisation de la conception physique des entrepôts de données. Permettant de construire

des solutions adaptées, les techniques de fouille de données sont très avantageuses pour réduire la complexité d'exploration de l'espace de recherche des solutions d'une part, et la construction d'une solution optimisée d'autre part. La mise en œuvre et l'évaluation expérimentale des approches proposées sur des benchmarks de référence montrent qu'elles répondent aux problématiques abordées et apportent des solutions de bonnes qualités. Différentes pistes ont émergé du travail effectué :

1. Les techniques d'optimisation étudiées ont été évaluées de manière isolée. Dès lors, il serait intéressant d'étudier à quel point la conjugaison des deux techniques d'optimisation peut améliorer les performances du système. On peut imaginer une sélection des index sur un entrepôt déjà fragmenté.
2. Le cadre d'extraction des motifs que nous avons proposé pour le problème de sélection d'index peut être naturellement étendu dans deux directions : (i) améliorer la mesure de corrélation entre les attributs indexables en exploitant en particulier les règles d'association, (ii) fournir un contexte de résolution plus adapté en alliant l'expertise humaine de l'administrateur aux modèles mathématiques de la fouille de données. Ce dernier point consiste à intégrer un module de spécification de contraintes imposées par l'administrateur sur les index recherchés, (connu dans la littérature par le terme *recherche de motifs sous contraintes*). Notons que des travaux ont déjà été réalisés dans ces deux directions [184][185].
3. Concernant le problème de la fragmentation verticale, l'une des perspectives est la définition de notre propre mesure de similarité entre les requêtes. Nous pensons que cette orientation apportera une valeur ajoutée pour le processus de classification des requêtes. Il serait également intéressant de mettre notre approche à l'épreuve dans le cas des données distribuées. Dans ce cas, les coûts de communication entre les nœuds du réseau sont un facteur qui doit être intégré dans le modèle de coût.
4. Une autre piste envisageable concerne l'analyse des charges de requêtes en tenant compte de leur aspect dynamique. La pertinence de l'information analysée est complètement perdue si l'on ne tient pas compte de cet aspect. Une solution, pour le problème de sélection d'index par exemple, consiste à adapter notre approche cette fois à une recherche incrémentale des motifs fréquents. Le problème de la recherche incrémentale des motifs fréquents concerne l'extraction des informations issues d'une base de transactions qui évolue dans le temps [186].
5. Une perspective applicative consiste à étendre l'application développée de sorte à pouvoir intégrer d'autres fonctionnalités. L'objectif est de réaliser une

sorte de boite à outils permettant des tests rapides sur une stratégie préalablement choisie.

6. Même si nos propositions ont été appliquées à la sélection d'index et à la fragmentation verticale, nous avons de bonnes raisons de penser qu'elles peuvent être facilement transposées à tout problème équivalent : identification des objets fréquents sans redondance en présence de contraintes. Nous pensons en particulier au problème de sélection des vues matérialisées.

A

ANNEXE A : ALGORITHME FPMAX POUR LA RECHERCHE DES MOTIFS FRÉQUENTS MAXIMAUX

A.1 INTRODUCTION

La recherche des motifs fréquents est un sujet d'actualité dans plusieurs applications de la fouille de données. Il est introduit pour la première fois par Agrawal et al. avec l'algorithme Apriori [34]. Cet algorithme, et ses variantes, utilisent une stratégie de parcours par niveaux. Ils identifient les i-motifs fréquents à la $i^{ème}$ itération puis génèrent les $(i+1)$-motifs fréquents à patir des i-motifs fréquents identifiés. Cependant, la phase de génération des motifs candidats est très coûteuse, plus particulièrement si le nombre de motifs est important et/ou la taille des motifs est grande. S'il existe un motif fréquent de longueur l, tous les 2^l sous-ensembles de ce motif sont générés. C'est pourquoi un intérêt particulier est donné à la recherche des motifs fréquents maximaux. L'objectif étant d'éviter un parcours exponentiel de l'ensemble des motifs candidats. Un motif fréquent *maximal* est un motif fréquent dont tous les sur-ensembles sont *non fréquents*. Les motifs fréquents maximaux forment ainsi une bordure au dessous de laquelle tous les motifs sont fréquents. Dans ce qui suit nous décrivons l'algorithme Fpmax pour la recherche des motifs fréquents maximaux.

A.2 RECHERCHE DES MOTIFS FRÉQUENTS MAXIMAUX AVEC FPMAX

A.2.1 *FP-Tree et FP-Growth*

FP-Growth [**?**] est un algorithme qui a marqué l'histoire de recherche des motifs fréquents. Il permet de générer les motifs fréquents en parcourant seulement deux fois la base des transactions en s'appuyant sur une structure d'arbre : le FP-tree (Frequent Pattern Tree).

Le premier parcours permet de générer tous les items fréquent (les items ayant un support $\geqslant$ à un support minimum donné). Le deuxième parcours construit le FP-Tree qui va contenir toutes les informations de la base d'origine. Ainsi, explorer la base des transactions revient simplement à manipuler l'arbre FP-Tree. L'approche mise en place pour découvrir les motifs fréquents à partir du FP-tree évite la génération d'un

grand nombre de motifs candidats. Le processus d'extraction se déroule en deux étapes principales :

1. La construction du FP-Tree,
2. La génération des motifs fréquents à partir du FP-Tree.

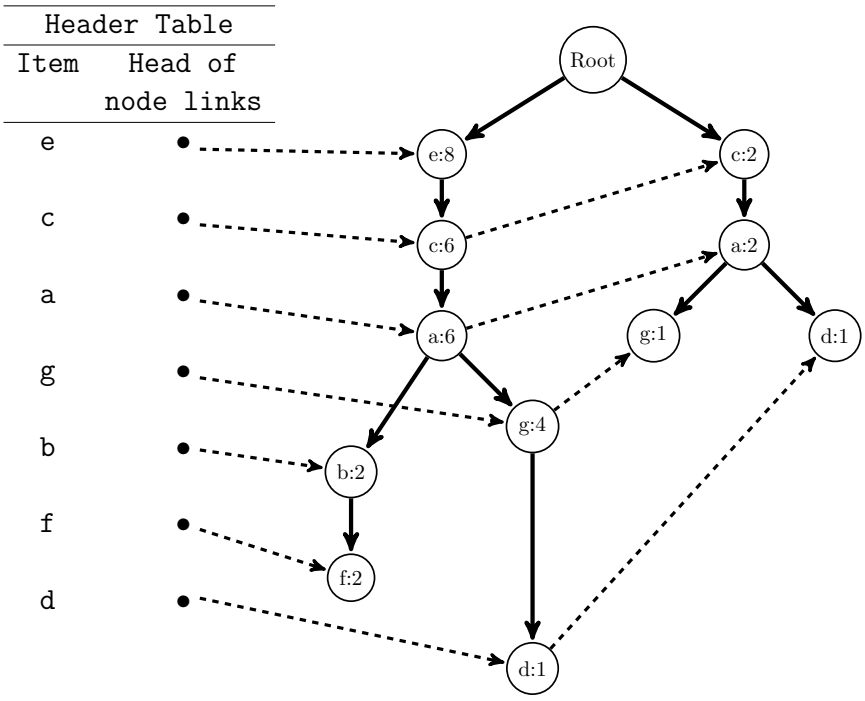
Le tableau 37 représente un exemple d'une base de transactions. Après le premier

a b c e f o
a c g
e i
a c d e g
a c e g i
e j
a b c e f p
a c d
a c e g m
a c e g n

TABLE 37: Exemple d'une base de transactions

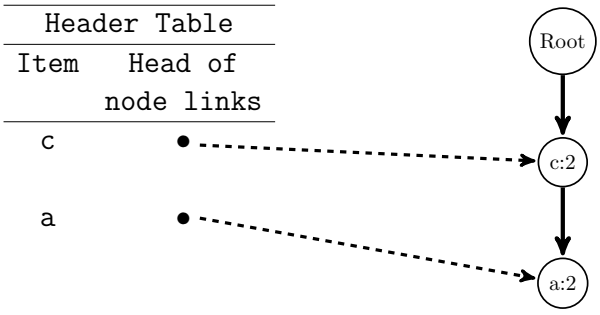
parcours de la base, tous les motifs fréquents sont insérés dans une table index (header table) associée au FP-Tree initial. La figure 30(a) illustre le FP-Tree initial construit après le second parcours de la base originale pour un support minimum de 20%. L'algorithme FP-Growth se base sur le principe suivant : si X et Y sont deux motifs alors le support du motif $X \bigcup Y$ dans la base des transactions est égal à celui de Y dans la restriction de la base originale aux transactions contenant X. Cette restriction de la base des transactions est appelée *base conditionnelle* de X.

Pour chaque base conditionnelle, l'algorithme FPGrowth construit un nouveau FPTree. La figure 30(b) indique la base conditionnelle et le FP-Tree correspondant pour l'item {d}. Cette étape est réalisée d'une manière récursive jusqu'a ce que le FP-Tree construit ne contient qu'un seul chemin. L'ensemble de tous les motifs fréquents est généré à partir des arbres (FP-Tree) à un seul chemin.



(a)

Base conditionnelle de {d}	
e c a g	: 1
c a	: 1



(b)

FIGURE 30: Exemple de FPTRee.

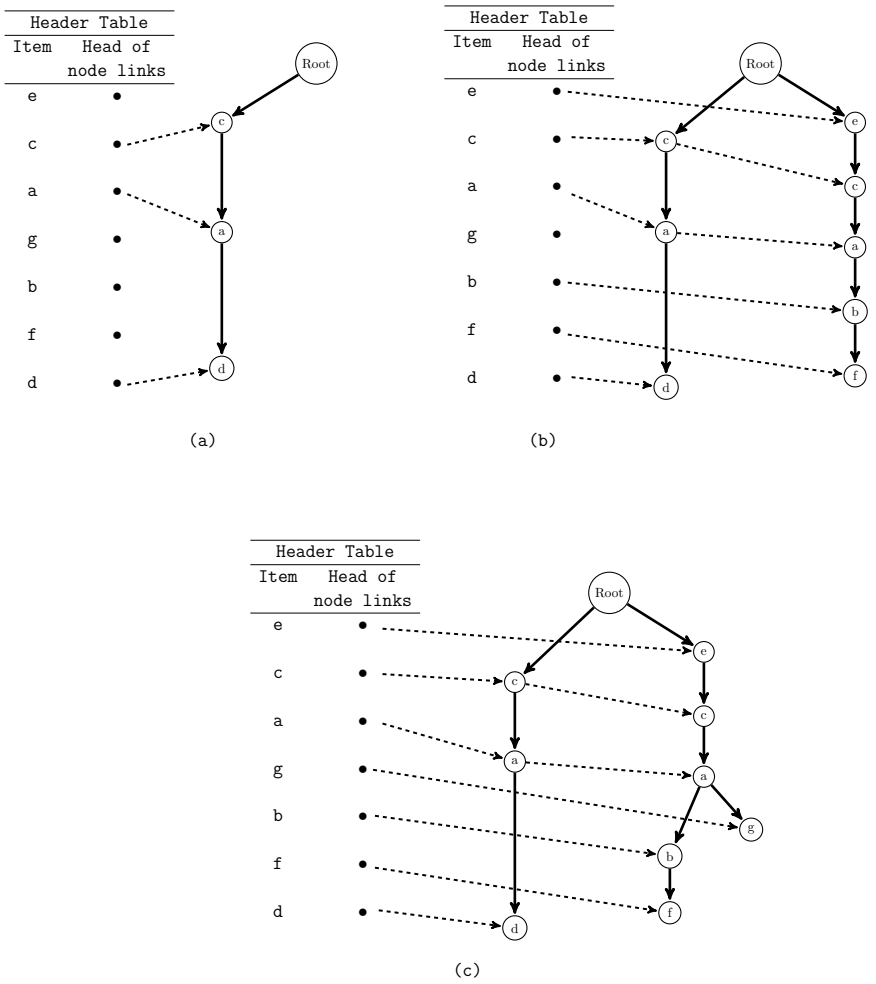


FIGURE 31: Exemple de MFIRee.

A.2.2 *FPMAX et MFI-Tree*

L'algorithme FPMAX est présenté dans [113] comme étant une extention de l'algorithme FP-Growth. A partir de la structure FP-Tree , FPMAX crée une structure similaire pour stocker les motifs fréquents maximaux : le MFI-Tree (Maximal Frequent Itemset Tree).

Avant d'appeler la procédure récursive FPMAX (Algorithme 6), le FP-Tree (T) est crée de la même manière que pour l'algorithme FP-Growth. Ensuite FPMAX est appelé avec T comme paramètre. Comme pour le FP-Growth, les items dans la table index (header table) sont traités à partir de le fin. A chaque étape, si le FP-Tree conditionnelle (T_i) correspondant à l'item{i} contient un seul chemin P alors ce dernier est inséré dans le MFI-Tree, sauf s'il est un sous ensemble d'un motif fréquent déjà inséré dans le MFI-Tree. Le résultat de cette procédure récursive est la construction de l'arbre MFI-Tree contenant tous les motifs fréquents maximaux.

Algorithme 6 : Algorithme *FPMAX*

Données : T : FPTRee ;
Résultat : M : MFITRee ;

```
1  début
2      MFIT : MFITRee = ∅;
3      Head, Tail : Items = ∅;
4      si T contient un seul chemin P alors
5          Insérer Head ∪ P dans MFIT;
6          sinon
7              pour chaque i dans le HeaderTable de T faire
8                  Ajouter i á Head;
9                  Construire la base conditionnelle B_i de i;
10                 Tail = les items fréquents dans B_i;
11                 si NON(Head ∪ Tail dans MFIT) alors
12                     Construire le FPTRee conditionnelle de Head (T_Head);
13                     fpmax(T_Head);
14                 fin
15                 Enlever i de Head;
16             fin
17         fin
18     fin
19 fin
```

Le MFI-Tree ressemble à un FP-Tree. Sa racine pointe vers "*null*". Chaque noeud de l'arbre contient deux champs : le nom de l'item et un lien qui pointe vers le prochain noeud ayant le même nom. Ainsi tous les noeuds portant le même nom son reliés ensembles. Un nouveau motif fréquent généré, à partir du FP-Tree initial, n'est inséré dans le MFI-Tree que s'il n'est pas un sous-ensemble d'un motif fréquent maximal déjà généré.

Pour illustrer la construction du MFI-Tree, reprenons l'exemple précédent. Le tableau 38 résume les bases conditionnelles et les FP-Tree corréspondants pour chaque item {i} dans l'index (*Header Table*).

Item	**Base conditionnelle**	**FP-Tree conditionnelle**
d	{(ecag :1),(ca :1)}	{(c :2,a :2)}
f	{(ecab :2)}	{(e :2,c :2,a :2,b :2)}
b	{(eca :2)}	{(e :2,c :2,a :2)}
g	{(eca :4),(ca :1)}	{(c :5,a :5,e :4)}
a	{(ec :6),(c :2)}	{(c :8,e :6)}
c	{(e :6)}	{(e :6)}
e	∅	∅

TABLE 38: Les bases conditionnelles et les FP-Tree correspondants

On commence par appeler FPMAX(T), T étant le FP-Tree de la base considérée (Figure 30). Comme T ne contient pas un seul chemin, une recherche des motifs fréquents maximaux est entammée en considérant les éléments de l'index (*Header Table*) de T à partir de la fin.

– Pour l'item {**d**} : l'arbre conditionnelle contient un seul chemin {c,a}. Comme le MFI-Tree ne contient aucun chemin (il est vide), {c,a,d} est alors inséré dans le MFI-Tree (Figure 31(a)).

– Pour l'item {**f**} : l'arbre conditionnelle contient un seul chemin {e,c,a,b}. Comme {e,c,a,b,f} ne figure pas dans MFI-Tree, il y est alors inséré (Figure 31(b)).

– Pour l'item {**b**} : l'arbre conditionnelle contient un seul chemin {e,c,a}. La vérification dans MFI-Tree montre que {e,c,a,b} existe déjà (il est sous-ensemble de {e,c,a,b, f}. Il n'est donc pas inséré dans le MFI-Tree.

– De même pour les autres items, on trouve que seul le motif {e,c,a,g} doit être ajouté à MFI-Tree(Figure 31(c)).

Chaque branche dans le MFI-Tree resprésente un motif fréquent maximal. Ainsi, les motifs fréquents maximaux pour cet exemple sont : {c, a, d}, {e, c, a, b, f}, {e, c, a, g}.

B

ANNEXE B : CHARGES UTILISÉES DANS LES EXPÉRIMENTATIONS

B.1 CHARGE UTILISÉE POUR LA SÉLECTION D'INDEX

```
(1) Select Time_Level,count(*) from Actvars A,ProdLevel P where A.Product_Level=
    P.Code_Level and P.Class_Level='PO0HV1RICH5W' Group by Time_Level
(2) Select Line_Level,sum(Dollarcost) from Actvars A,ProdLevel P where A.Product
    _Level=P.Code_Level and P.Class_Level='CI493YZ9KZUJ' Group by line_Level
(3) Select count(*) from Actvars A,ProdLevel P where A.Product_Level=P.Code_
    Level and P.Class_Level='FDXAQ1N5U026'
(4) Select Time_Level,Avg(UNITSSOLD) from Actvars A,TimeLevel T where A.Time_
    Level=T.TID and T.Quarter_Level='Q1' Group by Time_Level
(5) Select Division_Level,count(*) from Actvars A,ProdLevel P where A.Product_
    Level=P.Code_Level and P.Group_Level='E4NJTW0ZR9FN' Group by Division_Level
(6) Select Max(UNITSSOLD) from Actvars A,TimeLevel T where A.Time_Level=T.TID
    and T.Quarter_Level='Q2'
(7) Select Time_Level,count(*) from Actvars A,ProdLevel P where A.Product_Level=
    P.Code_Level and P.family_Level='BEMFVK0N8125' Group by Time_Level
(8) Select Year_Level,sum(Dollarcost) from Actvars A,ProdLevel P, TimeLevel T
    where A.Product_Level=P.Code_Level and A.Time_Level=T.TID and P.family_Level
    ='UJHZ4TZMJT6V' Group by Year_Level
(9) Select count(*) from Actvars A,ProdLevel P where A.Product_Level=P.Code_
    Level and P.family_Level='SHDF8QT29KFF'
(10) Select Customer_Level,Avg(Unitssold) from Actvars A,ProdLevel P where A.
    Product_Level=P.Code_Level and P.family_Level='AGG214DG271Q' Group by
    Customer_Level
(11) Select Month_Level,count(*) from Actvars A,TimeLevel Twhere A.Time_Level=T.
    TID and T.Quarter_Level='Q3' Group by Month_Level
(12) Select Sum(Dollarcost) from Actvars A,ProdLevel P where A.Product_Level=P.
    Code_Level and P.LINE_Level = 'MJ1F1U1EG009'
(13) Select Product_Level,count(*) from Actvars A,TimeLevel T where A.Time_Level
    =T.TID and T.Quarter_Level='Q4' Group by Product_Level
(14) Select Retailer_Level,Avg(unitssold) from Actvars A,ProdLevel P ,CustLevel
    C where A.Product_Level=P.Code_Level AND A.Custom_Level=C.Store_Level and  P
    .Division_Level = 'BCR2T4K2K9D3' Group by Retailer_Level
```

```
(15) Select count(*) from Actvars A,ProdLevel P where A.Product_Level=P.Code_
    Level and P.Division_Level='XRLXY6H61SLC'
(16) Select Customer_Level,Sum(Dollarcost) from Actvars A,ProdLevel P where A.
    Product_Level=P.Code_Level and P.Division_Level='RC5406URP1IE' Group by
    Customer_Level
(17) Select all_Level,count(*) from Actvars A,ProdLevel P ,ChanLevel H where A.
    Product_Level=P.Code_Level  AND A.Channel_Level=H.Base_Level  and P.Division
    _Level = 'G4HA5YITG3H7' Group by Channel_Level
(18) Select count(*) from Actvars A,TimeLevel T where A.Time_Level=T.TID and T.
    Year_Level='1995'
(19) Select Product_Level,Sum(Dollarcost) from Actvars A,TimeLevel T where A.
    Time_Level=T.TID and T.Year_Level='1996' Group by Product_Level
(20) Select Division_Level,Avg(Unitssold) from Actvars A,TimeLevel T ,ProdLevel
    P where A.Time_Level=T.TID AND A.Product_Level=P.Code_Level and T.Month_
    Level = '1' Group by Division_Level
(21) Select count(*) from Actvars A,TimeLevel T where A.Time_Level=T.TID and T.
    Month_Level='2'
(22) Select Product_Level,count(*) from Actvars A,TimeLevel T where A.Time_Level
    =T.TID and T.Month_Level='3' Group by Product_Level
(23) Select Division_Level,Sum(Dollarcost) from Actvars A,TimeLevel T ,ProdLevel
     P where A.Time_Level=T.TID AND A.Product_Level=P.Code_Level
and T.Month_Level='4' Group by Division_Level
(24) Select Avg(Unitssold) from Actvars A,TimeLevel T where A.Time_Level=T.TID
    and T.Month_Level='5'
(25) Select Product_Level,count(*) from Actvars A,TimeLevel T where A.Time_Level
    =T.TID and T.Month_Level='6' Group by Product_Level
(26) Select Division_Level,Sum(Dollarcost) from Actvars A,TimeLevel T ,ProdLevel
     P where A.Time_Level=T.TID and T.Month_Level='7' AND A.Product_Level=P.Code
    _Level Group by Division_Level
(27) Select Sum(Dollarcost) from Actvars A,TimeLevel T where A.Time_Level=T.TID
    and T.Month_Level='8'
(28) Select Customer_Level,count(*) from Actvars A,TimeLevel T where A.Time_
    Level=T.TID and T.Month_Level='9' Group by Customer_Level
(29) Select Year_Level,Sum(Dollarcost) from Actvars A,TimeLevel T where A.Time_
    Level=T.TID and T.Month_Level = '10' Group by Year_Level
(30) Select count(*) from Actvars A,TimeLevel T where A.Time_Level=T.TID and T.
    Month_Level='11'
(31) Select Product_Level,Time_Level,Avg(unitssold) from Actvars A,TimeLevel T
    where A.Time_Level=T.TID and T.Month_Level='12' Group by Product_Level,Time_
    Level
```

(32) Select Year_Level,Month_Level, Max(unitssold) from Actvars A,CUSTLevel C , TimeLevel T where A.Customer_Level=C.Store_Level and C.Retailer_Level = 'Z6OFS4YAAD4J' AND A.Time_Level=T.TID Group by Year_Level,Month_Level

(33) Select count(*) from Actvars A,CUSTLevel C where A.Customer_Level=C.Store_Level and C.Retailer_Level='RQJNEN0UPKMQ'

(34) Select Product_Level,Time_Level, Min(unitssold) from Actvars A,CUSTLevel C where A.Customer_Level=C.Store_Level and C.Retailer_Level='NXEYFSIQE3JM' Group by Product_Level,Time_Level

(35) Select Year_Level,Sum(Dollarcost) from Actvars A,CUSTLevel C ,TimeLevel T where A.Customer_Level=C.Store_Level AND A.Time_Level=T.TID and C.Gender_Level='M' Group by Year_Level

(36) Select count(*) from Actvars A,CHANLevel CH where A.Channel_Level=CH.Base_Level and CH.ALL_Level='ABCDEFGHIJKL'

(37) Select Channel_Level,Time_Level, Sum(Dollarcost) from Actvars A,CHANLevel CH where A.Channel_Level=CH.Base_Level and CH.ALL_Level='BCDEFGHIJKLM' Group by Channel_Level, Time_Level

(38) Select Class_Level,Year_Level, Time_Level, Avg(Unitssold) from Actvars A, CHANLevel CH ,ProdLevel P ,TimeLevel T where A.Channel_Level=CH.Base_Level AND A.Product_Level=P.Code_Level AND A.Time_Level=T.TID and CH.ALL_Level='CDEFGHIJKLMN' Group by Class_Level,Year_Level, Time_Level

(39) Select count(*) from Actvars A,CHANLevel CH where A.Channel_Level=CH.Base_Level and CH.ALL_Level ='DEFGHIJKLMNO'

(40) Select count(*) from Actvars A,CHANLevel CH where A.Channel_Level=CH.Base_Level and CH.ALL_Level ='EFGHIJKLMNOP' Group by Time_Level

(41) Select Year_Level,Month_Level, sum(dollarcost) from Actvars A,CUSTLevel C, ProdLevel P ,TimeLevel T where A.Customer_Level=C.Store_Level AND A.Time_Level=T.TID and A.Product_Level=P.Code_Level and P.Class_Level='CI493YZ9KZUJ' and C.Retailer_Level='RQJNEN0UPKMQ' AND C.City_Level='Bordeaux' Group by Year_Level,Month_Level

(42) Select sum(dollarcost) from Actvars A, ProdLevel P,TimeLevel T where A.Product_Level=P.Code_Level and A.Time_Level=T.TID and T.Quarter_LEVL='Q2' AND T.Year_Level='1996' and P.Class_Level='FDXAQ1N5U026'

(43) Select Customer_Level, Time_Level, Avg(Unitssold) from Actvars A, CHANLevel H,ProdLevel P, TimeLevel T,CUSTLevel C where A.Product_Level=P.Code_Level and A.Time_Level=T.TID AND A.Customer_Level=C.Store_Level and A.Channel_Level=H.Base_Level and P.Class_Level='FDXAQ1N5U026' and H.ALL_Level='ABCDEFGHIJKL' AND T.Quarter_Level='Q1' AND C.Gender_Level='F' Group by Customer_Level, Time_Level

(44) Select Year_Level, Division_Level, Max(Unitssold) from Actvars A, CUSTLevel C,TimeLevel T ,ProdLevel P where A.Customer_Level=C.Store_Level AND A.Product_Level=P.Code_Level and A.Time_Level=T.TID and T.Month_Level='1' and

C.Retailer_Level='RQJNEN0UPKMQ' AND C.Gender_Level='F' AND C.City_Level='Poitiers' Group by Year_Level, Division_Level

(45) Select sum(dollarcost), Avg(Unitssold) from Actvars A, CUSTLevel C, TimeLevel T where A.Customer_Level=C.Store_Level and A.Time_Level=T.TID and T.Month_Level='12' and C.Retailer='RQJNEN0UPKMQ' AND C.City_Level='Dijon'

(46) Select Customer_Level, Time_Level,Min(unitssold) from Actvars A, CHANLevel H,TimeLevel T, CUSTLevel C where A.Channel_Level=H.Base_Level and A.CUTOMER_Level=C.Store_Level AND A.Time_Level=T.TID and T.Year_Level='1996' AND T.Quarter_Level='Q3' and H.ALL_Level='DEFGHIJKLMNO' AND C.Gender_Level='M' Group by Customer_Level, Time_Level

(47) Select Month_Level, all_Level, Time_Level, Sum(Dollarcost) from Actvars A, CUSTLevel C,ProdLevel P,TimeLevel T ,ChanLevel H where A.Customer_Level=C.Store_Level and A.Product_Level=P.Code_Level AND A.Channel_Level=H.Base_Level and A.Time_Level=T.TID and T.Year_Level='1996' and C.Retailer_Level ='NXEYFSIQE3JM' AND C.City_Level='Paris' and P.LINE_Level='MJ1F1U1EG009' Group by Month_Level,all_Level, Time_Level

(48) Select Avg(unitssold) from Actvars A, CHANLevel H,ProdLevel P,TimeLevel T where A.Product_Level=P.Code_Level and A.Channel_Level=H.Base_Level and A.Time_Level=T.TID and T.Month_Level='1' and P.Division_Level='XRLXY6H61SLC' and H.ALL_Level='BCDEFGHIJKLM'

(49) Select Customer_Level, Product_Level, Channel_Level, sum(dollarcost) from Actvars A, CHANLevel H,CUSTLevel C,ProdLevel P where A.Customer_Level=C.Store_Level and A.Product_Level=P.Code_Level and A.Channel_Level=H.Base_Level and P.FAMILY_Level='AGG214DG271Q' and C.Retailer_Level='NXEYFSIQE3JM' AND C.Gender_Level='M' and H.ALL_Level='DEFGHIJKLMNO' Group by Customer_Level, Product_Level, Channel_Level

(50) Select Division_Level, Year_Level, Max(Unitssold) from Actvars A, CHANLevel H,CUSTLevel C,TimeLevel T ,ProdLevel P where A.Customer_Level=C.Store_Level AND A.Product_Level=P.Code_Level and A.Channel_Level=H.Base_Level and A.Time_Level=T.TID and T.Month_Level='4' and C.Retailer_Level='NXEYFSIQE3JM' AND C.City_Level='Poitiers' AND C.Gender_Level='F' and H.ALL_Level='BCDEFGHIJKLM' Group by Division_Level, Year_Level

(51) Select Min(Unitssold) from Actvars A, CHANLevel H,CUSTLevel C,ProdLevel P, TimeLevel T where A.Customer_Level=C.Store_Level and A.Product_Level=P.Code_Level and A.Channel_Level=H.Base_Level and A.Time_Level=T.TID and T.Month_Level='7' and P.Group_Level='E4NJTW0ZR9FN' and C.Retailer_Level='Z6OFS4YAAD4J' and H.ALL_Level='EFGHIJKLMNOP'

(52) Select Customer_Level, Channel_Level, sum(dollarcost) from Actvars A, CHANLevel H,CUSTLevel C,ProdLevel P,TimeLevel T where A.Customer_Level=C.Store_Level and A.Product_Level=P.Code_Level and A.Channel_Level=H.Base_Level and A.Time_Level=T.TID and T.Month_Level='11' and P.Class_Level ='FDXAQ1N5U026' and C.Retailer_Level ='RQJNEN0UPKMQ' AND C.City_Level='

```
    Poitiers' and H.ALL_Level='DEFGHIJKLMNO' Group by Customer_Level, Channel_
    Level
(53) Select line_Level, Month_Level, Avg(Unitssold) from Actvars A, CHANLevel H,
    CUSTLevel C,ProdLevel P,TimeLevel T where A.Customer_Level=C.Store_Level and
     A.Product_Level=P.Code_Level and A.Channel_Level=H.Base_Level and A.Time_
    Level=T.TID and T.Year_Level='1995' AND T.Quarter_Level='Q1' and P.Group_
    Level='E4NJTW0ZR9FN' and C.Retailer_Level='RQJNEN0UPKMQ' and H.ALL_Level='
    BCDEFGHIJKLM' Group by line_Level, Month_Level
(54) Select Min(Unitssold) from Actvars A, CHANLevel H,CUSTLevel C,ProdLevel P,
    TimeLevel T where A.Customer_Level=C.Store_Level and A.Product_Level=P.Code_
    Level and A.Channel_Level=H.Base_Level and A.Time_Level=T.TID and T.Year_
    Level='1995' AND T.Quarter_Level='Q1' and T.Month_Level in('2','3','4') and
    P.Class_Level='CI493YZ9KZUJ' and C.Retailer_Level in ('Z6OFS4YAAD4J','RQJNEN
    0UPKMQ') AND C.Gender_Level='F' AND C.City_Level='Poitiers' and H.ALL_Level
    ='ABCDEFGHIJKL'
(55) Select Customer_Level, Product_Level, Time_Level, Channel_Level, Max(
    Unitssold) from Actvars A, CHANLevel H,CUSTLevel C,ProdLevel P,TimeLevel T
    where A.Customer_Level=C.Store_Level and A.Product_Level=P.Code_Level and A.
    Channel_Level=H.Base_Level and A.Time_Level=T.TID and T.Year_Level='1996'
    and T.Month_Level in('5','6','7') and P.Class_Level='FDXAQ1N5U026' and C.
    Gender_Level='M' and C.Retailer_Level='RQJNEN0UPKMQ' and H.ALL_Level='
    DEFGHIJKLMNO' Group by Customer_Level, Product_Level, Time_Level, Channel_
    Level
(56) Select Sum(Dollarcost) from Actvars A,TimeLevel T where A.Time_Level=T.TID
    and C.Gender_Level='F'
(57) Select Month_Level,Sum(Dollarcost) from Actvars A,CUSTLevel C ,TimeLevel T
    where A.Customer_Level=C.Store_Level AND A.Time_Level=T.TID and C.City_Level
    ='Poitiers' Group by Month_Level
(58) Select Time_Level,avg(Dollarcost) from Actvars A,CUSTLevel C where A.
    Customer_Level=C.Store\_Level and C.City_Level='Bordeaux' Group by Time_
    Level
(59) Select avg(Dollarcost) from Actvars A,CUSTLevel C where A.Customer_Level=C.
    Store_Level and C.City_Level='Paris'
(60) Select Year_Level,max(Dollarcost) from Actvars A,CUSTLevel C where A.
    Customer_Level=C.Store_Level and C.City_Level='Dijon' Group by Year_Level
```

B.2 CHARGES UTILISÉES POUR LA FRAGMENTATION VERTICALE

POUR LA TABLE TAE

```
(1) Select speaker, course_instructor, course, semester, Class_size WHERE Class_
    size < 5
```

```
(2) Select speaker, course_instructor, semester, Class_size, Class_attribute
    WHERE course <= 20 AND Class_size > 18
(3) Select speaker, course, Class_attribute WHERE Class_size <= 25 AND (course_
    instructor <= 9 OR semester ='regula')
(4) Select speaker, course_instructor, course, semester, Class_attribute WHERE
    course_instructor < 10
(5) Select speaker, course_instructor, semester, Class_size, Class_attribute
    WHERE course_instructor < 15 AND (course >= 14 OR Class_attribute > 'low')
(6) Select speaker, course, semester, Class_size, Class_attribute WHERE Class_
    size = 64 AND semester='regular' AND course_instructor >= 5
(7) Select course_instructor, Class_size WHERE course_instructor = 11 OR Class_
    attribute > 'low' OR semester <= 'summer'
(8) Select speaker, course_instructor, semester, Class_size, Class_attribute
    WHERE speaker = 'english-speaker' AND course < 5
(9) Select Class_attribute WHERE Class_instructor <> 25 OR Class_attribute = '
    low'
(10) Select course_instructor, course, semester, Class_size WHERE Class_
    attribute <> 'low' AND Class_size >= 39
(11) INSERT SQL
(12) DELETE SQL
```

Pour la table ADULT

```
(1) Select Age, WorkClass, Final-weight, Education, Education-num, Occupation,
    Relationship, Race, Capital-gain, Hours-per-week, Native-country, Class
    WHERE (Material-status = 'Windowed' OR Marital-status = 'Divorced') AND Sex
    ='Female'
(2) Select Age, WorkClass, Final-weight, Education-num, Marital-status, Race,
    Sex, Capital-gain, Capitil-loss, Hours-per-week, Native-country, Class WHERE
     Relationship = 'Other-relative'
(3) Select Age, WorkClass, Final-weight, Education-num, Marital-status,
    Relationship, Race, Sex, Capital-gain, Capital-loss, Hours-per-week
WHERE Sex = 'Female'
(4) Select Education, Occupation, Relationship, Race, Capital-gain, Hours-per-
    week WHERE Native-country = 'United-states' AND Marital-status ='Divorced'
(5) Select Age, WorkClass, Education, Relationship, Class WHERE (Relationship =
    'Not-in family' OR Relationship = 'Unmarried') AND Class = '<=50K' AND Sex =
     'Male'
(6) Select Age, Final-Weight, Education, Education-num, Occupation, Relationship
    , Race, Capital-loss, Hours-per-week, Class WHERE  Age >= 50 AND (WorkClass
    = 'Self-emp-not-inc' OR WorkClass = 'Private')'
(7) Select Age, WorkClass, Education, Marital-status, Hours-per-week, Class
    WHERE Native-country = 'South'
```

```
(8) Select Age, Final-weight, Education-num, Occupation, Sex, Capital-gain,
    Capitalloss, Class WHERE Education-num < 10 AND Hours-per-week >50
(9) Select Education, Occupation, Capital-gain WHERE Capital-gain > 1000
(10) Select Age, Occupation, Capital-gain, Hours-per-week WHERE Race = 'White'
    AND Age <= 40
(11) Select  Education-num, Marital-status, Capital-gain WHERE (WorkClass = '
    State-gov' OR WorkClass = 'Federal-gov' OR WorkClass = 'Local-gov') AND
    Class = '<=50K'
(12) Select Education, Occupation, Capital-loss WHERE Relationship = 'Not-in
    family' AND Marital-status = 'Divorced'
(13) Select Age, Occupation WHERE Marital-status = 'Never-married'
(14) Select  Education, Education-num, Marital-Status, Capital-gain, Native-
    country WHERE Occupation = 'Farming-fishing' AND Class = '<=50K' AND Hours-
    per-week >=75
(15) Select Occupation, Hours-per-week WHERE Race = 'Black' AND WorkClass = '
    Federal-gov'
(16) Select Capital-loss, hours-per-week, Native-Country WHERE Education = '
    Masters' AND Capital-loss > 0
(17) Select Marital-status, Race, Native-country WHERE Occupation = 'Armed-
    Forces' OR Occupation = 'Protective-serv'
(18) Select Hours-per-week, Native-country WHERE WorkClass = 'Private' AND
    Relationship = 'Own-child' AND Sex = 'Male'
(19) INSERT SQL
(20) DELETE SQL
```

Pour le benchmark TPC-H

```
(1) Select l_returnflag, l_linestatus, l_quantity, l_extendedprice, o_orderdate,
     o_shippriority from lineitem, orders where l_orderkey=o_orderkey and o_
    orderdate < date ':2' and l_shipdate > date ':2' Group by l_orderkey, o_
    orderdate
(2) Select  sum(l_extendedprice * l_discount) as revenue, s_acctbal, s_name, n_
    name, p_partkey, p_mfgr, s_address, s_phone, s_comment
from lineitem, part, supplier, partsupp, nation, region where p_partkey = ps_
    partkey and s_suppkey = ps_suppkey and p_size = :1 and p_type like '%:2' and
     s_nationkey = n_nationkey and n_regionkey = r_regionkey and r_name = ':3'
    order by n_name, s_name, p_partkey;
(3) Select l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue, o_
    orderdate, o_shippriority from Customer, orders, lineitem
where c_mktsegment = ':1' and c_custkey = o_custkey and l_orderkey = o_orderkey
    and o_orderdate < date ':2' and l_shipdate > date ':2' Group by l_orderkey,
    o_orderdate, o_shippriority
```

```
(4) Select l_shipomode, l_quantity, o_orderdate, o_shippriority from lineitem,
    orders where l_orderkey = o_orderkey and l_commitdate < l_receiptdate Group
    by l_orderkey, o_orderdate, o_shippriority order by o_orderpriority;
(5) Select n_name, sum(l_extendedprice * (1 - l_discount)) as revenue from
    Customer, orders, lineitem, supplier, nation, region where   c_custkey = o_
    custkey and l_orderkey = o_orderkey and l_suppkey = s_suppkey and c_
    nationkey = s_nationkey and s_nationkey = n_nationkey and n_regionkey = r_
    regionkey and r_name = ':1' and o_orderdate >= date ':2' and o_orderdate <
    date ':2' + interval '1' Year Group by n_name
order by revenue;
(6) Select c_name, sum(l_extendedprice * l_discount) as revenue from Customer,
    lineitem where l_custkey = c_custkey and l_shipdate >= date ':1' and l_
    shipdate < date ':1' + interval '1' Year and l_discount between :2 - 0.01
    and :2 + 0.01 and l_quantity < :3 order by l_quantity
(7) Select s_nation, c_nation, extract(Year from l_shipdate) as l_Year, l_
    extendedprice * (1 - l_discount) as volume from supplier, lineitem, orders,
    Customer where s_suppkey = l_suppkey and o_orderkey = l_orderkey and c_
    custkey = l_custkey Group by s_nation, c_nation, l_Year
(8) Select extract(Year from o_orderdate) as o_Year, l_extendedprice * (1 - l_
    discount) as volume, n_name from  part, supplier, lineitem, orders, Customer
    , nation, region where        p_partkey = l_partkey and s_suppkey = l_
    suppkey and l_orderkey = o_orderkey and o_custkey = c_custkey
and c_nationkey = n_nationkey and s_nationkey = n_nationkey and o_orderdate
    between date '1995-01-01' and date '1996-12-31' and  p_type = ':3' Group by
    o_Year order by o_Year
(9) Select n_name, extract(Year from o_orderdate) as o_Year, l_extendedprice *
    (1 - l_discount) - ps_supplycost * l_quantity as amount
from part, supplier, lineitem, partsupp, orders, nation where s_suppkey = l_
    suppkey and ps_suppkey = l_suppkey and ps_partkey = l_partkey and p_partkey
    = l_partkey and o_orderkey = l_orderkey and s_nationkey = n_nationkey and p_
    name like '%:1%' Group by nation, o_Year order by nation, o_Year desc
(10) Select c_custkey, c_name, sum(l_extendedprice * (1 - l_discount)) as
    revenue, c_acctbal, n_name, c_address, c_phone, c_comment from Customer,
    orders, lineitem, nation where c_custkey = o_custkey and l_orderkey = o_
    orderkey and o_orderdate >= date ':1' and o_orderdate < date ':1' + interval
     '3' Month and l_returnflag = 'R' and c_nationkey = n_nationkey Group by c_
    custkey, c_name, c_acctbal, c_phone, n_name, c_address, c_comment order by
    revenue desc
(11) Select  l_returnflag, l_linestatus, l_quantity, l_extendedprice, n_name
    from lineitem, nation where l_nationkey = n_nationkey and l_shipdate <= date
     '1998-12-01' - interval ':1' day (3) Group by l_returnflag, l_linestatus
```

(12) Select l_shipmode, o_orderpriority from orders, lineitem where o_orderkey = l_orderkey and l_shipmode in (':1', ':2') and l_commitdate < l_receiptdate and l_shipdate < l_commitdate and l_receiptdate >= date ':3' and l_receiptdate < date ':3' + interval '1' Year Group by l_shipmode order by l_shipmode

(13) Select n_name, sum(l_extendedprice * (1 - l_discount)) as revenue from lineitem, nation where l_nationkey = n_nationkey Group by n_name

(14) Select 100.00 * sum(case when p_type like 'PROMO%' then l_extendedprice * (1 - l_discount) else 0 end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue from lineitem, part where l_partkey = p_partkey and l_shipdate >= date ':1' and l_shipdate < date ':1' + interval '1' Month;

(15) Select l_orderkey, l_extendedprice, l_discount, o_orderdate, o_shippriority, c_name from lineitem, Customer, orders where l_orderkey = o_orderkey and l_custkey = c_custkey and o_orderdate < date ':2' and l_shipdate > date ':2' Group by l_orderkey, o_orderdate, o_shippriority

(16) Select n_name as nation, l_linenumber, l_shipMode, from part, supplier, lineitem, partsupp, orders, nation where s_suppkey =l_suppkey and ps_suppkey = l_suppkey and ps_partkey = l_partkey and p_partkey = l_partkey and o_orderkey = l_orderkey and s_nationkey = n_nationkey and p_name like '%:1%' Group by nation order by nation

(17) Select sum(l_extendedprice) / 7.0 as avg_Yearly from lineitem, part where p_partkey = l_partkey and p_brand = ':1' and p_container = ':2' and l_quantity < avg(l_quantity)

(18) Select c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice, sum(l_quantity) from Customer, orders, lineitem where o_orderkey in l_orderkey having sum(l_quantity) > :1 and c_custkey = o_custkey and o_orderkey = l_orderkey Group by c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice

(19) Select sum(l_extendedprice* (1 - l_discount)) as revenue from lineitem, part where p_partkey = l_partkey and p_brand = ':1' and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG') and l_quantity >= :4 and l_quantity <= :4 + 10 and p_size between 1 and 5 and l_shipmode in ('AIR', 'AIR REG') and l_shipinstruct = 'DELIVER IN PERSON'

(20) Select s_name, s_address, 0.5 * sum(l_quantity) from lineitem, supplier, nation where l_suppkey = s_suppkey and l_shipdate >= date ':2' and l_shipdate < date ':2' + interval '1' Year order by s_name

(21) Select l_linenumber, l_returnflag, l_linestatus, l_quantity, s_name, s_address, s_phone, s_comment from lineitem, supplier where l_suppkey = s_suppkey and s_name = ':3' order by s_name.

BIBLIOGRAPHIE

[1] P. Lyman, Varian H.R, and K. Swearingen. *How much information*. Scool of Information Management and Systems, University of California at Berkeley, 2003. (Cité dans la page 1.)

[2] P. Gray and Hugh J. Watson. *Decision support in the data warehouse*. The Data warehousing institute series. Prentice Hall PTR, 1998. (Cité dans la page 1.)

[3] R. Grover. *Identification of Factors Affecting the Implementation of Data Warehouse*. PhD thesis, School of Management, Auburn University, 1998. (Cité dans la page 1.)

[4] J. Han and M. Kamber. *Data Mining : Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000. (Cité dans la page 1.)

[5] D. Kayser. *La représentation des connaissances*. Collection Informatique. Hermès, 1997. (Cité dans la page 1.)

[6] R. Benmessaoud. *Couplage de l'analyse en ligne et de la fouille de données pour l'exploration, l'agrégation et l'explication des données complexes*. PhD thesis, Université Lumière Lyon 2, 2006. (Cité dans la page 2.)

[7] Yoann Pitarch. *Résumé de Flots de Données : Motifs, Cubes et Hiérarchies*. PhD thesis, Université Montpellier 2, 2011. Thèse de doctorat. (Cité dans la page 2.)

[8] W. H. Inmon. *Building the Data Warehouse*. QED Information Sciences, Inc., Wellesley, MA, USA, 1992. (Cité dans la page 2.)

[9] A. Doucet and S. Gangarski. *Bases de Données et Internet, Modèles, langages et systèmes*. Editions Hermès, 2001. (Cité dans la page 2.)

[10] Xuedong Chen. *Star schema benchmark for data warehousing and adjoined dimension colimn clustering method for improved star schema query performance*. PhD thesis, University of Massachusetts Boston, 2008. (Cité dans la page 3.)

[11] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *SIGMOD Rec.*, 26(1) :65–74, March 1997. (Cité dans la page 3.)

[12] Maya Vilasrao Jawalge. Protecting and preserving mechanisms of dbms against insider threat. Master's thesis, University Magdeburg, Magdeburg, Germany, 2013. (Cité dans la page 3.)

[13] Louisa Demmou. Exploration de problèmes de performance d'un entrepôt de données. Master's thesis, Université de Sherbrooke, Québec, Canada, 2010. (Cité dans la page 3.)

[14] J. Darmont. Optimisation et évaluation de performance pour l'aide à la conception et à l'administration des entrepôts de données complexes. *HDR, Université Lumière Lyon 2*, 2006. (Cité dans les pages 3 et 38.)

[15] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit : The Complete Guide to Dimensional Modeling*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 2002. (Cité dans la page 3.)

[16] N. Karayannidis and A. Tsois. Processing star queries on hierarchically-clustered fact tables. *28th International Conference on Very Large Databases, Hong Kong*, pages 730–740, 2002. (Cité dans la page 3.)

[17] L. Bellatreche. *Utilisation des index et de la fragmentation dans la conception logique et physique d'un entrepôt de données*. PhD thesis, Université de Clermond Ferrand II, 2000. (Cité dans les pages 4, 36, 86, 87, 89, 116 et 121.)

[18] S. Krompass, H. Dayal, H. A. Kuno, and A. Kemper. Dynamic workload management for very large data warehouses : Juggling feathers and bowling balls. *In VLDB'07*, pages 1105—-1115, 2007. (Cité dans la page 4.)

[19] Kurt Stockinger, Kesheng Wu, and Arie Shoshani. Strategies for processing ad hoc queries on large data warehouses. In *Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP*, DOLAP '02, pages 72–79, New York, NY, USA, 2002. ACM. (Cité dans la page 5.)

[20] Kamel Boukhalfa. *De la conception physique aux outils d'administration et de tuning des entrepôts de données*. PhD thesis, Ecole nationale supérieure de mécanique et d'aérotechnique, Potiers, 2009. Thèse de doctorat. (Cité dans les pages 6, 34, 36, 38, 40, 44, 53, 69, 86, 87, 89 et 121.)

[21] Douglas Comer. The difficulty of optimum index selection. *ACM Trans. Database Syst.*, 3(4) :440–445, December 1978. (Cité dans les pages 5 et 39.)

[22] S. Chaudhuri. Index selection for databases : A hardness study and a principled heuristic solution. *IEEE Transactions on Knowledge and Data Engineering*, 16(11) :1313–1323, 2004. (Cité dans les pages 5, 16, 40, 41 et 53.)

[23] K. Boukhalfa, L. Bellatreche, and P. Richard. Fragmentation primaire et dérivée : Étude de complexité, algorithmes de sélection et validation sous oracle 10g. *Revue des Nouvelles Technologies de l'Information RNTI*, 2008. (Cité dans les pages 5 et 88.)

[24] A. G. Ganek and T. A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 42(1) :5–18, January 2003. (Cité dans la page 5.)

[25] Usama Fayyad, Gregory Piatetsky-shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17 :37–54, 1996. (Cité dans la page 7.)

[26] R. Rakotomalala and D. A. Zighed. *Extraction de connaissances à partir de données (ECD)*. Techniques de l'Ingénieur, H 3 744, Editions T.I., 2003. (Cité dans la page 8.)

[27] Benameur ZIANI. Application des techniques de fouille de données à l'analyse des fichiers log. Master's thesis, Université de Laghouat, 2007. Mémoire de Magister. (Cité dans la page 8.)

[28] Michael R. Garey and David S. Johnson. *Computers and Intractability ; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. (Cité dans les pages 10, 91 et 118.)

[29] Vincent Ng, Dik Man Law, Narasimhaiah Gorla, and Chi Kong Chan. Applying genetic algorithms in database partitioning. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, SAC '03, pages 544–549, New York, NY, USA, 2003. ACM. (Cité dans les pages 10, 88, 89, 91, 105 et 118.)

[30] Narasimhaiah Gorla and Betty Pang Wing Yan. Vertical fragmentation in databases using data-mining technique. In David Taniar and Laura Irina Rusu, editors, *Strategic Advancements in Utilizing Data Mining and Warehousing Technologies*, pages 178–197. IGI Global, 2010. (Cité dans les pages 10, 97, 99, 100, 102, 103, 105, 106, 107, 111, 112, 121, 131 et 137.)

[31] A.L. Hunor. *Contributions aux techniques de Prise de Décision et de Valorisation Financière*. PhD thesis, Institut National des Sciences Appliquées de Lyon, 2007. (Cité dans la page 16.)

[32] M. J. A. Berry and G. S. Linoff. *Data Mining Techniques : For Marketing, Sales, and Customer Relationship Management*. Wiley Publishing, 2004. (Cité dans la page 16.)

[33] S. Gosselin. *Recherche de motifs fréquents dans une base de cartes combinatoires*. PhD thesis, Université Claude Bernard Lyon 1, 2011. (Cité dans la page 17.)

[34] R. Agrawal ans T. Imielinski and A. Swami. Mining associations rules between sets of items in large databases. *the 1993 ACM SIGMOD Conference, Washington DC, USA*, 1993. (Cité dans les pages 17, 20 et 141.)

[35] J. Rabatel. *Extraction de motifs contextuels : Enjeux et applications dans les données séquentielles*. PhD thesis, Université Montpellier II, 2011. (Cité dans la page 18.)

[36] S. Parthasarathy and M. Coatney. Efficient discovery of common substructures in macromolecules. *3 IEEE International Conference on Data Mining, Maebashi City, Japan*, 2002. (Cité dans la page 18.)

[37] B. Possas, N. Ziviani, W. Meira Jr., and B. RibeiroNeto. Set-based model : A new approach for information retrieval. *25 International ACM SIGIR Conference on Research and Development in Information Retrieval , Tempere, Finland*, 2002. (Cité dans la page 18.)

[38] M. Zaki. Directions in protein contact map mining. *NSF Workshop on Next Generation Data Mining , Baltimore, USA*, 2002. (Cité dans la page 18.)

[39] F. Rioult and B. Crémilleux. Représentation condensée en présence de valeurs manquantes. *INFORSID*, pages 301–317, 2004. (Cité dans la page 21.)

[40] R. Ng, L.V.S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. *1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98), Seattle, WA*, pages 13–24, 1998. (Cité dans la page 21.)

[41] Heikki Mannila and Hannu Toivonen. Multiple uses of frequent sets and condensed representations (extended abstract). *In Proc. KDD Int. Conf. Knowledge Discovery in Databases*, pages 189–194, 1996. (Cité dans la page 21.)

[42] Francois Rioult. *Extraction de connaissances dans les bases de données comportant des valeurs manquantes ou un grand nombre d'attributs*. PhD thesis, Université de Caen Basse-Normandie, 2005. (Cité dans la page 21.)

[43] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proceedings of the 7th International Conference on Database Theory*, ICDT '99, pages 398–416, London, UK, UK, 1999. Springer-Verlag. (Cité dans la page 23.)

[44] Roberto J. Bayardo, Jr. Efficiently mining long patterns from databases. *SIGMOD Rec.*, 27(2) :85–93, June 1998. (Cité dans la page 23.)

[45] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Fast and memory efficient mining of frequent closed itemsets. *IEEE Transactions on Knowledge and Data Engineering*, 18(1) :21–36, 2006. (Cité dans la page 23.)

[46] S. Ben Yahia, T. Hamrouni, and E. Mephu Nguifo. Frequent closed itemset based algorithms : A thorough structural and analytical survey. *SIGKDD Explor. Newsl.*, 8(1) :93–104, June 2006. (Cité dans les pages 23 et 60.)

[47] Chedy Raïssi. *Extraction de Séquences Fréquentes : Des Bases de Données Statiques aux Flots de Données*. These, Université Montpellier II - Sciences et Techniques du Languedoc, July 2008. (Cité dans la page 23.)

[48] Alexandre Lourme. *Contribution à la Classification par Modèles de Mélange et Classification Simultanée d'Echantillons d'Origines Multiples*. PhD thesis, Université Lille 1, 2011. Thèse de doctorat. (Cité dans la page 25.)

[49] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering : A review. *ACM Comput. Surv.*, 31(3) :264–323, September 1999. (Cité dans la page 25.)

[50] Ardenne Cosmin LAZAR. *Méthodes non supervisées pour l'analyse des données multivariées*. PhD thesis, Université de Reims Champagne Ardenne, 2008. Thèse de doctorat. (Cité dans les pages 25 et 26.)

[51] Guénaël CABANES. *Classification non supervisée à deux niveaux guidée par le voisinage et la densité*. PhD thesis, Université Paris 13, 2010. Thèse de doctorat. (Cité dans les pages 25 et 26.)

[52] Guillaume Cleuziou, Lionel Martin, Viviane Clavier, and Christel Vrain. Ddoc : Overlapping clustering of words for document classification. In *SPIRE*, pages 127–128, 2004. (Cité dans la page 26.)

[53] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. Technical Report 00-034, University of Minnesota, 2000. (Cité dans la page 26.)

[54] Xiaowei Xu, Martin Ester, Hans-Peter Kriegel, and Jörg Sander. A distribution-based clustering algorithm for mining in large spatial databases. In *ICDE*, pages 324–331, 1998. (Cité dans la page 26.)

[55] Guillaume Cleuziou. *Une méthode de classiffication non-supervisee pour l'apprentissage de règles et la recherche d'information*. PhD thesis, Université d'Orléans, 2004. Thèse de doctorat. (Cité dans les pages 26 et 27.)

[56] D. Medin, R. Goldstone, and D. Gentner. Respects for similarity. *Psychological Review*, 100 :254–278, 1993. (Cité dans la page 27.)

[57] Mounzer BOUBOU. *Contribution aux méthodes de classification non supervisée via des approches prétopologiques et d'agrégation d'opinions*. PhD thesis, Université de Claude Bernard - Lyon I, 2007. Thèse de doctorat. (Cité dans la page 28.)

[58] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967. (Cité dans la page 28.)

[59] Nicoleta ROGOVSCHI. *Classiffication à base de modèles de mélanges topologiques des données catégorielles et continues*. PhD thesis, Paris 13 - Insitut Galilée, 2009. Thèse de doctorat. (Cité dans les pages 28 et 122.)

[60] Salma Jamoussi. *Méthodes statistiques pour la compréhension automatique de la parole*. PhD thesis, Université Henri-Poincaré Nancy 1, 2004. Thèse de doctorat. (Cité dans la page 28.)

[61] Douglas. Steinley. K-means clustering : A half-century synthesis. *British Journal of Mathematical and Statistical Psychology*, 59(1) :1–34, 2006. (Cité dans la page 28.)

[62] Ming-Chuan Wu. *Encoded Bitmap Indexes and Their Use for Data Warehouse Optimization*. PhD thesis, Université technique de Darmstadt, 2001. (Cité dans la page 34.)

[63] Kamel Aouiche. *Techniques de fouille de données pour l'optimisation automatique des performances des entrepôts de données*. PhD thesis, Université Lumière Lyon 2, 2005. Thèse de doctorat. (Cité dans les pages 34, 35, 47, 48, 53, 55, 56, 71 et 121.)

[64] Douglas Comer. Ubiquitous b-tree. *ACM Comput. Surv.*, 11(2) :121–137, June 1979. (Cité dans la page 34.)

[65] R. Strohm. *Oracle Database Concepts 1g,Oracle*. Redwood City,CA 94065, 2007. (Cité dans la page 34.)

[66] C. Russell and J. Stephens. *Beginning MySQL Database Design and Optimization : From Novice to Professional*. Apresspod Series. Apress, 2004. (Cité dans la page 34.)

[67] R. Bayer and E. McCreight. Organization and maintenance of large ordered indices. In *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '70, pages 107–141, New York, NY, USA, 1970. ACM. (Cité dans la page 34.)

[68] Thomas F. Zurek. *Optimisation of Partitioned Temporal Joins*. PhD thesis, University of Edinburgh, 1997. (Cité dans la page 36.)

[69] P. ONeill and D. Quass. Improved Query Performance with Variant Indexes. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 38–49. ACM, 1997. (Cité dans la page 36.)

[70] M. C. Wu and A. P. Buchmann. Encoded Bitmap Indexing for Data Warehouses. *International Conference on Data Engineering*, 1998. (Cité dans les pages 36 et 60.)

[71] Patrick O'Neil and Goetz Graefe. Multi-table joins through bitmapped join indices. *SIGMOD Rec.*, 24(3) :8–11, September 1995. (Cité dans la page 36.)

[72] Eduardo Gutarra Velez. Multi-column bitmap indexing. Master's thesis, University of New Brunswick, 2012. (Cité dans la page 38.)

[73] Y. Sharma and N. Goyal. An Efficient Multi-Component Indexing Embedded Bitmap Compression for Data Reorganization. *Information Technology Journal*, 7(1) :160–164, 2008. (Cité dans la page 38.)

[74] Kesheng Wu, Ekow Otoo, and Arie Shoshani. On the performance of bitmap indices for high cardinality attributes. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 24–35. VLDB Endowment, 2004. (Cité dans la page 38.)

[75] Kesheng Wu, Ekow J. Otoo, and Arie Shoshani. Optimizing bitmap indices with efficient compression. *ACM Trans. Database Syst.*, 31(1) :1–38, March 2006. (Cité dans la page 38.)

[76] Kesheng Wu, Kurt Stockinger, and Arie Shoshani. Breaking the curse of cardinality on bitmap indexes. In Bertram Ludäscher and Nikos Mamoulis, editors, *SSDBM*, volume 5069 of *Lecture Notes in Computer Science*, pages 348–365. Springer, 2008. (Cité dans la page 38.)

[77] M. Zaker S. Phon-Amnuaisuk and S.C. Haw. An adequate design for large data warehouse systems : Bitmap index versus b-tree index. *International Journal of Computer and Communication*, 2(2) :39–46, 2008. (Cité dans la page 38.)

[78] Kurt Stockinger, Kesheng Wu, and Arie Shoshani. Strategies for processing ad hoc queries on large data warehouses. In *Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP*, DOLAP '02, pages 72–79, New York, NY, USA, 2002. ACM. (Cité dans la page 38.)

[79] Piotr Kołaczkowski. *Autonomic Index Selection in Relational Database Management Systems by Evolutionary Transformations of Query Plans*. PhD thesis, Warsaw University of Technology, 2010. (Cité dans la page 38.)

[80] Mehdi Goli and Seyed Mohammad Taghi Rouhani Rankoohi. A new vertical fragmentation algorithm based on ant collective behavior in distributed database systems. *Knowl. Inf. Syst.*, 30(2) :435–455, 2012. (Cité dans la page 38.)

[81] lhem BOUSSAÏD. *Perfectionnement de méthaeuristiques pour l'optimisation continue*. PhD thesis, Université Paris-Est Créteil, 2013. Thèse de doctorat. (Cité dans la page 39.)

[82] K.Y. Whang. Index selection in relational databases. In *In International Conference on Foundations of Data Organization (FODO 85)*, page 487–500, 1985. (Cité dans les pages 40 et 53.)

[83] Frank M.R.and Omiecinski E. and Navathe S.B. Adaptive and automated index selection in rdbms. In *in 3rd International Conference on Extending Database Technology, EDBT*, pages 277–292, 1992. (Cité dans les pages 40, 41 et 53.)

[84] Chaudhuri S. and Narasayya V.R. Autoadmin 'what-if' index analysis utility. In *in ACM SIGMOD International Conference on Management of Data*, pages 367–378, 1998. (Cité dans les pages 40, 41 et 53.)

[85] M. Golfarelli, S. Rizzi, and E. Saltarelli. Index selection for data warehousing. In *4th International Workshop on Design and Management of Data Warehouses (DMDW 02)*, page 33–42, 2002. (Cité dans les pages 40, 43 et 53.)

[86] L. Bellatreche and K. Boukhalfa. Yet another algorithms for selecting bitmap join index. In *12th International Conference on Data Warehousing and Knowledge Discovery DAWAK'12, Bilbao, Spain*, pages 105–116, 2010. (Cité dans les pages 40, 44, 53, 56 et 59.)

[87] K. Boukhalfa, L. Bellatreche, and B. Ziani. Index de jointure binaire : Stratégies de sélection et étude de performance. *RNTI*, 2(2) :39–46, 2010. (Cité dans les pages 40, 44, 56 et 59.)

[88] Kratica J., Ljubi¢ I., and To2i¢ D. A genetic algorithm for the index selection problem. *Applications of Evolutionary Computing, LNCS*, 2611 :281–291, 2003. (Cité dans les pages 46 et 53.)

[89] Syswerda G. Uniform crossover in genetic algorithms. In *3rd International Conference on Genetic Algorithms ICGA*, page 2–9, 1989. (Cité dans la page 47.)

[90] Filipovi¢ V. Proposition for improvement tournament selection operator in génétic algorithms. Master's thesis, Faculty of Mathematics, Belgrade, 2000. (Cité dans la page 47.)

[91] G. Valentin M., Zuliani D., Zilio G., Lohman, and A. Skelley. B2 advisor : An optimizer smart enough to recommend its own indexes. In *16th International Conference on Data Engineering (ICDE 00)*, page 101–110, 2000. (Cité dans les pages 47 et 53.)

[92] Habiba Drias and Ibtissem Frihi. ACO based approach and integrating information retrieval technologies in selecting bitmap join indexes. In *2010 IEEE/-WIC/ACM International Conference on Web Intelligence, WI 2010, Toronto, Canada, August 31 - September 3, 2010, Main Conference Proceedings*, pages 448–451, 2010. (Cité dans les pages 48 et 53.)

[93] K. Aouiche, J. Darmont, O. Boussaid, and F. Bentayeb. Automatic selection of bitmap join indexes in data warehouses. In *7th International Conference on Data Warehousing and Knowledge Discovery (DAWAK 05)*, page 101–110, 2005. (Cité dans les pages 48, 50, 53, 56 et 59.)

[94] Aouiche K. and Darmont J. Data mining-based materialized view and index selection in data warehouses. *Journal of Intelligent Information Systems*, 33(1) :65–93, 2009. (Cité dans les pages 48, 56 et 59.)

[95] L. Bellatreche, R. Missaoui, H. Necir, and H. Drias. Selection and pruning algorithms for bitmap index selection problem using data mining. In *9th International Conference on Data Warehousing and Knowledge Discovery (DaWaK '07)*, page 221–230, 2007. (Cité dans les pages 50 et 53.)

[96] L. Bellatreche, R. Missaoui, H. Necir, and H. Drias. A data mining approach for selecting bitmap join indices. *Journal of Computing Science and Engineering*, 2(1) :206–223, 2008. (Cité dans les pages 50 et 53.)

[97] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of ACM SIGMOD international conference on Management of data*, page 23–34, 1979. (Cité dans les pages 51 et 64.)

[98] Mishra P. and Eich M. Join processing in relational databases. *ACM Computing Surveys*, 24(1) :63–113, 1992. (Cité dans la page 55.)

[99] Wu M. Query optimization for selections using bitmaps. In *ACM SIGMOD International Conference on Management of Data (SIGMOD 1999), Philadelphia, USA*, pages 227–238, 1999. (Cité dans la page 56.)

[100] N. Bruno. *Automated Physical Database Design and Tuning*. Database systems and applications. Taylor & Francis, 2011. (Cité dans la page 57.)

[101] P. Besse, C. Le Gall, N. Raimbault, and S. Sarpy. Data mining et statistique. *Journal de la société française de statistique*, 142(1) :5–36, 2001. (Cité dans la page 59.)

[102] Frédéric Pennerath. *Méthodes d'extraction de connaissances à partir de données modélisables par des graphes. Application à des problèmes de synthèse organique*. PhD thesis, Université Henri Poincaré – Nancy 1, 2009. Thèse de doctorat. (Cité dans la page 59.)

[103] J. Pei, B. Han, and W. Wang. Mining sequential patterns with constraints in large databases. In *the 11th International Conference on Information and Knowledge Management (CIKM'02), McLean, VA, USA*, page 18–25, 2002. (Cité dans la page 60.)

[104] Andreea Maria JULEA. *Extraction de motifs spatio-temporels dans des séries d'images de télédétection - Application à des données optiques et radar-*. PhD thesis, Université De Savoie, 2011. Thèse de doctorat. (Cité dans la page 60.)

[105] Raghav Kaushik, Philip Bohannon, Jeffrey F. Naughton, and Henry F. Korth. Covering Indexes for Branching Path Queries. In *ACM SIGMOD*, pages 133–144, Madison, WI, 2002. (Cité dans la page 65.)

[106] Faiza GHOZZI JEDIDI. *Conception et manipulation de bases de données dimensionnelles à contraintes*. PhD thesis, Université TOULOUSE III - PAUL SABATIER, 2004. Thèse de doctorat. (Cité dans la page 65.)

[107] B. Ziani and Y. Ouinten. An improved approach for automatic selection of multi-tables indexes in ralational data warehouses using maximal frequent itemsets. *Int. Dec. Tech.*, 7(4) :279–292, October 2013. (Cité dans la page 67.)

[108] OLAP Council. Apb-1 olap benchmark, release ii, www.olapcouncil.org, 1998. (Cité dans la page 69.)

[109] Priti Mishra and Margaret H. Eich. Join processing in relational databases. *ACM Comput. Surv.*, 24(1) :63–113, March 1992. (Cité dans la page 71.)

[110] Frédéric FLOUVAT. Etude expérimentale de la distribution des bordures de motifs fréquents. In *INFORSID*, pages 369–384, , Grenoble, 38000, 2005. (Cité dans la page 72.)

[111] Bart Goethals and Mohammed Javeed Zaki. Advances in frequent itemset mining implementations : Introduction to fimi03. In B. Goethals and M.J. Zaki, editors, *FIMI*, CEUR Workshop Proceedings. CEUR-WS.org, 2003. (Cité dans la page 72.)

[112] Frédéric Flouvat. *Vers des solutions adaptatives et génériques pour l'extraction de motifs intéressants dans les données*. PhD thesis, Université Blaise Pascal - Clermont II-, 2006. Thèse de doctorat. (Cité dans la page 72.)

[113] G. Grahne and J. Zhu. High performance mining of maximal frequent itemsets. In *SIAM Workshop High Performance Data Mining : Pervasive and Data Stream Mining*, 2003. (Cité dans les pages 72 et 145.)

[114] B. Ziani and Y. Ouinten. Mining maximal frequent itemsets : A java implementation of fpmax algorithm. In *Proceedings of the 6th International Conference on Innovations in Information Technology*, IIT'09, pages 11–15, Piscataway, NJ, USA, 2009. IEEE Press. (Cité dans les pages 72 et 112.)

[115] Doug Burdick, Manuel Calimlim, and Johannes Gehrke. Mafia : A maximal frequent itemset algorithm for transactional databases. In *In ICDE*, pages 443–452, 2001. (Cité dans la page 73.)

[116] Guimei Liu, Hongjun Lu, Jeffrey Xu Yu, Wei Wang 0011, and Xiangye Xiao. Afopt : An efficient implementation of pattern growth approach. In Bart Goethals and Mohammed Javeed Zaki, editors, *FIMI*, volume 90 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003. (Cité dans la page 73.)

[117] Gösta Grahne and Jianfei Zhu. Efficiently using prefix-trees in mining frequent itemsets. In Bart Goethals and Mohammed Javeed Zaki, editors, *FIMI*, volume 90 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003. (Cité dans la page 73.)

[118] Navathe S.B. Karlapalem K. and Morsi. Issues in distribution design of object-oriented databases. *Distributed Object Management*, 1994. (Cité dans la page 86.)

[119] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, Inc., New York, NY, USA, 3 edition, 2003. (Cité dans la page 86.)

[120] Hui Ma, Klaus-Dieter Schewe, and Qing Wang. A heuristic approach to cost-efficient fragmentation and allocation of complex value databases. In Gillian Dobbie and James Bailey, editors, *Seventeenth Australasian Database Conference (ADC2006)*, volume 49 of *CRPIT*, pages 183–192, Hobart, Australia, 2006. ACS. (Cité dans la page 86.)

[121] Ken Barker and Subhraj Bhar. A graphical approach to allocating class fragments in distributed object base systems. *Distributed and Parallel Databases*, 10(3) :207–239, 2001. (Cité dans les pages 87 et 89.)

[122] C. I. Ezeife and Ken Barker. Distributed object based design : Vertical fragmentation of classes. *Distributed and Parallel Databases*, 6(4) :317–350, 1998. (Cité dans les pages 87 et 89.)

[123] M. Tamer Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999. (Cité dans les pages 87, 89, 91, 98 et 104.)

[124] Hui Ma. *Distribution Design for Complex Value Databases*. PhD thesis, Massey University, 2007. (Cité dans les pages 87 et 91.)

[125] Jan-Marco Bremer and Michael Gertz. On distributing XML repositories. In *International Workshop on Web and Databases, San Diego, California*, pages 73–78, 2003. (Cité dans la page 87.)

[126] A. Koreichi and B. L. Cun. On data fragmentation and allocation in distributed object oriented databases. Technical report, PRiSM, Versailles University, 1997. (Cité dans la page 87.)

[127] Stefano Ceri and Giuseppe Pelagatti. *Distributed Databases Principles and Systems*. McGraw-Hill, Inc., New York, NY, USA, 1984. (Cité dans la page 89.)

[128] Jeyakumar Muhuraj. *A Formal Approach to the Vertical Partitioning problem in Distribued Database Design*. PhD thesis, University of Florida, 1992. (Cité dans les pages 89 et 91.)

[129] M. Ra. Horizontal partitioning for distributed database design. *Advances in Database Research*, page 101–120, 1993. (Cité dans la page 89.)

[130] Muhuraj J., Chakravarthy S.and Varadarajan R., , and Navathe S.B. A formal approach to the vertical partitioning problem in distributed database design.

In *Proceedings of the Second International Conference on Parallel and Distributed Information Systems*, page 26–34, 1993. (Cité dans la page 89.)

[131] Son J. H. and Kim M. H. An adaptable vertical partitioning method in distributed systems. *Journal of Systems and Software*, 73(3) :551–561, 2004. (Cité dans les pages 89 et 91.)

[132] Ezeife C. and Zheng J. Measuring the performance of database object horizontal fragmentation schemes. In *Proceedings of the International Database Engineering and Applications Symposium (IDEAS)*, page 408–414. IEEE Computer Society Press, 1999. (Cité dans les pages 89 et 105.)

[133] C. I. Ezeife and Ken Barker. A comprehensive approach to horizontal class fragmentation in a distributed object based system. *Distributed and Parallel Databases*, 3(3) :247–272, 1995. (Cité dans la page 89.)

[134] Ladjel Bellatreche, Kamalakar Karlapalem, and Ana Simonet. Horizontal class partitioning in object-oriented databases. In *Database and Expert Systems Applications*, pages 58–67. Springer Berlin Heidelberg, 1997. (Cité dans la page 89.)

[135] Ladjel Bellatreche, Kamalakar Karlapalem, and Gopal K. Basak. Query-driven horizontal class partitioning for object-oriented databases. In Gerald Quirchmayr, Erich Schweighofer, and Trevor J. M. Bench-Capon, editors, *DEXA*, volume 1460 of *Lecture Notes in Computer Science*, pages 692–701. Springer, 1998. (Cité dans la page 89.)

[136] Ladjel Bellatreche, Kamalakar Karlapalem, and Ana Simonet. Algorithms and support for horizontal class partitioning in object-oriented databases. *Distributed and Parallel Databases*, 8(2) :155–179, 2000. (Cité dans la page 89.)

[137] Ladjel Bellatreche, Ana Simonet, and Michel Simonet. Vertical fragmentation in distributed object database systems with complex attributes and methods. In *DEXA Workshop*, pages 15–21, 1996. (Cité dans la page 89.)

[138] Fernanda Baio and Marta Mattoso. A mixed fragmentation algorithm for distributed object oriented databases. In *In Proc. Of the 9th Int. Conf. on Computing Information*, pages 141–148, 1998. (Cité dans la page 89.)

[139] Chi-Wai Fung, Kamalakar Karlapalem, and Qing Li. Cost-driven vertical class partitioning for methods in object oriented databases. *The VLDB Journal*, 12(3) :187–210, October 2003. (Cité dans les pages 89 et 105.)

[140] K.-D. Schewe. Fragmentation of object oriented and semi-structured data. *Databases and Information Systems II*, pages 1–14, 2003. (Cité dans la page 89.)

[141] D. DeWitt and J. Gray. Parallel database systems : the future of high performance database systems. *Commun. ACM*, 35(6) :85–98, 1992. (Cité dans la page 89.)

[142] J. Rao C., Zhang G., Lohman, and N. Megiddo. Automating physical database design in a parallel database. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, page 558–569, 2002. (Cité dans la page 89.)

[143] P. Valduriez. Parallel database systems : Open problems and new issues. *Distributed and Parallel Databases*, 1(2) :137–165, 1993. (Cité dans la page 89.)

[144] T. Stöhrand H. Märtens and E. Rahm. Multidimensional database allocation for parallel data warehouses. In *Proceedings of the International Conference on Very Large Databases*, page 273–284, 2000. (Cité dans la page 89.)

[145] ELhoussaine ZIYATI. *Optimisation de requêtes OLAP en Entrepôts de Données : Approche basée sur la fragmentation génétique*. PhD thesis, Universié de Rabat, 2010. (Cité dans les pages 89, 91, 98 et 100.)

[146] W.J Labio, D. Quass, and B. Adelberg. Physical database design for data warehouses. In *Proceedings of the IEEE International Conference on Data Engineering*, page 277–288, 1997. (Cité dans la page 89.)

[147] Golfarelli M., Maio D., and Rizzi S. Vertical fragmentation of views in relational data warehouses. In *Proceedings of the Settimo Convegno Nazionale su Sistemi Evoluti Per Basi Di Dati (SEBD 99)*, page 19–33, 1999. (Cité dans les pages 89, 98 et 100.)

[148] Amin Y. Noaman and Ken Barker. A horizontal fragmentation algorithm for the fact relation in a distributed data warehouse. In *Proceedings of the Eighth International Conference on Information and Knowledge Management*, CIKM '99, pages 154–161, New York, NY, USA, 1999. ACM. (Cité dans la page 89.)

[149] C. I. Ezeife. Selecting and materializing horizentally partitioning warehouse views. *Data and Knowledge Engineering*, 36 :185–210, 2001. (Cité dans la page 89.)

[150] Camille Furtado, Alexandre A. B. Lima, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. Physical and virtual partitioning in olap database clusters. In *SBAC-PAD*, pages 143–150. IEEE Computer Society, 2005. (Cité dans la page 89.)

[151] Michael Hammer and Bahram Niamir. A heuristic approach to attribute partitioning. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, SIGMOD '79, pages 93–101, New York, NY, USA, 1979. ACM. (Cité dans la page 91.)

[152] Hassan I. Abdalla and F. Marir. Vertical partitioning impact on performance and manageability of distributed database systems (a comparative study of some vertical partitioning algorithms). In *18TH NATIONAL COMPUTER CONFERENCE*, page 8. Saudi Computer Society, 2006. (Cité dans la page 92.)

[153] Hoffer J. A. and Severance D. G. The use of cluster analysis in physical database design. In *Proceedings of the First International Conference on Very Large Data Bases (VLDB)*, page 69–86, 1975. (Cité dans les pages 92 et 100.)

[154] McCormick W. T., Schweitzer P. J., and White T. W. Problem decomposition and data reorganization by a clustering technique. *Oper. Res*, 20(5), 1972. (Cité dans la page 92.)

[155] Sylvain Guinepain and Le Gruenwald. Using cluster computing to support automatic and dynamic database clustering. In *CLUSTER*, pages 394–401. IEEE, 2008. (Cité dans la page 93.)

[156] Navathe S. B., S. Ceri, Wiederhold G., and Dour J. Vertical partitioning algorithm for database design. *ACM Transactions on Database Systems (TODS)*, 9(4), 1984. (Cité dans les pages 94 et 100.)

[157] Navathe S. and Ra M. Vertical partitioning for database design : A graphical algorithm. In *Proceedings of the ACM SIGMOD, Portland, OR*, 1989. (Cité dans les pages 94 et 100.)

[158] Wesley W. Chu and I. Ieong. A transaction-based approach to vertical partitioning for relational database systems. *IEEE Transactions on Software Engineering*, 19(8), 1993. (Cité dans les pages 95 et 100.)

[159] S. Agrawal, S. Chaudhuri, L. Kollar, A. Marathe, V. Narasayya, and M. Syamala. Database tuning advisor for microsoft sql server 2005 : Demo. In *Proceedings of the SIGMOD 2005*, 2005. (Cité dans les pages 95 et 100.)

[160] Datta A., Ramamritham K., and Thomas H. M. Curio : A novel solution for efficient storage and indexing in data warehouses. In *25th International Conference on Very Large Data Bases (VLDB 99), Edinburgh, UK, Morgan Kaufmann.*, page 730–733, 1999. (Cité dans les pages 97 et 100.)

[161] E. Ziyati, D. Aboutajdine, and A. El Qadi. Complete algorithm for fragmentation in data warehouse. In *Proceedings of The 7th WSEAS International Conference on Artificial Intelligence Knowledge Engineering and Data Bases (AIKED'08) Cambridge, England*, 2008. (Cité dans la page 98.)

[162] E. Ziyati, A. El Qadi, D.Aboutajdine, and L.Bellaterche. Genetic optimization in data warehouse design. *Computational Science*, 4(1) :70–79, 2010. (Cité dans la page 98.)

[163] L. Bellatreche and K. Boukhalfa. La fragmentation dans les entrepôts de données : une approche basée sur les algorithmes génétiques. *Revue des Nouvelles Technologies de l'Information (EDA'2005)*, pages 141–160, 2005. (Cité dans les pages 98 et 121.)

[164] Surajit Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '98, pages 34–43, New York, NY, USA, 1998. ACM. (Cité dans la page 104.)

[165] Mikal ZIANE. *Transformations de Programmes et de Requêtes pour Automatiser ou Faciliter la Production et la Maintenance du Logiciel*. PhD thesis, Université Pierre et Marie Curie (Paris 6), 2004. HDR. (Cité dans la page 104.)

[166] Chandrasekar.C Paramasivam . K. Multi-join operation using genetic algorithm in active data warehouse. *Asian Journal of Computer Science and Information Technology*, 2(5) :123–127, 2012. (Cité dans la page 104.)

[167] Toshihide Ibaraki and Tiko Kameda. On the optimal nesting order for computing n-relational joins. *ACM Trans. Database Syst.*, 9(3) :482–502, September 1984. (Cité dans la page 104.)

[168] Matthias Jarke and Jurgen Koch. Query optimization in database systems. *ACM Comput. Surv.*, 16(2) :111–152, June 1984. (Cité dans la page 104.)

[169] Robert Taylor. *Query Optimization for Distributed Database Systems*. PhD thesis, Hertford College University of Oxford, 2010. (Cité dans la page 104.)

[170] Andreas Matthias Weiner. *Cost-Based XQuery Optimization in Native XML Database Systems : Concepts, Implementation, and Empirical Evaluation*. PhD thesis, Technical University of Kaiserslautern, 2011. (Cité dans la page 104.)

[171] Priti Mishra and Margaret H. Eich. Join processing in relational databases. *ACM Comput. Surv.*, 24(1) :63–113, March 1992. (Cité dans les pages 105 et 127.)

[172] S. B. Yao. Approximating block accesses in database organizations. *Commun. ACM*, 20(4) :260–261, April 1977. (Cité dans les pages 105 et 106.)

[173] M. Lichman. UCI machine learning repository, 2013. (Cité dans la page 112.)

[174] Nikos Karayannidis, Aris Tsois, Timos Sellis, Roland Pieringer, Volker Markl, Frank Ramsak, Robert Fenk, Klaus Elhardt, and Rudolf Bayer. Processing star queries on hierarchically-clustered fact tables. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 730–741. VLDB Endowment, 2002. (Cité dans la page 116.)

[175] Hui Lei and Kenneth A. Ross. Faster joins, self-joins and multi-way joins using join indices. *Data Knowl. Eng.*, 28(3) :277–298, December 1998. (Cité dans la page 117.)

[176] Peter J. Haas, Fabian Hueske, and Volker Markl. Detecting attribute dependencies from query feedback. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 830–841. VLDB Endowment, 2007. (Cité dans la page 118.)

[177] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. Sw-store : A vertically partitioned dbms for semantic web data management. *The VLDB Journal*, 18(2) :385–406, April 2009. (Cité dans la page 120.)

[178] Narasimhaiah Gorla. A methodology for vertically partitioning in a multi-relation database environment narasimhaiah gorla. *Journal of Computer Science and Technology JCST*, 7(3), 2007. (Cité dans la page 121.)

[179] Pierrick Bruneau. *Contributions en classiffication automatique : agregation bayesienne de melanges de lois et visualisation interactive*. PhD thesis, Université de Nantes, 2010. (Cité dans la page 122.)

[180] Haytham ELGHAZEL. *Classification et Prévision des Données Hétérogènes : Application aux Trajectoires et Séjours Hospitaliers*. PhD thesis, Université Claude Bernard Lyon 1, 2007. (Cité dans la page 122.)

[181] Dawid Weiss. *Descriptive Clustering as a Method for Exploring Text Collections*. PhD thesis, Poznan University of Technology, 2006. (Cité dans la page 122.)

[182] Jérôme Darmont, Omar Boussaid, and Fadila Bentayeb. Dweb : A data warehouse engineering benchmark. In A. Min Tjoa and Juan Trujillo, editors, *DaWaK*, volume 3589 of *Lecture Notes in Computer Science*, pages 85–94. Springer, 2005. (Cité dans la page 124.)

[183] TPC-H. Transaction processing performance council, 2013. (Cité dans la page 125.)

[184] Benameur Ziani, François Rioult, and Youcef Ouinten. A constraint-based mining approach for multi-attribute index selection. In *ICEIS 2012 - Proceedings of the 14th International Conference on Enterprise Information Systems, Volume 1, Wroclaw, Poland, 28 June - 1 July, 2012*, pages 93–98, 2012. (Cité dans la page 139.)

[185] Benameur Ziani, Ahmed Benmlouka, and Youcef Ouinten. Improving index selection accuracy for star join queries processing : An association rules based approach. In *Advances in Intelligent Systems and Computing*, volume 220, pages 67–74. Springer, 2013. (Cité dans la page 139.)

[186] Raudel Hernández-León, José Hernández Palancar, Jesús Ariel Carrasco-Ochoa, and José Francisco Martínez Trinidad. A novel incremental algorithm for frequent itemsets mining in dynamic datasets. In José Ruiz-Shulcloper and Walter G. Kropatsch, editors, *CIARP*, volume 5197 of *Lecture Notes in Computer Science*, pages 145–152. Springer, 2008. (Cité dans la page 139.)