UNIVERSITE SAAD DAHLAB DE BLIDA

Faculté des Sciences

Département d'Informatique

MEMOIRE DE MAGISTER

En Informatique

Spécialité : Systèmes d'Informations et de Connaissances

Par

Mr MESSAOUDI Mounir

LA FRAGMENTATION VERTICALE DANS LES BASES DE DONNEES VOLUMINEUSES

Devant le jury composé de

A. GESSOUM	Professeur, USD de Blida	Président
H. BOUARFA ABED	Professeur, USD de Blida	Examinatrice
F. NADER	Maître de conférences A, ESI	Examinatrice
S. OUKID-KHOUAS	Maître de conférences A, USD de Blida	Promotrice
B.LADJEL	Professeur, ENSMA, Poitiers, France	Co-Promoteur

Blida, Novembre 2012

RESUME

L'objectif de notre travail est de faire un état de l'art sur la fragmentation verticale (FV) et d'explorer les différents algorithmes pour sa résolution. Cette étude nous a permis de proposer une méthode de sélection d'un schéma de FV pour un entrepôt de données relationnelles. Nous avons développé un algorithme basé sur les algorithmes génétiques (AGs) pour sélectionner un schéma de FV presque optimal d'une ou de plusieurs tables de dimension. Le modèle de coût mathématique que nous avons proposé estime le nombre d'entrées/sorties nécessaires pour l'exécution d'une requête.

Nous avons réalisé une implémentation et des expérimentations afin d'évaluer notre algorithme. Finalement, nous avons effectué une validation sous ORACLE10g en utilisant les données issues du banc d'essai DWEB.

Mots clés : Techniques d'optimisation, Fragmentation verticale, Conception physique, Tuning, Entrepôt de données.

ملخص

الهدف من عملنا هو حالة القيام بدراسة حول التقسيم العمودي لأعمدة جداول قاعدة البيانات واستكشاف مختلف الخوارزميات لحل مشاكله .سمحت لنا هذه الدراسة باقتراح طريقة لاختيار مخطط للتقسيم العمودي لمستودع بيانات علائقي .لقد طورنا في عملنا هذا خوارزمية مبنية على الخوارزميات الجينية لاختيار مخطط التقسيم العودي الأمثل لجدول بعد واحد أو عدة جداول .نموذج التكلفة الذي اقترحناه يستند على التقديرات الحسابية لعدد المدخلات والمخرجات اللازمة لتنفيذ الاستعلام.

لقد قمنا بتحقيق تنفيذ وتجارب تهدف لتقييم الخوارزمية اتي اقترحناها. أخيرا أجرينا مجموعة من التحقيقات الصحة على نضامOrace10G باستخدام بيانات من DWEB.

كلمات المفتاح. : تقنيات التحسين، التقسيم العمودي، التصميم المادي، التضبيط، مستودع البيانات.

ABSTRACT

The aim of our work is to do a state of the art on the vertical fragmentation (VF) and to explore different algorithms for its resolution. This study allowed us to propose a method for selecting a pattern of VF for a relational datawarhouse. We have developed an algorithm based on genetic algorithms (GAs) to select a pattern of VF near-optimal of one or more dimension tables. The mathematical cost model that we proposed estimates the number of I/O required for the execution of a query.

We performed an implementation and experiments to evaluate our algorithm. Finally, we conducted a validation in ORACLE10g on using the data descended from test bench DWEB.

Keywords: Optimization techniques, vertical fragmentation, Physical Design, Tuning, Data Warehouse.

REMERCIEMENTS

Au terme de ce modeste travail, je voudrais adresser mes vifs remerciements à toutes les personnes qui ont contribué, de près ou de loin, à accomplir ce présent travail.

Je remercie Mr Bellatreche L., Professeur à l'ENSMA, Poitiers en France, pour m'avoir accordé l'honneur de travailler avec lui, pour m'avoir permis, par la même oc- casion, de mettre un pas dans le grand monde de la recherche et pour tous les conseils qu'il m'a prodigué tout au long de la réalisation de mon travail.

J'exprime ma profonde gratitude à Mme Oukid.K, Docteur à l'USDB de Blida, pour sa disponibilité, pour le temps précieux qu'elle m'a accordé et pour ses observations qui ont contribué à améliorer la qualité de ce travail.

Je tiens à remercier très sincèrement l'ensemble des membres du jury qui me font le grand honneur d'accepter de juger mon travail.

Cette thèse a été menée dans le cadre de l'Ecole Doctorale (STIC) de l'Université Saad Dahlab de Blida (USDB). Ainsi, mes remerciements s'adressent à tous enseignants et mes collègues de l'Ecole Doctorale (STIC).

Un grand merci à ma mère et ma sœur qui m'ont toujours soutenu, toujours encouragé et qui ont fait de moi ce que je suis aujourd'hui. Merci à ma famille pour tous les moments de bonheur partagés.

A mes amis de tout temps et tout bord, à ceux et celles qui ont sû trouver les mots pour m'aider à franchir les obstacles et à avancer dans la vie.

TABLES DES MATIERES

TABLE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GENERALE

Les entrepôts de données (ED) [1] sont dédiés aux applications d'analyse et de prise de décision. Cette analyse est souvent réalisée par l'intermédiaire de requêtes complexes caractérisées par des opérations de sélection, de jointure et d'agrégation. Exécuter de telles requêtes sur un entrepôt volumineux nécessite un temps de réponse très élevé qui n'est pas acceptable par les décideurs qui exigent un temps de réponse raisonnable afin de répondre aux besoins décisionnels. Pour réduire ce dernier et satisfaire les besoins des décideurs, l'administrateur de l'entrepôt est amené à faire une bonne conception physique, où il doit sélectionner un ensemble de techniques d'optimisation qu'il considère pertinent pour l'ensemble des besoins des décideurs (exprimés sous forme de requêtes) [2].

En plus des index, et les vues matérialisées, la fragmentation est une technique de conception et d'optimisation des bases de données utilisées dans les modèles relationnels et objets. Elle consiste à partitionner un schéma d'une base de données en plusieurs sous-schémas dans le but de réduire le temps d'exécution des requêtes [3]. Un schéma de fragmentation (SF) est le résultat d'un processus de fragmentation. Il y a trois sortes de fragmentations [4] possibles à savoir, la fragmentation horizontale, la fragmentation verticale et la fragmentation hybride.

La **fragmentation horizontale** (FH) partitionne une relation selon ses tuples : chaque fragment a un sous ensemble de tuples de la relation. Il existe deux manières pour fragmenter une base de données horizontalement : la fragmentation horizontale primaire et la fragmentation horizontale dérivée.

La fragmentation primaire [5] d'une relation est effectuée grâce à des prédicats de sélection définis sur la relation.

Par exemple : Soit la table Client(NClient, Nom, Ville) peut être fragmentée comme suit :

$$Client_1 = SELECT * FROM\ Client\ WHERE\ Ville = "Alger"$$
$$Client_2 = SELECT * FROM\ Client\ WHERE\ Ville <> "Alger"$$

La reconstruction de la relation initiale est réalisée à l'aide d'une opération d'union :

$$Client = Client_1 \cup Client_2$$

La fragmentation horizontale dérivée s'effectue avec des prédicats de sélection définis sur une autre relation. Les fragments dérivés sont obtenus par l'opération de semi-jointure.

Par exemple, considérons la table Commande (NClient, NProduit, Date, Qte, Nprésentant) :

$$Commande_1 = SELECT * FROM\ Commande\ WHERE\ NClient\ IN\ (SELECT\ NClient\ FROM\ Client_1)$$
$$Commande_2 = SELECT * FROM\ Commande\ WHERE\ NClient\ IN\ (SELECT\ NClient\ FROM\ Client_2)$$

La reconstruction de la relation initiale est comme suite :

$$Commande = Commande_1 \cup Commande_2$$

La **fragmentation verticale** (FV) consiste à diviser une relation R en sous relations $\{R_1, R_2, \ldots, R_n\}$ appelées fragments verticaux qui sont des projections appliquées à la relation. Chaque fragment contient un sous ensemble d'attributs de R en plus de la clé primaire. Par exemple, la table Client(CID, Nom, Sexe, Ville) peut être fragmentée comme suit :

$$Client_1 = SELECT\ CID,\ Nom\ FROM\ Client$$
$$Client_2 = SELECT\ CID,\ Sexe,\ Ville\ FROM\ Client$$

La reconstruction de la relation initiale est définie par l'opération de jointure sur la clé primaire de la table :

$$Client = Client_1\ Join\ Client_2$$

Dans la plupart des cas, une fragmentation horizontale ou verticale d'une base de données ne satisfait pas les demandes des applications utilisateur. **La fragmentation hybride consiste** à effectuer une fragmentation horizontale suivie par une fragmentation verticale ou vice-versa. Dans notre travail nous nous intéressons à la fragmentation verticale (dite aussi partitionnement verticale).

En raison de la criticité des performances des bases de données, plusieurs chercheurs ont contribué à la fragmentation verticale. Cette dernière a été appliquée dans les bases de données relationnelles centralisées [6,10, 15, 18, 28, 20, 21, 40], les bases de données réparties [3, 48, 41, 42, 39], et dans la conception des bases de données orientées objets ([12], [14]).

Dans les travaux ([37],[53],[55]), les chercheurs ont proposé des solutions basées sur la FV dans la conception des entrepôts de données.

Une relation avec n attributs peut être partitionnée en B_n façons différentes, où B_n est le $n^{ième}$ nombre de Bell [64] :

$$B_n = \sum_{k=1}^{n} \left\{ {n \atop k} \right\}$$

Où $\left\{ {n \atop k} \right\}$ représente le nombre de Stirling de seconde espèce.

Considérons un schéma relationnel avec N relations, où n_i est le nombre d'attributs de la relation i, en utilisant l'énumération exhaustive, le nombre de schémas de fragmentations vertical (SFVs) possibles pour le schéma de N-relations est d'environ
$B_{n_1} \times B_{n_2} \times \ldots B_{n_N}$. Yu, Chen, Heiss, et Lee [63] ont trouvé que le nombre d'attributs pour les tables de base et les vues dans un environnement relationnel typique est de 18 et 41, respectivement. Si nous considérons un petit schéma de 10 relations avec 15 attributs par relation, le nombre de fragments possibles est assez grand environ $10 \times B15 = (10^9)^{10} = 10^{90}$.

Ceci montre la complexité de la FV qui est essentiellement dû au grand nombre d'alternatives possibles. Donc, il est nécessaire de définir des algorithmes approchés qui permettent de mieux parcourir l'espace de recherche et de retrouver une solution satisfaisante. La fragmentation verticale représente un enjeu important dans les entrepôts que dans un contexte de base de données relationnelles ou objets. Cette importance est due au choix des tables (dimensions ou de faits) à fragmenter. Les choix possibles sont les suivants :

– Fragmenter verticalement une ou plusieurs tables de dimension. Ce choix pourrait être intéressant dans le cas où la taille des tables de dimension est importante. Cette fragmentation pourrait réduire le coût des opérations de jointures en étoile.

– Partitionner seulement la table de faits. Celle-ci est composée des clés étrangères des tables de dimension et des mesures. La fragmentation consiste à partitionner l'ensemble des mesures en différents fragments. Les clés étrangères seront dupliquées dans chaque fragment vertical.

Dans ce contexte, nous avons étudié le problème de sélection de schéma de fragmentation verticale dans un entrepôt de données relationnelles, et nous avons proposé un algorithme approché pour la sélection d'un schéma de FV d'une table de dimension.

Objectifs et contributions

L'objectif de notre travail est d'étudier les différents algorithmes de résolution de la fragmentation verticale, de comparer les différents travaux et présenter leurs avantages et leurs inconvénients.

Cette étude nous a permis de proposer des solutions (algorithmiques) au problème de sélection de schéma de fragmentation verticale dans un entrepôt de don- nées relationnelle. Nous avons développé un algorithme [92] basé sur les algorithmes génétiques (AGs) pour sélectionner un schéma de FV presque optimal d'une ou de plu- sieurs tables de dimension. Le modèle de coût

mathématique que nous avons proposé estime le nombre d'entrées/sorties nécessaires pour l'exécution d'une requête. Notre algorithme a été implémenté en utilisant le langage Java et une étude expérimentale a été effectuée sous ORACLE 10G en utilisant les données issues du banc d'essai DWEB.

Organisation du mémoire

Notre mémoire est organisée comme suit :

Le chapitre un, s'articule autour des entrepôts de données, leurs caractéristiques et architecture, ainsi que les modèles et les langages de modélisation. Ensuite, nous exposons les techniques d'optimisation des requêtes, à savoir les vues matérialisées, les index et la fragmentation verticale dans les entrepôts de données.

Dans le deuxième chapitre, nous présentons un état de l'art des travaux concernant la fragmentation verticale de données dans le contexte relationnel centralisé et distribué, objet distribué et les entrepôts de données.

Le troisième chapitre, nous détaillons quelques algorithmes de fragmentation verticale. Nous expliquons leurs démarches et nous essayons d'extraire leurs avantages et leurs inconvénients.

Notre contribution est présentée dans le chapitre quatre. Vue que la complexité du problème de la FV qui est du au nombre exhaustive des solutions, pour cela, nous proposons une méthode approchée de résolution de problème de sélection d'un schéma de fragmentation verticale basée sur les algorithmes génétiques.

Dans le chapitre cinq, nous donnons une implémentation de l'algorithme génétique et les résultats des expériences que nous avons menées. Nous décrivons d'abord la plateforme logicielle adoptée pour le construire, puis l'architecture globale de l'implémentation. Ensuite nous présentons quelques interfaces de prototype et les tests d'expérimentation sur un banc d'essai.

CHAPITRE 1
Les entrepôts de données: concepts et techniques d'optimisation

1.1 Introduction

Actuellement, les données utilisées et échangées par les applications décisionnelles sont de plus en plus diverses et hétérogènes. Le concept d'entrepôt de données est développé au début des années 90, il est devenu depuis la clé de ce que l'on appelle l'informatique décisionnelle. La vogue grandissante de ce concept est la résultante de l'évolution de l'informatique dans les organisations.

Dans ce chapitre nous présentons les concepts des entrepôts de données (ED) et les techniques principales utilisées pour optimiser le temps d'exécution des requêtes de systèmes de gestions des bases de données, ainsi que l'application de la fragmentation verticale dans les entrepôts de données.

1.2 Les entrepôts de données

Il existe plusieurs définitions d'un entrepôt de données (Data warehouse), selon certains auteurs [4, 44, 2] ont donné lieu à la définition d'un entrepôt de données.

Définition 1.2.1. *Les entrepôts de données sont définis par Inmon et Hackarton [2] en 1994 : «A data warehouse is a subject-oriented, integrated, time-variant, non-volatile col lection of data used in support of management decision making processes».*

On peut traduire cette définition comme suit : «Les données d'un entrepôt de données sont intégrées, orientées sujet, non volatiles, historiées, résumées et disponibles pour l'interrogation et l'analyse» [27].

Définition 1.2.2. *Un entrepôt de données peut être vu comme « un ensemble de vues matérialisées définies par des relations sur des sources de données distantes »* [4].

Cette définition semble être une simple explication d'une méthode pratique pour réaliser un entrepôt, les vues matérialisées ne permettent pas de résoudre tous les problèmes d'implémentation d'un entrepôt, même si elles peuvent faciliter le chargement des don- nées. Cette définition ne tient pas compte de la nature historique d'un entrepôt, elle ne prévoit pas de méthode pour historier les données qui proviennent des sources de données de l'entrepôt. Des tables supplémentaires sont nécessaires pour créer un historique, car une vue matérialisée effectue une copie des données et supprime la version précédente.

Le rôle principal de l'administrateur est de minimiser l'accès aux sources de données qui ont la particularité d'être volumineuses et hétérogènes grâce à l'utilisation de techniques d'optimisation de requêtes. Parmi ces techniques, nous pouvons citer les index et les vues matérialisées. Il existe d'autres techniques telles que la fragmentation, le clustering et le traitement parallèle des requêtes.

1.2.1 Caractéristiques des entrepôts de données

A partir de la définition 1.2.1 sur les entrepôts de données, les données d'un entrepôt sont :

– *Orientées sujet* : Les données sont orientées métiers et donc triées par thèmes.

  L'intégration dans une structure unique est indispensable pour éviter aux données concernées par plusieurs sujets d'être dupliquées. Cependant, en pratique, il existe également des entrepôts plus (data marts) : l'entrepôt est fragmenté en plusieurs bases qui supportent l'orientation sujet. L'intérêt de cette organisation est de pou- voir disposer de l'ensemble des informations utiles sur un sujet le plus souvent transversal aux structures fonctionnelles et organisationnelles de l'entreprise.

– *Intégrées* : Les données d'un entrepôt sont le résultat de l'intégration de données en provenance de multiples sources. Avant d'être intégrées dans l'entrepôt, les données doivent êtres mises en forme et unifiées afin d'avoir un état cohérent. Cela nécessite un gros travail de normalisation, de gestion des référentiels et de cohérence. Une donnée doit avoir une description et un codage unique. Cette phase d'intégration ou de nettoyage des données est très complexe et représente 60 à 90% de la charge totale d'un projet.

– *Non volatiles* : Les données sont stables et non modifiables. Un entrepôt de données doit garantir qu'une requête lancée à différentes dates sur les mêmes données donne toujours les mêmes résultats. De plus, les données d'un entrepôt sont mises à jour périodiquement, ce ne sont donc pas des informations en temps réel.

– *Historisées* : Les données sont historiées et donc datées : l'historisation est nécessaire pour suivre dans le temps l'évolution des différentes valeurs des indicateurs à analyser. Ainsi, un référentiel temps doit être associé aux données afin de permettre l'identification de valeurs précises dans la durée.

– *Aide à la décision* : Un entrepôt de données est destiné à l'aide à la décision, ce qui fait que les traitements qui s'y appliquent sont de nature différente de ceux des systèmes transactionnels, on parle ainsi de OLAP par opposition à OLTP. La plupart des traitements transactionnels en ligne n'impliquent que quelques données, occasionnent des changements dans la base de données et requièrent une réponse instantanée alors que les traitements mis en œuvre pour l'aide à la décision impliquent la lecture de nombreuses données mais n'entraînent pas de changement dans la base de données et ne requièrent pas une réponse instantanée.

1.2.2 Architecture d'un entrepôt de données

L'architecture des entrepôts de données repose souvent sur un SGBD séparé d'un système de production de l'entreprise qui contient les données de l'entrepôt. Le processus d'extraction des données permet d'alimenter périodiquement ce SGBD. Néanmoins avant d'exécuter ce processus, une phase de transformation est appliquée aux données opérationnelles. Celle-ci consiste à les préparer (mise en correspondance des formats de données), les nettoyer, les filtrer,...etc., pour aboutir finalement à leur stockage dans l'entrepôt.

La figure 1.1 illustre le schéma conceptuel d'un entrepôt de données. Ce dernier stocke des données brutes ou modifiées issus de sources d'information (hétérogènes, distribuées).

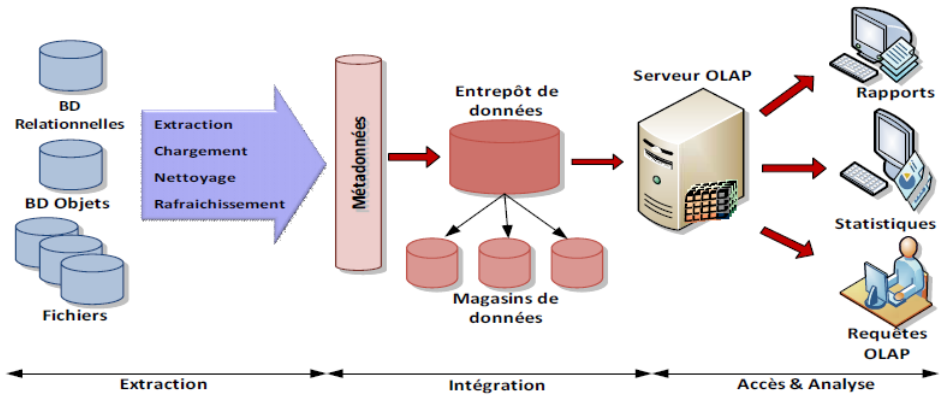

Figure 1.1: Schéma conceptuel d'un entrepôt de données

Les sources : Les données de l'entrepôt sont extraites de diverses sources souvent réparties et hétérogènes, et qui doivent être transformées avant leur stockage dans l'entrepôt. Les données sources peuvent être internes ou externe de l'entreprise. Les données internes (saisies à partir des différents systèmes de production qui rassemblent les divers SGBD opérationnels, ainsi que des anciens systèmes de production qui contiennent des données encore exploitées par l'entreprise). Les données externes de l'entreprise (Souvent fournies par une tierce partie).

L'entrepôt de données : c'est le noyau qui contient un ensemble de data mart représentant chacune une activité ou fonctionnalité d'une organisation. Il existe plusieurs types de données dans un entrepôt, qui correspondent à diverses utilisations, à savoir :

– *Données de détail courantes :* ce sont l'ensemble des données quotidiennes et plus couramment utilisées. Ces données sont généralement stockées sur le disque pour avoir un accès rapide.

– *Données de détail ancien :* correspond aux données quotidiennes concernant des événements passés, comme par exemple le détail des ventes des deux dernières années.

– *Données résumées ou agrégées* : ils sont des données moins détaillées par rapport aux deux premiers types de données. Les données résumées permettent de réduire le volume des données à stocker.

– *Les métadonnées :* Ce sont des données essentielles pour parvenir à une exploitation efficace du contenu d'un entrepôt. Elles représentent des informations nécessaires à l'accès et l'exploitation des données dans l'entrepôt comme : la sémantique (leur signification), l'origine (leur provenance), les règles d'agrégation (leur périmètre), le stockage (leur format, par exemple :dinar, euro,...) et finalement l'utilisation (par quels programmes sont-elles utilisées).

Les outils : Il existe plusieurs outils pour l'aide à la décision, on peut citer les outils de fouille de données ou datamining (pour découvrir des liens sémantiques), les outils d'analyse en ligne (pour la synthèse et l'analyse des données multidimensionnelles) et les outils d'interrogation (pour faciliter l'accès aux données en fournissant une interface conviviale au langage de requêtes).

1.2.3 Comparaison entre une base de données et un entrepôt de données

Dans l'environnement des entrepôts de données, les opérations, l'organisation des don- nées, les critères de performance, la gestion des métadonnées, la gestion des transactions et le processus de requêtes sont très différents des systèmes de bases de données opérationnels. Par conséquent, les SGBD relationnels orientés vers l'environnement opérationnel, ne peuvent pas être directement transplantés dans un système d'entrepôt de données [1].

Les systèmes OLTP (Online transaction processing) sont conçus pour la gestion des systèmes opérationnels des organisations (insertion, modification et suppression des enregistrements). Par contre, les entrepôts de données ont été conçus pour l'aide à la prise de décision. Ils intègrent les informations qui ont pour objectif de fournir une vue globale de l'information-aux analystes et aux décideurs.

Une comparaison entre les systèmes de gestion de bases de données et les entrepôts de données est présentée dans le tableau suivant :

	SGBD	**Entrepôt de données**
Objectifs	Gestion et production	Consultation et analyse
Utilisateurs	Gestionnaire de production	Décideurs, analystes
Taille de la base	Plusieurs Gigaoctets	Plusieurs Téraoctets
Organisation de données	Par traitement	Par métier
Types de données	Données de gestion	Données d'analyses
Requêtes	Simples, prédéterminées	Complexes,spécifiques
Mode d'accès	Lecture/écriture	Lecture

1.3 Modélisation multidimensionnelle

La modélisation multidimensionnelle [79] consiste à considérer un sujet analysé comme un point dans un espace à plusieurs dimensions. Les données sont organisées de manière à mettre en évidence le sujet analysé et les différentes perspectives de l'analyse. Cette modélisation multidimensionnelle a donnée naissance aux concepts de fait et de dimension. Avant de décrire les différents schémas, on commence par définir quelques concepts de base.

1.3.1 Le concept de fait

Définition 1.3.1. *Le fait modélise le sujet de l'analyse. Un fait est formé de mesures correspondant aux informations de l'activité analysée. Les mesures d'un fait sont numériques et généralement valorisées de manière continue [79]. Par exemple, dans le fait Ventes, on peut considérer la mesure "Quantité de produits vendus par magasin".*

Le sujet analysé est représenté par le concept de fait.

1.3.2 Le concept de dimension

Définition 1.3.2. *Une dimension modélise une perspective de l'analyse. Une dimension se compose de paramètres correspondant aux informations faisant varier les mesures de l'activité.*

Le sujet analysé, c'est à dire le fait, est analysé suivant différentes perspectives. Elles correspondent à une catégorie utilisée pour caractériser les mesures d'activité analysées [30], on parle de dimensions.

1.3.3 Concept hiérarchie

Définition 1.3.3. *Une hiérarchie organise les paramètres d'une dimension selon une relation "est-plus-fin" conformément à leur niveau de détail (Par exemple Ville, Département, Région).*

La hiérarchie sert lors des analyses pour restreindre ou accroître les niveaux de détail de l'analyse.

La conception et la construction d'un entrepôt de données représente un enjeu important pour la communauté des bases de données. Cette conception concerne les trois niveaux de la modélisation classique : conceptuelle, logique et physique.

1.3.4 Les modèles et les langages de modélisation

Dans le but de construire un modèle approprié pour un entrepôt de données, nous pouvons choisir, soit un schéma relationnel (schéma en étoile, en flocon de neige ou en constellation) où les données sont stockées dans un SGBD relationnel, soit un schéma multidimensionnel (Cube) où les données sont stockées dans une base de données multidimensionnelle. Ces schémas relevant d'un niveau logique de conception, en raison de recours à la notion de table (table de faits, table de dimension).

1.3.4.1 Le modèle en étoile

Un modèle en étoile (star schema) [79] représente visuellement une étoile. On parle de ce modèle, où tous les faits sont définis dans une simple table relationnelle. Cette table est reliée par sa clé primaire à d'autres tables correspondant aux dimensions. Nous illustrons ce modèle dans la figure 1.2. En fait, ce modèle essaie de superposer une structure multidimensionnelle au-dessus d'un modèle relationnel normalisé à deux dimensions. Le modèle en étoile simplifie le modèle logique normalisé en organisant les données de manière optimale pour les traitements d'analyse.

Dans la figure 1.2, on remarque que la table de faits est Ventes, et que les tables de dimension sont Client, Produit, Temps et Magasin. Ces tables sont toutes liées par une clé à la table Ventes.

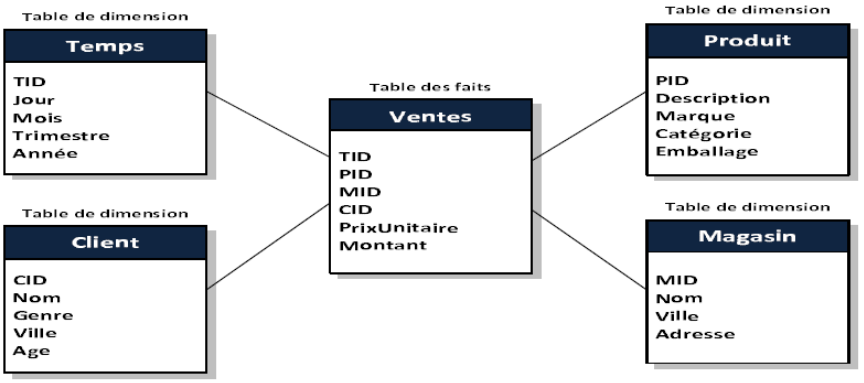

Figure 1.2: Exemple d'un schéma en étoile

1.3.4.2 Le modèle en flocon de neige

La modélisation en flocon (snowflake) est une modélisation en étoile pour laquelle nous éclatons les tables de dimension en sous-tables selon la hiérarchie de chaque dimension comme montre la figure 1.3.

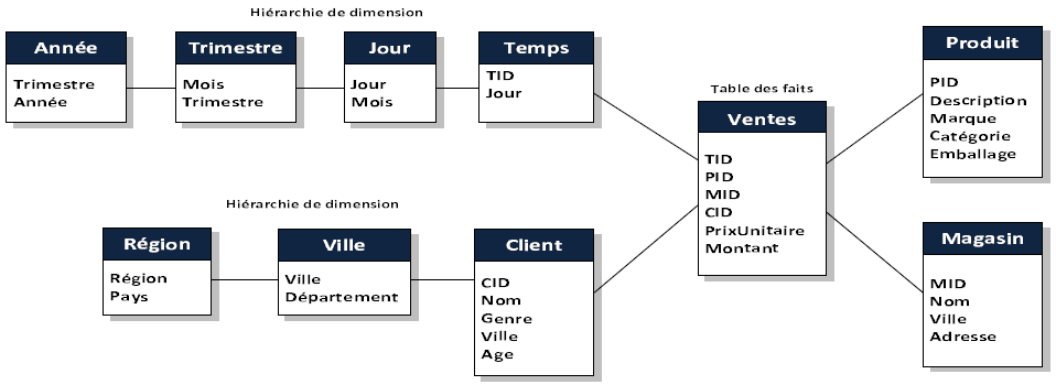

Figure 1.3: Exemple d'un schéma en flocon de neige

1.3.4.3 Le modèle en constellation

La modélisation en constellation consiste à fusionner plusieurs modèles en étoile qui utilisent des dimensions communes. Un modèle en constellation comprend donc plusieurs faits et des dimensions communes ou non.

1.3.4.4 Hypercube

Les outils OLAP reposent sur des bases de données multidimensionnelles destinées à exploiter rapidement les dimensions d'une population de données. La plupart des solutions OLAP reposent sur un même principe : restructurer et stocker dans un format multidimensionnel les données issues de fichiers plats ou de bases relationnelles [67]. Ce format multidimensionnel, connu sous le nom d'hypercube, organise les données le long de dimension comme montre la figure 1.4. Ainsi, les utilisateurs analysent les données suivant les axes propres à leur métier.

Un hypercube est un tableau à n dimensions. Chaque dimension possède une hiérarchie associée de niveaux de consolidation. Chaque position dans un tableau multidimensionnel, correspondant à une intersection de toutes les dimensions est appelée une cellule. Ces dimensions peuvent être affinées, décomposées en hiérarchies, afin de permettre à l'utilisateur d'examiner ses indicateurs à différents niveaux de détail, de descendre dans les données, allant du niveau global au niveau le plus fin.

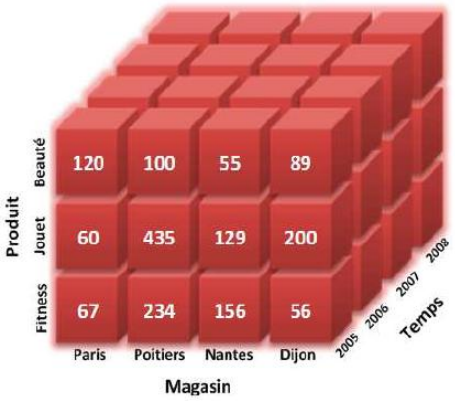

Figure 1.4: Un exemple d'un cube(Hypercube) de données

1.4 Implémentation des modèles multidimensionnels

Il existe actuellement trois approches concernant l'implantation d'un entrepôt de don- nées [79] : relationnelle OLAP (ROLAP), multidimensionnelle OLAP (MOLAP) et hybride OLAP (HOLAP).

1.4.1 L'approche ROLAP

Les données sont organisées selon des schémas relationnels spécialisés (voir figure 1.2, 1.3). Le langage de requêtes SQL et d'autres fonctionnalités des systèmes relationnels ont été adaptés à cette organisation.

1.4.2 L'approche MOLAP

Les données sont organisées sous forme de cube. Les systèmes de bases de données qui supportent cette organisation offrent un langage propriétaire pour l'interrogation et la manipulation de ces cubes.

1.4.3 L'approche HOLAP

Les données sont maintenues par un SGBD relationnel alors que les agrégations sont dans un SGBD multidimensionnel.

1.5 Niveau physique et techniques d'optimisation dans les entrepôts de données

La conception physique d'une base ou entrepôt de données consiste à établir une configuration physique sur le support de stockage. Cela comprend la spécification détaillée des éléments de données, les types de données et la sélection des techniques d'optimisation. Dans la première génération de moteurs d'exécution de requêtes dédiés aux bases de données traditionnelles, la conception physique n'avait pas autant d'importance. Aujourd'hui face à la complexité des requêtes à traiter et le volume important des données, la conception physique a reçu une importance phénoménale [81].

Le stockage de données et les méthodes d'accès associées (les techniques d'indexation) ont été étudiés depuis plusieurs années. Cependant, le volume important géré par les entrepôts et les besoins d'accès performant lors de l'interrogation ont remis en cause les techniques traditionnelles. Dans cette section nous présentons quelque technique de stockages, d'optimisations et d'accès aux données adaptées aux entrepôts de données.

1.5.1 Le stockage

Le stockage des modèles multidimensionnels (cubes) est réduit au problème de stockage d'un vecteur multidimensionnel. Dans chaque

dimension de cube est associée à un axe du vecteur et chaque valeur d'une dimension est associée à une position de l'axe correspondant à la dimension. Les cellules du vecteur, c'est-à-dire les intersections entre les positions des axes, contiennent les mesures.

Le problème principal posé dans la représentation d'un cube sous la forme d'un vecteur multidimensionnel est que celui-ci est creux. Ceci est dû au fait qu'en général seulement un nombre réduit de cellule d'un cube ont une valeur de mesure associées.

1.5.2 Les techniques d'optimisation dans les entrepôts de données

Vu la taille importante des données dans un entrepôt de données, qui rend leur interrogation lente mesurée par le temps de réponse d'une part, et d'autre part la complexité des requêtes décisionnelles qu'il l'exploite. Cette complexité est due aux opérations de jointure et d'agrégation utilisées par les requêtes, qui détériorent de manière significative les performances de l'entrepôt.

Les travaux concernant l'optimisation des performances des requêtes décisionnelles sont principalement basés sur des techniques héritées des bases de données relationnelles/objets. Plusieurs techniques d'optimisation ont été proposées dans la littérature et supportées par les systèmes de gestion de bases de données commerciaux. Ils ont été classées en deux catégories principales (voir [82, 83, 91]) : techniques redondantes et techniques non redondantes :

– Les techniques redondantes [22, 39] optimisent les requêtes, mais exigent un coût de stockage et de maintenance. Cette catégorie regroupe les vues matérialisées, les index, la fragmentation verticale, etc.

– Les techniques non redondantes ne nécessitent ni coût de stockage ni coût de maintenance. Cette catégorie regroupe la fragmentation horizontale, le traitement parallèle,
...etc.

La figure 1.5 montre une classification des principales techniques d'optimisation.

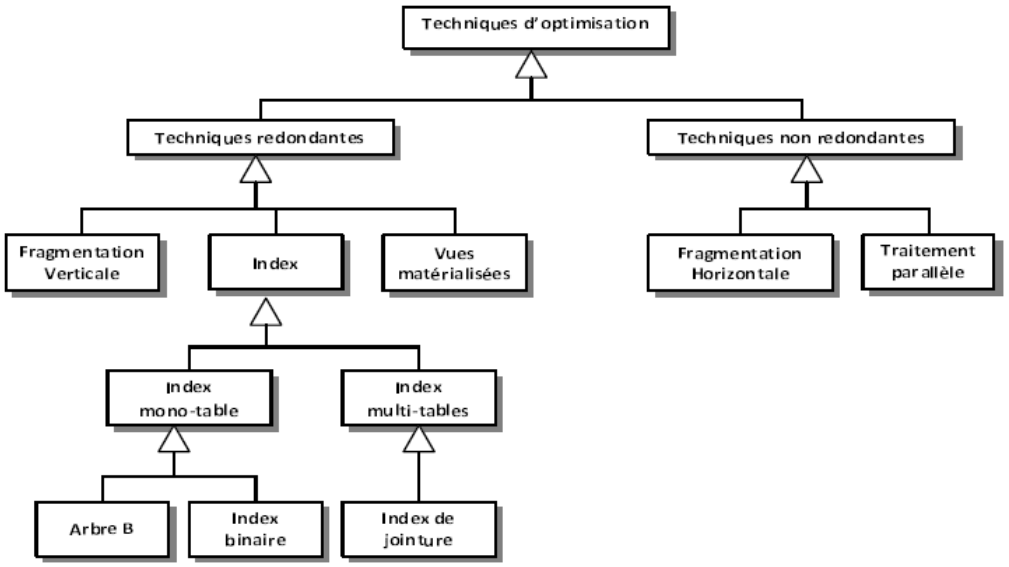

Figure 1.5: Classification des techniques d'optimisation

1.5.2.1 Les vues matérialisées

Une vue matérialisée est une table contenant les résultats d'une requête. Les vues améliorent l'exécution des requêtes en pré-calculant les opérations les plus coûteuses comme les jointures et les agrégations et en stockant les résultats dans la base de donnée.

En conséquence, certaines requêtes nécessitent seulement l'accès aux vues matérialisées et sont ainsi exécutées plus rapidement. Cependant, la mise à jour des données implique systématiquement celle des vues matérialisées calculées à partir de ces données afin de conserver la cohérence et l'intégrité des données. Cela induit une surcharge du système liée au coût de maintenance des vues matérialisées.

De plus, la matérialisation des vues requiert un espace de stockage additionnel que l'administrateur alloue à ces vues.

1.5.2.2 Les index

Interroger des tables volumineuses via un ensemble de requêtes pour accéder à un certain nombre de n-uplets est une tâche fréquente dans un environnement d'entrepôt de données.

Répondre efficacement à ces requêtes est souvent difficile compte tenu de la nature complexe des requêtes OLAP et des volumes de données. La manière la plus facile de procéder consiste à effectuer un balayage complet des tables et vérifier pour chaque n-uplet s'il satisfait le prédicat de la requête. Ce balayage peut être très coûteux lorsque les tables scannées sont volumineuses. Pour pallier à ce problème, plusieurs techniques d'indexation ont été proposées. Un index est une structure redondante ajoutée à la base de données pour permettre les accès rapides aux données. Il permet à partir d'une clé d'index de trouver l'emplacement physique des n-uplets recherchés.

Parmi les techniques d'indexation proposées dans le cadre des bases de données classiques, nous pouvons citer l'index B-tree, l'index de hachage, l'index de projection, l'index de jointure, etc. La majorité de ces index sont aussi utilisés dans le cadre des entrepôts relationnels. Certaines techniques d'indexation sont apparues dans le contexte d'entrepôts de données comme les index binaires, les index de jointure binaires, les index de jointure en étoile, etc.

Index binaire(bitmap index)

L'index binaire repose sur l'utilisation d'un ensemble de vecteurs binaires (contenant des valeurs 0 ou 1) pour référencer l'ensemble des n-uplets d'une table. Pour chaque valeur de l'attribut indexé, un vecteur de bits dit bitmap est stocké. Ce vecteur contient autant de bits qu'il y a de n-uplets dans la table indexée.

Table Client				
CID	Nom	Age	Sexe	Ville
616	Gilles	15	M	Poitiers
515	Yves	25	F	Paris
414	Patrick	33	M	Nantes
313	Didier	50	M	Nantes
212	Eric	40	F	Poitiers
111	Pascal	20	M	Poitiers

Index binaire sur l'attribut Ville		
Poitiers	Paris	Nantes
1	0	0
0	1	0
0	0	1
0	0	1
1	0	0
1	0	0

Figure 1.6: Index binaire

Actuellement, presque tous les systèmes de gestion des bases de données commerciaux fournissent des indexes binaire. L'avantage de ces index est qu'il est possible d'exécuter des opérations logiques (par exemple les opérations ET, OU, XOR, NOT) de manière performante [37]. D'autres avantages sont que, pour des attributs nombre limités de valeurs, l'espace nécessaire pour leurs stockages est réduit. Dans ce cas, l'index binaire peut être géré en mémoire, ce qui améliore les performances du système. Des techniques de compression de données sont utilisées pour gérer les indexes binaires qui contiennent presque dans leurs totalité des bits à zéro.

Index de jointure (join indices)

L'opération de jointure est toujours présente dans les requêtes OLAP. Elle est très coûteuse, puisqu'elle manipule de grands volumes de données. Plusieurs implémentations de la jointure ont été proposées dans les bases de données traditionnelles : les boucles imbriquées, les fonctions de hachage, le tri-fusion, etc. Ces implémentations sont limitées lorsque la taille des tables concernées par la jointure est importante.

Valduriez [52] a proposé un index de jointure qui pré calcule la jointure entre deux tables. L'index de jointure matérialise les liens existant entre deux tables en utilisant une table à deux colonnes chacune représentant l'identifiant d'une table. Comme montre la figure suivante.

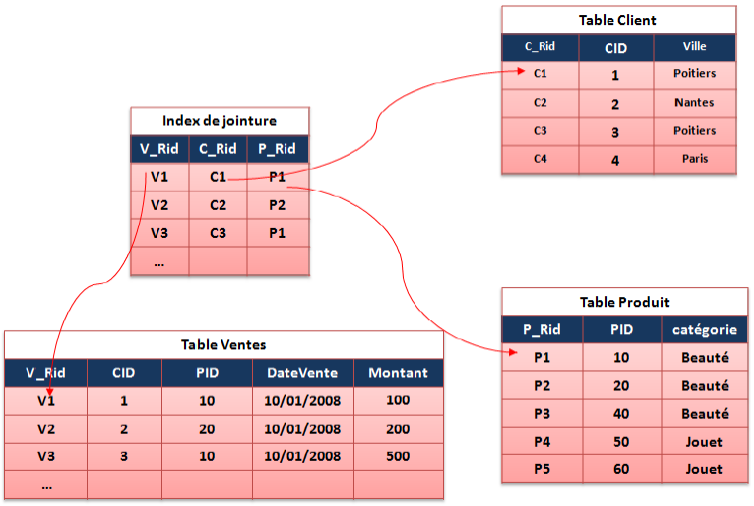

Figure 1.7: Index de jointure

Index de jointure en étoile

Dans le contexte d'entrepôts de données, les requêtes OLAP comportent généralement plusieurs jointures entre la table des faits et les tables de dimension. Par exemple pour exploiter ces index dans un entrepôt modélisé par un schéma en étoile, il faut subdiviser la requête en fonction des jointures. Effectuer cette décomposition revient à choisir un ordre de jointure, or pour N jointures, N ! ordres sont possibles (problème d'ordre de jointure). Pour pallier ce problème, Red Brick [80] a proposé un index appelé index de jointure en étoile, adapté aux requêtes définies sur les entrepôts de données modélisés en étoile. Un index de jointure en étoile peut contenir toute combinaison des identifiants des n-uplets de la table de faits et des identifiants des n-uplets des tables de dimension pouvant être jointes.

Ce type d'index est dit complet s'il est construit en joignant toutes les tables de dimension avec la table des faits. L'index de jointure en étoile complet est bénéfique à n'importe quelle requête définie sur un entrepôt modélisé en étoile.

1.5.2.3 La fragmentation

Dans la littérature, deux termes sont utilisés pour la segmentation d'une relation : partitionnement et fragmentation. Le partitionnement d'une relation

se définit par la division de cette dernière en plusieurs partitions disjointes. La fragmentation consiste en la division en plusieurs fragments (sous-ensembles de la relation) qui peuvent être non disjoints. Cependant, la plupart des chercheurs utilisent les deux termes indifféremment [35].

Dans ce mémoire, nous ne ferons pas de différence entre ces deux concepts.

La fragmentation horizontale

La fragmentation horizontale consiste à partitionner les objets de la base de données (tables, vues et index) en plusieurs ensembles de lignes appelés fragments horizontaux. Chaque ligne représente une instance de l'objet fragmenté. Les instances appartenant au même fragment horizontal vérifient généralement un prédicat de sélection. Chaque fragment horizontal T_i d'une table T est défini par une clause de sélection sur la table T comme suit :

$$T_i = \sigma_{cl_i}(T)$$

Deux types de fragmentation horizontale sont disponibles : primaire et dérivée. La fragmentation horizontale primaire d'une table se base sur les prédicats de sélection définis sur cette table. La fragmentation horizontale dérivée exploite le lien existant entre deux tables pour fragmenter l'une d'entre elles en fonction des fragments de l'autre. Par conséquent, la fragmentation horizontale dérivée d'une table se base sur les prédicats de sélection définis sur une autre table. Concrètement, la fragmentation dérivée d'une table S n'est possible que lorsqu'elle est liée avec une table T par sa clé étrangère. Une fois la table T fragmentée par la fragmentation primaire, les fragments de S sont générés par une opération de semi-jointure entre S et chaque fragment de la table T. Les deux tables seront équi-partitionnées grâce au lien père-fils.

Exemple

Soit deux tables Client et Ventes liées par une relation de clé étrangère dont les instances sont représentées dans la figure 1.8(a). La fragmentation horizontale de la table Client en trois fragments $Client_1$, $Client_2$ et $Client_3$ est illustrée dans la figure 1.8(b). Chaque fragment est défini par un prédicat de sélection sur l'attribut Ville de cette table, comme suit :

– $Client_1 : \sigma_{Ville='Poitiers'}(Client)$

– $Client_2 : \sigma_{Ville='Paris'}(Client)$

– $Client_3 : \sigma_{Ville='Nantes'}(Client)$

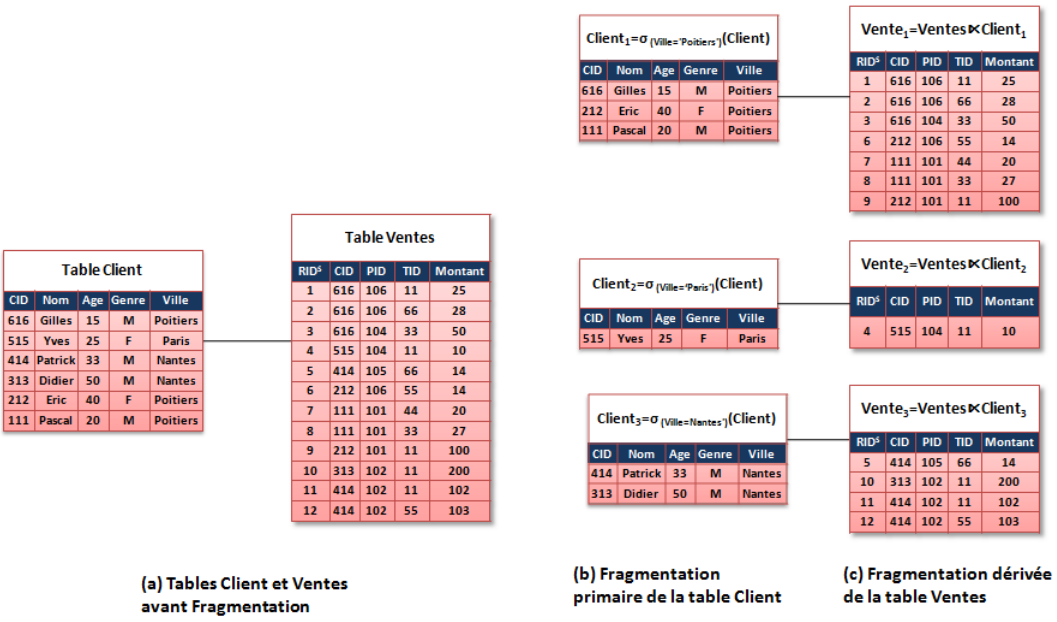

Figure 1.8: Exemple d'une fragmentation primaire et dérivée

La figure 1.8(c) représente la fragmentation dérivée de la table Ventes en trois fragments en fonction des trois fragments de la table Client. Chaque fragment de la table Ventes est généré à l'aide d'une opération de semi-jointure entre un fragment de la table Client et la table des faits comme suit :

– $Ventes_1 = Ventes \ltimes Client_1$

– $Ventes_2 = Ventes \ltimes Client_2$

– $Ventes_3 = Ventes \ltimes Client_3$

La fragmentation verticale

Comme nous l'avons déjà définie, la fragmentation verticale d'une relation R répartie les attributs de la relation afin de générer plusieurs tables $R_1, R_2, .., R_n$, contenant chacun un sous-ensemble des attributs de R en plus de la clé primaire de R. Cette technique est redondante car elle duplique la clé primaire de la table fragmentée, plus la duplication d'autres attributs selon le besoin de fragmentation. Elle nécessite donc un espace de stockage supplémentaire. La fragmentation verticale accélère les opérations de projection.

En effet, le traitement de la requête de projection nécessite uniquement le chargement des sous relation dont les attributs figurent dans la requête, il n'est pas nécessaire de charger toute la table. Par contre, son principal inconvénient est quelle nécessite des opérations de jointures très coûteuses si une requête accède à plusieurs fragments verticaux [91].

Exemple

Soit une table *Client*. Le résultat de fragmentation verticale de cette dimension est présenté dans la figure 1.9.

Table Client			
Id	Nom	Age	Genre
1	Ali	30	M
2	Kassi	25	F
3	Lee	15	M
4	Jones	45	M

Fragmentation Verticale →

Client1		
Id	Nom	Genre
1	Ali	M
2	Kassi	F
3	Lee	M
4	Jones	M

Client2	
Id	Age
1	30
2	25
3	15
4	45

Figure 1.9: Exemple de fragmentation verticale

Un problème qui se pose pour la fragmentation est comment définir un bon degré de fragmentation. Il existe trois règles pour la fragmentation :

1. Complétude : pour toute donnée d'une relation globale R, il existe au moins un fragment R_i de la relation R qui possède cette donnée.
2. Reconstruction : pour toute relation R décomposée en un ensemble de

fragments $R_{i,}$ il existe une opération de reconstruction à définir en fonction de la fragmentation. Pour les fragmentations horizontales, l'opération de reconstruction est l'union. Pour les fragmentations verticales c'est la jointure.

3. Disjonction : une donnée n'est présente que dans un seul fragment, sauf dans le cas de la fragmentation verticale pour la clé primaire qui doit être présente dans l'ensemble des fragments issus d'une relation.

Comparaison entre les techniques d'optimisation

Le tableau présenté dans la figure 1.10 montre une comparaison entre les techniques d'optimisation dans les entrepôts de données suivant plusieurs critères : la nécessité d'un espace supplémentaire de stockage, le besoin ou pas d'un coût de maintenance, la transparence d'utilisation de la technique pour l'utilisateur de l'entrepôt, le changement du schéma des relations de base et enfin l'apport de chaque technique dans l'optimisation des entrepôts de données.

Nous remarquons que les index, vues matérialisées et fragmentation verticale nécessitent un espace de stockage puisque ce sont des techniques redondantes. Un coût de maintenance existe uniquement pour les index et vues. Pour ce qui est de la fragmentation (horizontale ou verticale) dans les ED, les opérations les plus fréquentes sont les sélections et insertions qui ne nécessitent pas une maintenance du schéma de fragmentation.

Technique	Espace de stockage	Coût de maintenance	Transparence à l'utilisateur	Changement du schéma	Type d'optimisation
Index	X	X	X	-	jointures en étoile Opérations de sélection
VM	X	X	X	-	Les requêtes les plus fréquentes
FV	X	-	X	X (altère les relations)	Les requêtes de projection
FH	-	-	X	-	Les jointures en étoile Opérations de sélection

Figure 1.10: Tableau comparatif des techniques d'optimisations dans les ED

Il suffit de détecter le fragment adéquat pour les nouvelles données et les insérer. Certaines techniques provoquent un changement dans le schéma physique initial comme la fragmentation verticale. Cette fragmentation restructure une relation en plusieurs sous relations, chacune contient un sous ensemble d'attributs de la relation initiale. Par contre, les utilisateurs d'un ED doivent pouvoir exécuter leurs requêtes sans se soucier de la représentation physique des données. Il faut prendre en compte l'adaptation des requêtes sur le schéma de données (réécriture des requêtes pour les vues, jointures dans le cas de fragmentation verticale) afin de garantir la transparence d'accès aux données pour l'utilisateur.

1.6 Problèmes de sélection des techniques d'optimisation

La conception d'une démarche de sélection des techniques d'optimisation doit assurer la réduction du coût d'exécution des opérations ou les requêtes les plus complexes. Cette complexité peut être évaluée par la durée d'exécution de la requête (estimée en secondes, heures ou jours dans les entrepôts de données) ou par le nombre d'entrées/sorties entre disque et mémoire vive. En effet, afin d'exécuter une requête, il faut charger, dans la mémoire vive, les données stockées sur disque afin d'effectuer les opérations nécessaires à l'exécution de la requête (jointures, sélection, restrictions, etc.) [85].

La conception physique d'un entrepôt ou base de données passe en deux phases importantes :

- La sélection d'une stratégie d'optimisation : qui consiste à choisir, parmi la pano- plie des techniques d'optimisation, celles qui optimisent aux mieux les traitements exécutées (choisir les index à implémenter, sélectionner le schéma de fragmentation des tables, etc.)
- L'implémentation physique des techniques d'optimisation sélectionnées. Elle représente la création physique des structures préalablement

sélectionnées. (Création des index, matérialisation des vues, fragmentation effectives des tables).

Dans les bases de données classiques, ces deux phases de conceptions physiques sont à la charge de l'administrateur de base de données. Elles sont réalisées suivant l'expérience et le bon sens de celui-ci. Par contre, dans le contexte d'entrepôt de données, la tâche d'administration devient de plus en plus difficile et l'administrateur doit être assisté par des outils qui procurent les bonnes recommandations pour la bonne conception et la bonne optimisation d'un ED. Devant la difficulté de l'administrateur à choisir une stratégie d'optimisation, plusieurs travaux se sont penchés sur l'établissement, la mise en œuvre et l'automatisation de stratégies de sélection des techniques d'optimisation qui doivent prendre en compte les choix d'optimisation exprimés par l'administrateur (choix des tables, des techniques, des attributs, etc.). Souvent, un concepteur d'une démarche de sélection à pour rôle de concevoir un outil d'assistance qui va permettre de saisir les besoins en optimisation de l'administrateur et de lui proposer un schéma d'optimisation pouvant réduire le coût d'exécution d'une charge de requêtes.

Ainsi, toute démarche de sélection d'un schéma d'optimisation possède deux grands axes : le premier axe concerne le rôle de l'administrateur dans le choix des paramètres d'une stratégie de sélection. Le second axe concerne le rôle du concepteur d'une démarche de sélection dans la mise en œuvre de cette stratégie.

1.6.1 Rôle de l'administrateur de bases ou entrepôts de données

Afin de mettre en œuvre une stratégie d'optimisation, l'administrateur (de bases ou entrepôts de données) doit prendre plusieurs décisions pertinentes (voir 1.11) :

– Choisir les techniques d'optimisation à implémenter (index, vues, fragmentation, etc.).
– Décider la nature de sélection : isolée où une seule technique est choisie, combinée où plusieurs techniques sont sélectionnés à la fois.

– Choisir l'algorithme de sélection qui va permettre de sélectionner les structures d'optimisation à implémenter sur l'entrepôt ou la base de données (algorithmes simples, heuristiques, algorithmes génétiques etc.).
– Choisir les tables et les attributs du schéma de données sur lesquels les structures vont être définies.

1.6.1.1 Nature de sélection

Il existe deux façons de sélection des techniques d'optimisation [85] : la sélection isolée et celle combinée.

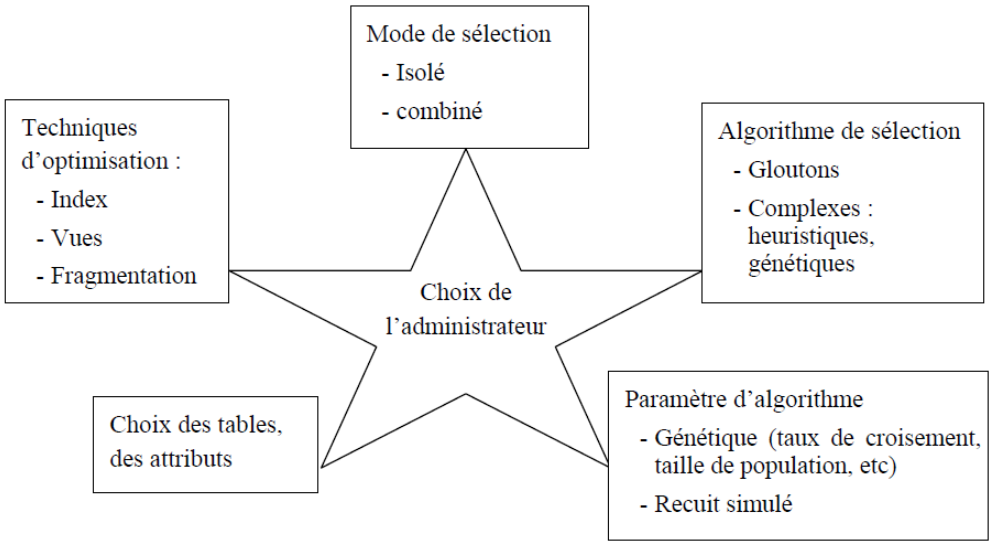

Figure 1.11: Choix de l'administrateur pour une stratégie d'optimisation

– La sélection isolée consiste à choisir d'implémenter une seule technique d'optimisation (redondante ou non redondante) indépendamment de toute influence ou similarité avec une autre technique.

– La sélection multiple, elle permet de réaliser l'implémentation conjointe de deux ou plusieurs techniques qui disposent du même espace de stockage et du même algorithme d'implémentation. Dans ce cas l'influence et l'interaction entre les techniques sont prises en compte pour une meilleure

optimisation et exploitation des ressources.

1.6.1.2 Types d'algorithmes de sélection

Etant donné une structure d'optimisation, plusieurs configurations sont possibles pour optimiser les données. Pour les index par exemple, on peut créer autant d'index que d'attribut existants dans le schéma des tables. Puisque les index nécessitent un espace de stockage, on ne peut pas tout indexer. Il faut impérativement choisir les index qui permettent d'optimiser au mieux les traitements et requêtes exécutés. Afin d'automatiser la sélection des techniques d'optimisation, plusieurs algorithmes de résolutions sont utilisés pour choisir une configuration de structures optimale. Ce sont des algorithmes approximatifs classifiés en deux catégories : les algorithmes simples ou gloutons et les algorithmes complexes comme les heuristiques ou algorithmes de Datamining [90].

– Les algorithmes gloutons : Ces algorithmes sont utilisés pour choisir la configuration de structures optimale. Cette dernière est obtenue en effectuant une suite de choix. A chaque étape de l'algorithme, le choix qui semble le meilleur est effectué. Ce choix

– Les algorithmes complexes : Contrairement aux algorithmes exacts, qui permettent de trouver la configuration optimale à un problème mais dont la complexité est exponentielle, les algorithmes complexes regroupent les algorithmes non exhaustifs qui permettent de trouver une solution quasi optimale (acceptable) en un temps raisonnable. Parmi ces algorithmes on trouve les heuristiques (Hill Climbing, Recuit Simulé, etc.) les algorithmes génétiques ou les algorithmes de fouilles de données. Généralement ces algorithmes démarrent d'une ou plusieurs solutions initiales et visent à les améliorer par application d'opérations (Split and Merge pour le Hill Climbing. Sélection, croisement, mutation pour les génétiques). Certains algorithmes ont pour handicape la possibilité de choisir un optimum local comme solution optimale (Hill Climbing). Ainsi, des algorithmes comme le Recuit simulé permettent d'admettre de mauvaises solutions ou des solutions

inadmissibles, avec une certaine probabilité, afin de sortir des optimums locaux.

1.6.2 Rôle du concepteur de la stratégie de sélection des techniques d'optimisation

Le concepteur doit mettre en œuvre une stratégie qui permet de sélectionner les structures d'optimisation, en prenant en compte les décisions prise par l'administrateur, puis d'implémenter ces structures sur les données physiques.

Pour ce faire, le concepteur doit formaliser le problème de sélection des structures d'optimisation afin d'identifier les contraintes sur la sélection (par exemple la contrainte d'espace de stockage pour les index et vue matérialisées) et pouvoir proposer les algorithmes de sélection les plus adéquats (si le problème est NP-Complet par exemple, il faut employer les algorithmes complexes afin d'obtenir un bon résultat). Concernant l'implémentation des algorithmes de sélection, il faut prendre en compte l'optimisation de la charge de requêtes. Chaque algorithme de sélection doit permettre de sélectionner les structures d'optimisation qui optimisent le coût d'exécution de cette charge. Ce coût est évalué par modèle de coût qui permet de guider la sélection des structures afin de choisir la configuration optimale.

Nous présentons dans ce qui suit, la formalisation générale d'un problème de sélection puis le principe de modèle de coût.

1.6.2.1 Formalisation générale d'un problème de sélection

Etant donné une charge de requêtes Q (récupérée généralement à partir du journal de transaction du SGBD ou spécifiée par l'administrateur) et une ou plusieurs contraintes d'optimisations C (espace de stockage, coût de maintenance, temps d'exécution). Il faut trouver une configuration de structures physiques en un temps raisonnable, qui, une fois implémentées, permettent de réduire le coût d'exécution des requêtes Q sans violer la contrainte C [90].

1.6.2.2 Modèle de coût

Nous allons présenter dans ce qui suit, le principe, les domaines d'utilisations ainsi que les méthodes d'estimation d'un modèle de coût.

a) Principe et domaines d'utilisation : Le modèle de coût permet d'estimer le coût d'exécution d'une requête en se basant sur plusieurs paramètres physiques qui sont :

– La taille des tables, des tuples et le temps d'accès aux données qu'elles soient en mémoire centrale ou en mémoire secondaire.

– Le facteur de sélectivité d'une opération qui représente le rapport entre le nombre de tuples sélectionné par l'opération et la taille globale de la table.

– L'implémentation physique de chaque opération : jointure, sélection etc. Par exemple, pour la jointure, il faut préciser si c'est une jointure par boucles imbriquées (NES- TED LOOP), tri-fusion (SORT-JOIN) ou par hachage (HASH-JOIN) [91].

– Données système comme la taille d'une page système.

Le modèle de coût peut être utilisé dans plusieurs cas [91] :

– Choisir le meilleur plan d'exécution pour une requête en incluant les algorithmes d'implémentations des opérateurs. Le plan dont le coût optimal est sélectionné.

– Guider les algorithmes de sélection des techniques d'optimisation. Le modèle de coût permet d'évaluer chaque configuration générée afin de choisir la plus optimale.

b) Méthodes d'évaluation : Le modèle de coût peut être évalué de deux manières : la première fait appel à l'optimiseur du SGBD. La seconde évalue le modèle de coût par une fonction mathématique.

1) Le modèle de coût basé sur appel d'optimiseur a l'avantage d'être précis car il se base sur l'estimation réelle de la taille des tables, des données

chargées, de la taille des tuples etc. Son principal inconvénient et qu'il rend la technique de sélection dé- pendante du SGBD. De plus, les appels fréquents de l'optimiseur peuvent dégrader les performances.

2) Le modèle de coût basé sur une fonction mathématique a l'avantage de rendre la stratégie d'optimisation rapide et complètement indépendante du SGBD. Par contre, il nécessite l'utilisation d'hypothèses qui peuvent ne pas refléter exactement la réalité d'exécution des requêtes. Afin de mettre en œuvre un modèle de coût mathématique il faut :

– Identifier les algorithmes d'implémentation des opérateurs.

– Estimer les paramètres de coût : ils désignent des statistiques et des estimations sur la sélectivité des prédicats, les tables (nombre de tuples d'une table, taille de tuples. . .), sur le système (taille d'une page système, le coût de chargement) et enfin des estimations sur les techniques d'optimisation utilisées (espace de stockage, coût de maintenance).

c) Intérêt d'utiliser un modèle de coût : Dans la plus part des SGBD, le modèle de coût est employé pour choisir le meilleur plan d'exécution d'une requête. En effet, ce modèle permet de prendre en compte la représentation des données sur lesquelles s'exécutent les requêtes de traitements. L'exécution d'une requête représente une succession d'opérations de sélections, jointures projections, etc. appelée plan d'exécution. Afin de sélectionner le meilleur plan, l'optimiseur d'un SGBD utilise deux méthodes : soit par *Rule Based* ou par *Cost Based* :

1. Le Rule Base est basée sur le principe de réécriture de la requête en utilisant l'algèbre relationnelle. La requête représente un ensemble de tables sur lesquelles sont appliqués des opérateurs algébriques comme la jointure, la sélection, la projection et les opérateurs ensemblistes. Cette requête peut être illustrée sous forme d'arbre algébrique. Afin d'optimiser la requête, on effectue des modifications sur l'arbre suivant plusieurs règles comme exécution le plus tôt possible les opérations de sélection afin de réduire la taille des tables à manipuler. L'inconvénient principal du Rule Based est qu'il est complètement indépendant de

l'implémentation physique des données ou des opérateurs (jointure, sélection etc.). Ainsi, la pluparts des SGBD emploie l'optimisation par modèle de coût (Cost Based).

2. Le Cost Based est basée sur l'évaluation de l'exécution d'une requête en exploitant un modèle de coût. L'optimiseur établit un plan d'exécution qui représente un arbre algébrique. La différence avec le Rule Base est qu'au niveau de chaque nœud, l'algorithme d'implémentation de chaque opérateur est précisé. Ce plan est évalué en calculant le coût d'exécution. Le plan choisi par la requête est de coût minimal.

1.7 Conclusion

Nous avons présenté dans ce chapitre les concepts des entrepôts de données et les techniques d'optimisation des requêtes les plus utilisées dans ce contexte. Les principales caractéristiques des entrepôts de données sont leur grande taille et la complexité des requêtes décisionnelles dues aux opérations de jointure et d'agrégation. Plusieurs techniques d'optimisation ont été proposées pour réduire le coût d'exécution de requêtes. Ces techniques peuvent être divisées en deux catégories : structures redondantes et structures non redondantes. Les structures redondantes nécessitent un espace de stockage et un coût de mises à jour. La deuxième catégorie ne nécessite pas de coût de stockage.

Nous allons voir dans le chapitre suivant un état de l'art sur la fragmentation verticale

CHAPITRE 2
La fragmentation verticale

2.1 Introduction

La fragmentation verticale (aussi appelée partition des attributs) est un problème de clustering d'attributs d'une relation en fragments dans le but de minimiser le temps d'exécution des requêtes. Par rapport à d'autres types de fragmentation des données, la fragmentation verticale est plus compliquée en raison de l'augmentation du nombre d'alternatives possibles [35].

Dans ce chapitre, nous allons mener une étude bibliographique portant sur les travaux qui traitent la fragmentation verticale. Cette étude présente la technique de fragmentation verticale dans le contexte des bases de données relationnelles centralisées, distribuées, objets, et les entrepôts de données.

2.2 État de l'art

Le concept d'utilisation de la fragmentation des données comme un moyen d'amélioration des performances des systèmes de gestion des bases de données a souvent apparu dans la littérature dans les systèmes des fichiers et l'optimisation des requêtes. La fragmentation verticale a été appliquée dans les bases de données relationnels centralisées [18, 28, 15, 10, 6, 20, 21, 40], distribuées [3, 48, 41, 42, 39], orientées objet [37, 53, 54] et la conception des entrepôts de donnés [10, 53, 54, 55].

2.2.1 Fragmentation verticale (relationnelles centralisées)

La technique de groupement des attributs a été reconnue pour la première fois par McCormick [18] avec son algorithme dit Bond Energy Algorithm (BEA). L'algorithme de BEA est une heuristique pour réorganiser les colonnes et les

lignes d'une matrice de sorte que chaque entrée est plus prés numériquement possible à ses quatre voisins. L'algorithme prend en entrée la matrice d'affinité des attributs Aff (c'est une matrice de N × N où N est le nombre d'attributs et la valeur $Aff_{i,j}$ représente la fréquence d'accès des requêtes entre l'attribut i et j) et il génère une matrice d'affinité groupée (voir la section 3.1 pour plus de détails).

Les permutations sont effectuées en maximisant la formule de mesure d'affinité globale suivante :

$$AM = \sum_{i=0}^{n} \sum_{j=0}^{n} Aff_{ij} [Aff_{i,j-1} + Aff_{i,j+1} + Aff_{i-1,j} + Aff_{i+1,j}]$$

Où

$$Aff_{0,j} = Aff_{i,0} = Aff_{n+1,j} = Aff_{i,n+1} = 0$$

La matrice résultante est sous forme de bloc diagonale où chaque bloc d'attributs peut représenter un fragment. L'algorithme de BEA ne donne pas le nombre de fragments et les attributs qu'ils contiennent.

Stocker et Dearnley [8],[28] ont présenté une mise en œuvre d'un système de gestion de base de données auto organisationnel qui effectue le groupement des attributs. Ils ont montré également que dans un système de gestion de base de données, quand le coût de stockage est inférieur au coût d'accès au sous fichiers, il est important de grouper les attributs, vu que l'augmentation dans le coût de stockage est plus que la compensation par des économies en coût d'accès.

Kennedy [14],[15] a définit un modèle mathématique de partitionnement des attributs, où chaque attribut a_i a une taille connue et une probabilité p_i d'être utilisée par une requête donnée. La probabilité commune que l'attribut a_i et a_j soient utilisés par la même requête est supposée ($p_i.p_j$). Une fonction de coût sur la base de cette hypothèse est dérivée qui représente la quantité de

données qu'elle doit être transmise afin de répondre à la requête. L'objectif dans cette approche est de choisir un schéma de partitionnement tel que cette fonction de coût est minimisée.

Hoffer et Severance [11] regroupent les attributs d'une relation dans lesquelles ils sont utilisés simultanément en basant sur une mesure d'affinité entre deux paires d'attributs. Le regroupement des attributs est basé sur leurs affinités et il est réalisé avec l'utilisation de l'algorithme de BEA [18].

Hoffer [12] a développé un programme non linéaire en variables bivalentes (0-1) qui minimise une combinaison linéaire des coûts de stockage, de suppression et de mise à jour avec des contraintes de capacités pour chaque fichier.

Eisner et Severance [9] ont proposé de partitionner un fichier en deux sous fichiers : un sous fichier primaire et un autre secondaire. Deux formes de fonctions de coûts ont été utilisées dans cette approche : la première fonction est la somme des charges de stockage pour les sous tuples dans le sous fichier primaire, et le coût d'accès à tous les sous tuples résidant dans le sous fichier secondaire. La deuxième fonction est non linéaire, et elle mesure le totale des coûts d'accès, de transfert et de stockage pour les sous tuples dans le sous fichiers primaire et secondaire.

La limitation de cette approche est qu'au plus deux sous fichiers sont autorisées, et les coûts associés au traitement d'une requête est supposé être le coût d'accès à la totalité du sous fichier (primaire ou secondaire) dans l'ensemble plutôt que le coût de la récupération des sous-tuples du sous fichier réellement nécessaire pour répondre à la requête seulement [16].

March et Severance [21] ont étendu le modèle de [9] pour intégrer des facteurs de blocage pour la mémoire primaire et secondaire. La taille des pages dans le sous fichier primaire et secondaire des sous tuples ne sont pas nécessairement les mêmes, mais une contrainte a été imposée est que la somme des tailles des pages de sous fichier primaire et secondaire est constante. La fonction objective non linéaire de coût qu'ils ont proposée ne dépend pas

seulement de la manière dont les attributs sont partitionnés entre les deux sous fichiers, mais aussi de la taille des pages sélectionnées pour chacun des sous fichiers primaire et secondaire. L'inconvénient de cette approche est quelle ne contient pas un modèle précis du coût d'accès aux sous tuples sélectionné dans les requêtes. Par contre, le sous fichier primaire et secondaire sont censés d'être accessibles dans leurs totalités à chaque fois où l'un de leurs attributs est demandé par une requête.

Dans [10], Hammer et Niamir ont mis au point deux heuristiques, le groupement et le regroupement pour effectuer la fragmentation verticale. L'heuristique de groupement commence initialement à assigner chaque attribut à une partition triviale et elle pro- duit toutes les partitions qui peuvent être obtenues en groupant les paires des blocs des partitions triviales, l'heuristique évalue ensuite toutes les partitions générées avec un estimateur des coûts, et elle trouve le schéma de partitionnement dont la performance du coût est minimale par rapport à tous les schémas générées. A chaque itération, tout groupement possible de ces partitions est considéré et l'un avec le maximum d'amélioration est choisi comme un groupement candidat pour la prochaine itération. Au cours de regroupement, les attributs sont déplacés entre les partitions pour atteindre d'autres améliorations possibles.

Navathe et al [20] ont étendu le travail de [11]. Ils ont proposé une méthode de fragmentations verticale par répartition des attributs basée sur une approche de regroupe- ment binaires. Cette répartition se base sur une relation à m attributs à fragmenter et un ensemble de n requêtes les plus fréquentes. Les auteurs ont utilisé dans leur travail une matrice d'affinité triée avec l'algorithme de BEA [18], cependant, la détermination des fragments verticaux se fait automatiquement. L'algorithme proposé est basé sur une approche de fragmentation verticale en deux phases. L'algorithme commence par la création de la matrice d'affinité des attributs de la base de données, et il utilise la matrice d'usage comme paramètre d'entrée :

– Dans la *première étape*, l'algorithme BEA est appliqué pour réorganiser les lignes et les colonnes de la matrice d'affinité afin de maximiser la valeur

de la fonction d'affinité globale. La matrice résultat (Matrice d'affinité groupée) devient une entrée pour la *deuxième phase* nommée l'étape de partitionnement binaire vertical (Binary Vertical Partioning-BVP)

Cette phase partitionne récursivement la matrice groupée en deux parties dans le but de minimiser le nombre de transactions qui accèdent aux deux parties. Une fonction objective est utilisée pour guider la génération des fragments verticaux de façon itérative et binaire.

– Dans la *deuxième phase* l'algorithme prend en considération des facteurs d'estimation de coût qui reflétant l'environnement physique de stockage des fragments afin de raffiner le schéma de la fragmentation. L'inconvénient majeur dans cette approche est que le résultat de fragmentation contient seulement deux fragments.

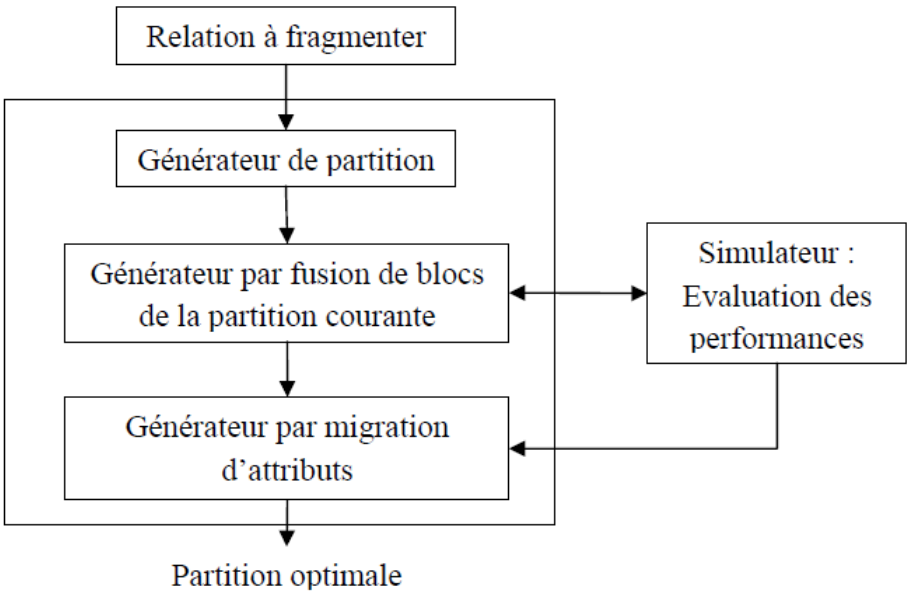

Figure 2.1: Fragmentation verticale par Hammer et al

Cornell et Yu [6] ont proposé un algorithme comme une extension de l'approche de Navathe et al [20] et ils ont appliqué ce dernier dans la conception physique des bases de données relationnelles. L'algorithme est basé sur les méthodes de programmation en nombre entiers et il nécessite des connaissances sur plusieurs facteurs physiques concernant la relation tels que le nombre d'attributs, leurs tailles et sélectivités ainsi que la cardinalité de la relation,..., etc. Leur approche s'appuie sur la décomposition du processus de

conception en sous problèmes de conception et elle n'a pas fournit des améliorations algorithmiques dans le processus de fragmentation verticale lui-même [22].

Navathe et Ra [22] ont présenté un algorithme de fragmentation verticale basé sur les concepts de la théorie des graphes, où les nœuds correspondent aux attributs et les arrêtes correspondent à la valeur d'affinité entres les attributs. Leur approche est basée sur une observation où toutes paires d'attributs dans un fragment doivent avoir une affinité élevée dans le même fragment, mais une faible affinité entre les attributs des autres fragments.

L'algorithme commence à partir de la matrice d'affinité des attributs, qui est transformée en un graphe nommé Graphe d'affinité. Il forme un arbre de recouvrement linéairement connecté et il génère tous les fragments dans une seule itération en considérant un cycle comme un fragment. Le but est de trouver les composantes connexes du graphe où chaque composante est un fragment. La réduction dans la complexité de l'algorithme a été visée comme le principal avantage dans cette approche.

Chu et Ieong [3] ont développé une approche basée sur les transactions pour la fragmentation verticale, dans laquelle ces transactions sont utilisées comme unité d'analyse plutôt que l'attribut. Leur approche permet l'optimisation du partitionnement qui est basé sur un ensemble de transactions sélectionnées importantes. L'algorithme de partitionnement binaire optimal (Optimal Binary Partitionin algorithm, OBP) proposé par Chu et Ieong est basé sur une méthode de type Séparation et évaluation (branch and bound) [65] et il est présenté avec le pire des cas avec une complexité de O(2n) où n est le nombre de transactions. Une fonction de coût est appliquée afin de déterminer le partitionnement binaire. L'algorithme OBP possède deux inconvénients : le nombre de partitions à examiner en vue de trouver le meilleur schéma de fragmentation binaire peut être élevé, et le résultat de fragmentation contient seulement deux groupes.

Gorla et Song [40] ont utilisé les algorithmes génétiques pour obtenir des solutions à la fois pour la fragmentation verticale et pour les chemins d'accès de ces fragments. Ils ont également utilisé le nombre d'accès disque comme un critère d'évaluation de partitionnement.

2.2.1.1 Synthèse

La plupart des algorithmes cités précédemment emploient des itérations multiples de partitionnement binaire pour trouver une schéma de fragmentation verticale [6, 11, 20,40]. Des efforts ont également été faits pour utiliser d'autres techniques d'optimisation pour bénéficier de la fragmentation verticale [4],[40].

L'algorithme de BEA, partitionnement binaire vertical, graphes, algorithmes génétiques, …etc., sont des méthodes qui diffèrent dans l'approche de fragmentation.

La plupart de ces algorithmes proposent de partitionner les attributs à la base des fréquences des requêtes, mais les résultats de fragmentation des algorithmes présentés ne sont pas totalement similaires.

L'algorithme proposé par Navathe et Ra [22] est jugé intéressant par rapport aux autres en vue de complexité (une approche basé sur la théorie des graphes) et de nombre d'itérations (une seule itération pour avoir le schéma de fragmentation).

2.2.2 Fragmentation verticale (relationnelles distribuées)

Dans la conception de bases de données réparties, la technique de fragmentation verticale consiste à trouver un schéma de partitionnement qui puisse réduire le coût de traitement des transactions en augmentant le traitement local de transactions (sur un site), ainsi par la réduction de nombre d'accès aux données qui ne sont pas locales (sur les sites distants).

Ceri, Pernici et Wiederhold [50] ont proposé deux outils pour la fragmentation verticale : DEVIDE et CONQUER. L'outil DEVIDE implémente l'algorithme proposé dans le travail de Navathe et al [20] et il effectue seulement

la fragmentation et l'allocation des fragments aux sites. Quant à l'outil CONQUER, il effectue l'opération de fragmentation et l'allocation, de plus il assure l'optimisation et l'allocation des opérations.

Charkavarthy et al [48],[49] ont révélé que les algorithmes antérieures pour la fragmentation verticale sont ad hoc. Ils ont proposé une nouvelle fonction objective nommée Partition Evaluator (PE) pour déterminer la qualité des schémas de partitionnements générés par les différents algorithmes de fragmentation verticale. L'évaluateur de partitionnement (PE) à deux termes nommés : le coût d'accès aux attributs locaux non pertinents (les attributs locaux qui ne sont pas accessible par la transaction - irrelevant local attribute access cost) et le coût d'accès aux attributs distants pertinents (i.e. les attributs non locaux accessibles par une transaction - relevant remote attribute access cost). Le terme d'accès aux attributs locaux non pertinents mesure le coût de traitement local des transactions qui est dû à des attributs non importants. Le terme d'accès aux attributs distants pertinents mesure le coût de traitement distant qui est dû à des attributs pertinents. Les deux composants d'évaluateur de partitionnement sont sensibles au nombre de partition générés. Le cas idéal pour l'évaluateur de partitionnement est résumé dans la figure suivant :

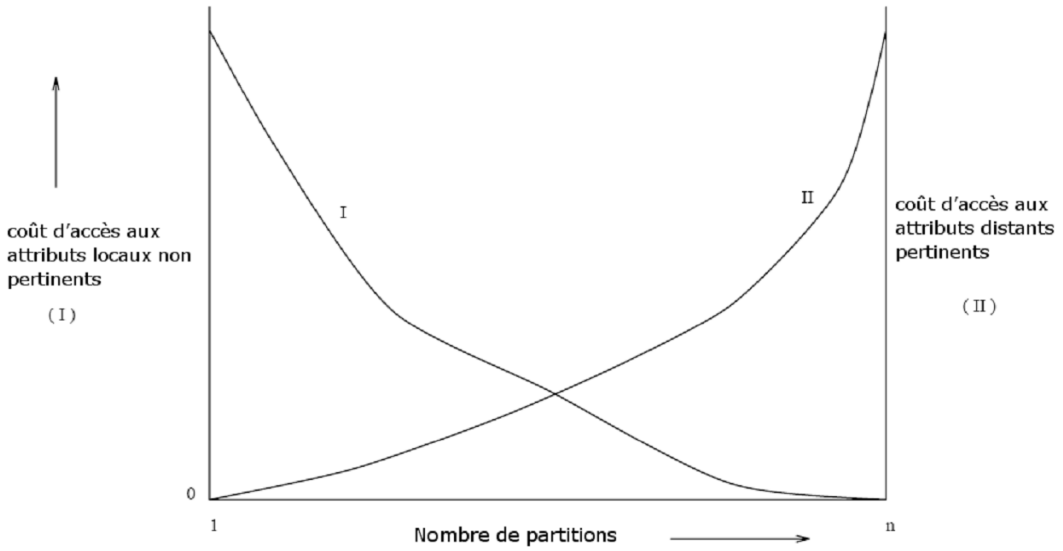

Figure 2.2: Evolution des coûts en fonction du nombre de fragments

La fonction objective utilise les données de la matrice d'affinité pour les deux termes. L'objectif dans cet évaluateur de coût est de maximiser la somme des deux termes.

E. Hartuv et R.Shamir [41] ont proposé une technique de clustering basée sur les graphes dont le but est de partitionner les données des expressions des gènes dans le domaine de la bioinformatique. L'algorithme proposé est polynomial, un graphe de similarité est définit et les classes correspond aux sous-graphes dont la connectivité de leurs arrêtes dépassent le nombre de sommets (attributs). Le principal inconvénient de cet algorithme est que la fréquence des requêtes n'est pas prise en compte, si deux attributs sont utilisés dans des requêtes plus que le seuil, ils seront liés dans le graphe. Cela implique que les arêtes du graphe sont non pondérées et tous les liens de similarités sont considérés égaux.

Ozsu et Valduriez [35] ont étudié la fragmentation verticale dans les bases de données distribués en utilisant des fréquences d'accès et l'algorithme de Bond Energy [18]. Les groupes d'attributs sont partitionnés et des équations de coût sont utilisées pour définir le meilleur schéma de partitionnement au long de la matrice d'affinité groupée afin de diviser les relations dans des fragments.

Cheng, Lee et Wong [42] ont utilisé les algorithmes génétiques (AG) et les algorithmes de clustering basés sur le problème de voyageur de commerce afin d'obtenir un schéma de fragmentation verticale.

En 2006, J.Du R.Alhajj et K.Barker [39] ont proposé un AG pour résoudre le problème de fragmentation verticale. Ils ont appliqué la contrainte des chaînes RG (Restricted Growth) [61] pour manipuler les chromosomes de sorte que les chromosomes redondantes sont exclus pendant le processus de AG. Le codage en chaînes RG représente un ensemble de solutions dans un vecteur d'entiers dénoté par r[n], où n est le nombre d'attributs dans la relation (Ruskey,1993). Les éléments du tableau doivent être des entiers entre 1 et n et ils doivent satisfaire la formule suivante :

$$r[i] \leq (\max(r[0], r[1], ..., r[i-1] + 1), \quad 0 < i < n, \quad r[0] = 1$$

Les auteurs ont proposé aussi une nouvelle approche nommé *Group oriented Restricted Growth String GA*(GRGS-GA) qui intègre le croisement et la mutation orienté groupes. Cette approche utilise le code en chaînes RG afin d'appliquer des nouveaux opérateurs de croissement et de mutation. Deux autres approches de fragmentation ont été présentées dans leur travail :

1. L'approche *RD-GA*(RanDom integer string GA) : elle structure chaque chromosome comme un vecteur d'entiers sans restriction.

2. . L'approche *RGS-GA* (Restricted Growth string GA) : est basée sur le codage en chaîne RG, elle nécessite que chaque chromosome devient une chaîne RG, ce qui permet d'éliminer les représentations redondants des partitions solutions.

La fonction objective utilisée dans cet algorithme est empirique, c'est une version modifiée de l'estimateur du coût qui a été proposé par Chakravarthy et al [48]. Leurs expérimentation a montré une amélioration significative de l'approche GRGS-GA dans les résultats et la vitesse de fragmentation.

Eltayeb Salih Abuelyaman [43] a proposé un algorithme de fragmentation verticale nommé *statPart* basé sur le schéma de la base de données et indépendant de fréquences des requêtes. L'algorithme utilise des statistiques comme paramètres d'entrés qui reflétant les nécessités de l'entreprise sur l'ensemble des requêtes à utilisation immédiats, les informations à propos du plan future de l'entreprise et l'ensemble de requêtes qui peuvent être ajoutées dans le future.

2.2.2.1 Synthèse

Les travaux que nous venons de présenter proposent des techniques de fragmentation qui interviennent dans le contexte des bases de données relationnelles distribuées. Ces travaux visent à améliorer le traitement des transactions et de répartir les fragments sur des sites géographiquement distants.

La plupart des approches de fragmentation verticale dans le contexte des bases de données distribuées utilisent des heuristiques. Les algorithmes proposés dans ce contexte adoptent soit des solutions données dans le contexte centralisé, par des algorithmes génétiques, ou par des algorithmes de clustering.

Il nous semble que l'estimateur du coût de Charkavarthy et al [48] est intéressant, du fait qu'il peut être appliqué dans les deux contextes centralisé et distribué et il permet l'ajout d'autres paramètres dans ses deux termes.

2.2.3 Fragmentation verticale (orientées-objet)

La conception des bases de données orientées-objet hérite des problèmes de la conception des bases de données relationnelles, et elle présente des difficultés supplémentaires liées au schéma représenté par les classes et par leur structure hiérarchique complexe [34]. Par rapport aux bases de données relationnelles, l'accès aux données dans les bases de données orientées-objet par l'utilisateur est défini par des méthodes qui compliquent encore le problème. La fragmentation verticale a pour but de fragmenter une classe de façon à ce que tous les attributs et les méthodes de la classe qui sont fréquemment utilisés ensemble sont groupés dans un même ensemble.

L'approche générale de la FV consiste dans une première étape de grouper pour chacune des classes les méthodes fréquemment utilisées ensemble [45]. Dans la deuxième étape, chaque groupe de méthodes est étendu afin d'incorporer tous les attributs accédés par les méthodes du groupe. Comme un attribut peut être référencé par des méthodes appartenant à des fragments différents, ces derniers peuvent être non disjoints.

Une mesure d'affinité a été utilisée dans l'algorithme de [45] afin de permettre de décider l'appartenance d'un attribut à un fragment.

Ezeife et K. Barker [45] ont établi une taxonomie en se basant sur le type d'attributs (simple ou complexes) et le type de méthodes (simple ou complexes). Les auteurs ont dégagé quatre modèles de classes qui peuvent

être définis dans un système d'objets distribués :

– modèle de classe constituée d'attributs simples et de méthodes simples
– modèle de classe constituée d'attributs complexes et de méthodes simples
– modèle de classe constituée d'attributs simples et de méthodes complexes
– modèle de classe constituée d'attributs complexes et de méthodes complexes

Grâce à cette classification, ils ont pu prendre en considération dans leur algorithme de fragmentation verticale toutes les caractéristiques de l'orienté objet. Des techniques de groupement étendues à l'objet, similaires à celles utilisées pour le groupement des attributs dans le modèle relationnel sont utilisées : Matrice d'usage des méthodes, fréquence des applications, matrice d'affinité des méthodes, . . . etc.

Une autre approche de fragmentation verticale qui a été proposée dans [46] qui consiste à prendre en compte de l'information sémantique véhiculée par les liens existant entre les classes. Le graphe de dépendance entre les classes est partitionné en un ensemble d'arbres de partition appelé forêt de partition et chaque classe appartenant au schéma d'origine apparait dans un seul arbre de partition. Un arbre de partition est construit en choisissant itérativement un nœud racine et en enlevant du graphe le sous graphe qui lui est connecté. A chaque étape de la construction, le nœud ayant le poids le plus fort est sélectionné comme racine, le poids d'un nœud étant égal à la somme des poids associés aux arcs incidents au nœud. Une fonction de coût exploitant des informations quantitatives telles que la fréquence d'accès aux méthodes a également été proposée par les auteurs. Cette fonction de coût a pour objectif d'optimiser le partitionnement. La décision de l'utiliser ou non est laissée à la charge du concepteur de la répartition. Le point de départ du processus de fragmentation est la forêt de partition. La fragmentation de la racine de chaque arbre est répercutée sur les classes membres correspondantes.

J. Nachouki et H. Briand [47] ont proposé de grouper les méthodes des classes en utilisant des mesures d'affinités entre méthodes et ils ont appliqués l'algorithme de fragmentation verticale Bond Energy Algorithm [18]. A partir d'un groupement de méthodes, ils déduisent quels objets, attributs et prédicats qui sont utilisés, ce qui produit des frag- ments mixtes. Comme ils recherchent des fragments mixtes disjoints, le cas de fragments ayant en commun des attributs et des objets est traité.

Song et Gorla [40] ont utilisé les algorithmes génétiques pour déterminer les instances des variables qui doivent être stockés dans chaque classe/sous-classe dans une hiérarchie de sous-classes, de sorte que le coût total des opérations sur la base de données est réduit au minimum.

2.2.3.1 Synthèse

La complexité des modèles de bases de données orientée objet est due à la hiérarchie des sous classes et la composition hiérarchique des classes qui compliquent la définition et la représentation des fragments verticales des classes.

Plusieurs techniques de fragmentation verticale des données orientées-objet ont été proposées dans la littérature. Plusieurs travaux ont essayé d'adapter les techniques de fragmentation verticale déjà utiliser dans le contexte relationnel et les adoptés dans le contexte orientée-objet centralisé et distribué.

2.2.4 Fragmentation verticale (Entrepôts de données)

Les entrepôts de données sont connus par leur volumétrie et les requêtes complexes caractérisées par des jointures de sélections et d'agrégations. Pour optimiser ces opérations et faciliter la gestion des ces données, la fragmentation a été utilisée dans les entrepôts de donnée en vue d'optimiser les requêtes OLAP par la conception des clusters de base de données parallèle [56] et pour générer les data cubes ROLAP basées sur la fragmentation des données [88].

Wu et al. [51] ont recommandé l'utilisation de la fragmentation verticale dans les entrepôts de données pour améliorer l'évaluation des requêtes en évitant le balayage des grandes tables. Ils ont considéré l'utilisation de la fragmentation verticale dans le cas suivant : La table des faits peut être fragmentée verticalement en fonction de ses dimensions, c'est à dire toutes les clés étrangères de la table des faits y sont partitionnées dans des tables différentes. Évidemment, dans un environnement distribué, quand la table des faits et les table de dimension sont stockées dans des sites distribués, une requête de semi jointure parallèle peut être appliquée par l'envoi de la partition verticale des clés étrangères aux tables de dimension. Cependant la reconstruction des tables fragmentées verticalement nécessite l'opération de jointure, qui est très coûteuse. Les auteurs n'ont pas proposé un algorithme de fragmentation.

O'Neil et D. Quass [37] ont introduit le concept de fragmentation verticale dans la défi- nition des index de projection dans les entrepôts par des index de projection ressemblent à un fragment vertical d'une relation.

Chaudhuri et al [62] ont développé une technique nommée « index merging » pour réduire le coût de stockage et de maintenance des index avec l'utilisation du concept de fragmentation verticale. L'« index merging » prendre un jeu existant d'index (peut être optimisé pour des requêtes bien précises dans la charge de travail) et produire un nouvel ensemble d'index avec un coût de stockage et de maintenance inférieure, et il conserve pratiquement tous les avantages d'interrogation de la première série d'index.

Les auteurs ont présenté un algorithme pour le « index merging» et ils ont démontré des économies dans le stockage des index et de maintenance par des tests d'expérimentation sur Microsoft SQL Server.

A.Datta, K. Ramamritham, et H. Thomas[53] ont exploité la fragmentation verticale pour construire un nouvel index appelé « Curio » dans les entrepôts de données modélisés par un schéma en étoile. L'index « Curio » est basé sur la fragmentation verticale de la table des faits, il permet d'accélérer l'exécution des requêtes décisionnelles et l'accès aux données, et il élimine la duplication

des données en stockant l'index et non pas les colonnes indexées (matérialisation des fragments au lieu des attributs indexés). Cela permet de réduire l'espace nécessaire pour ces index.

Golfarelli et al [58] ont utilisé la fragmentation verticale pour partitionner des vues définies sur un entrepôt de données dans le but de minimiser le temps de réponse des requêtes. Cette fragmentation est basée sur une charge de requêtes et un modèle de coût. Selon les auteurs, la fragmentation verticale désigne deux opérations : d'une part, le partitionnement d'une vue en plusieurs fragments et, d'autre part, l'unification en une seule vue de deux ou plusieurs vues ayant une clé commune. L'unification respecte la règle de reconstruction d'une table fragmentée à partir de ses fragments verticaux [35] et vise à réduire la redondance des vues. Les auteurs ont supposé que leur approche peut être bénéfique pour la distribution de l'entrepôt sur une architecture parallèle et ils ont proposé de combiner leur algorithme de fragmentation avec un algorithme d'allocation des fragments sur les nœuds distants [35].

Munneke et al. [54] ont proposé un type de fragmentation appelé « Sever », qui est équivalent à une fragmentation verticale dans une base de données relationnelle. La fragmentation Sever élimine une ou plusieurs dimensions dans un cube pour produire un fragment. Afin d'assurer la reconstruction des fragments, une ou plusieurs dimensions sont dupliquées dans tous les fragments.

Golfarelli. M et al. [55] ont étudié le problème de la fragmentation verticale des vues relationnelles pour minimiser le temps de réponse global des requêtes. Chaque vue inclut habituellement plusieurs mesures qui, dans la charge de travail, sont rarement demandées ensemble. Les auteurs ont supposés que les performances du système peuvent être augmentées en divisant les vues à matérialisées dans des petites tables, chacune comprenant seulement les mesures qui sont typiquement apparaissent ensemble dans les requêtes. D'autre part, les requêtes *drill-across* sont formulées sur des cubes de données multiples et ils impliquent des mesures prises de deux vues ou plus. Les coûts d'accès pour ces requêtes peuvent être diminués en unifiant ces vues dans des tables plus grandes où toutes les mesures exigées sont stockées

ensemble. Les auteurs ont remarqué que dans le contexte d'entrepôt de données, la présence des vues superflues rend le problème de fragmentation plus complexe que dans les bases de données relationnelles traditionnelles puisqu'elle exige pour décider de quelles vues chaque requête devrait être exécuté.

Ils ont formulé le problème de fragmentation comme un problème de minimisation d'un programme linéaire en nombre entier 0-1 et ils ont défini une fonction de coût. Quant à sa résolution, les auteurs ont proposé un algorithme de type séparation et évaluation (branch-and-bound).

2.2.4.1 Synthèse

Les travaux qui traitent le problème de la fragmentation verticale dans les entrepôts de données sont peu nombreux. Certains utilisent une fragmentation verticale pour construire des index. En effet, un index de projection ressemble à un fragment vertical de relation.

La majorité des travaux réalisés sur la fragmentation verticale dans les entrepôts de données ont été appliqués dans le niveau physique de la conception des entrepôts (index, problème de sélection).

2.3 Synthèse générale

La fragmentation verticale a été étudiée depuis les années 70s. Il y a deux approches principales [96] : la première est basée sur l'affinité et la deuxième sur un facteur de coût.

L'approche de FV basée sur l'affinité est originaire des bases de données centralisées, dans laquelle le nombre d'accès disque est le facteur principal qui influe sur les performances du système. Cette approche a été adoptée dans les bases de données distribuées avec l'ajout d'un coût de transfert entre les sites comme étant un deuxième facteur qui influe sur les performances du système. L'affinité entre les attributs peut refléter seule- ment l'attachement des attributs utilisés par les requêtes. La FV basée sur l'affinité peut réduire le nombre d'accès disque, par contre, il n'y a pas une preuve claire dans les

travaux antérieur qui montrent vraiment la disponibilité des données et l'amélioration des performances du système.

L'approche basée sur un facteur de coût est dirigée par un modèle de coût pour évaluer et effectuer la fragmentation d'un schéma de base de données. La solution optimale choisis dans cette approche est le schéma de fragmentation qui génère le moins d'accès disque.

Les algorithmes de fragmentation verticale que nous avons étudiée reposent sur une méthode d'optimisation et une fonction objective. Toutefois, l'approche graphique développée par Navathe et Ra [22] est une exception, elle ne dispose pas d'une fonction coût explicite. Cependant, La plus part de ces derniers ont quelques limitations qui compliquent la tâche du concepteur des bases de données :

– Le concepteur de base des bases de données dispose d'insuffisances empiriques dans les fréquences des requêtes. Cette limitation rend la fragmentation inapplicable sur les nouveaux schémas de base des données.
– La fréquence des requêtes est une fonction qui dépend de plusieurs variables (le temps, l'utilisateur et les besoins futurs d'une organisation). Les changements dans les structures organisationnelles ou les besoins de l'entreprise peuvent faire appel à d'autres attributs.
– La plus part des algorithmes proposent de partitionner les attributs à la base des fréquences des requêtes. Cette limitation tiges de la nature dynamique des fréquences des requêtes.
– L'affinité des attributs est une mesure imaginaire d'un lien entre les pairs d'attributs, cette mesure ne peut pas refléter la proximité en affinité quand plus des deux attributs sont utilisés dans la partition formée.

Dans le contexte des entrepôts de données, le choix des tables (de dimensions ou de faits) à fragmenter dans les entrepôts joue un rôle très important dans la qualité de fragmentation. Les choix possibles sont les suivants :

– Fragmenter verticalement une ou les tables de dimension : ce choix pourrait être intéressant dans le cas où la taille des tables de dimension est importante. Cette fragmentation pourrait réduire le coût des opérations de jointures en étoile. Elle pourrait être intéressante aussi dans le cas d'une table de dimension avec un nombre d'attributs élevé.
– Fragmenter seulement la table de faits : celle-ci est composée des clés étrangères des tables de dimension et des mesures. La FV possible de la table de faits consiste à fragmenter l'ensemble des mesures en différents fragments. Les clés étrangères seront dupliquées dans chaque fragment vertical.

2.4 Conclusion

Nous avons présenté dans ce chapitre un état de l'art sur les principaux travaux de recherche sur le problème de fragmentation verticale dans les bases de données. Cette étude a été complétée par une analyse et une synthèse sur les différentes approches proposées dans ce domaine.

Nous avons remarqué que la plus part des algorithmes de fragmentation verticale [2, 10, 11, 20, 24] contiennent deux parties essentielles : une méthode d'optimisation et une fonction objectif. Plusieurs algorithmes de FV utilisent des heuristiques pour la création des fragments d'une relation [9, 22, 11, 45], d'autres utilisent des méthodes mathématiques pour trouver le schéma de fragmentation optimal [2, 12, 14, 55]. Nous avons remarqué aussi que les résultats obtenus par les algorithmes de fragmentation verticale sont souvent différentes voire pour la même matrice d'affinité se qui montre que les fonctions d'objectifs de ces algorithmes sont aussi différentes.

Dans ce qui suit, nous allons détailler quelques algorithmes de la fragmentation verticale.

CHAPITRE 3
Les algorithmes de fragmentation verticale

3.1 Introduction

Telle qu'elle est déjà définie, la fragmentation verticale d'une relation R répartie les attributs de la relation afin de générer plusieurs tables $\{R_1, R_2, \ldots, R_n\}$, contenant chacun un sous-ensemble des attributs de R en plus de la clé primaire de R. Elle permet d'accélérer l'exécution des requêtes décisionnelles, et élimine la duplication des données en stockant l'index et non les colonnes indexées, cela permet de réduire l'espace nécessaire pour ces index.

Nous allons voir dans ce chapitre quelques algorithmes de la FV, à savoir l'algorithme de BEA (Bond Energy Algorithm), algorithme de fragmentation verticale binaire (BVP), algorithme basé sur la théorie des graphes et les algorithmes génétiques.

3.2 Les entrées dans les algorithmes de fragmentation verticale

L'entrée de plusieurs algorithmes de fragmentation est la matrice d'usage. La matrice indique l'utilisation des attributs selon les transactions.

Soit $Q = \{q_1, q_2, \ldots, q_m\}$ l'ensemble de requêtes destinées à s'exécuter sur une relation

$R = \{a_1, a_2, \ldots, a_n\}$. A chaque requête q_i et chaque attribut a_j, on associe une valeur d'usage de l'attribut noté $Use(q_i, a_j)$ définie comme suit :

$$Use(q_i, a_j) = \begin{cases} 1 & \text{si } a_j \text{ est utilisé dans la requête } q_i \\ 0 & \text{sinon} \end{cases}$$

$$\begin{pmatrix} & a_1 & a_2 & . & . & a_n \\ q_1 & 1 & 0 & & & \\ q_2 & 0 & & & & \\ . & 1 & & & & . \\ . & . & & & 1 & 0 \\ q_m & . & . & . & 0 & 1 \end{pmatrix}$$

Cependant cette matrice n'est pas suffisante pour fragmenter une relation car les valeurs qu'elle contient ne décrivent pas les fréquences des différentes applications. Les auteurs [18] ont introduit alors la quantité $Aff(a_i, a_j)$, qui mesure l'affinité entre deux attributs a_i et a_j d'une relation R en fonction de l'ensemble Q. L'affinité entre l'attribut i et j est définit comme suit :

$$Aff(a_i, a_j) = \sum_{\substack{k: \\ Use(qk,ai)=1 \\ Use(qk,aj)=1}} \left(\sum_{sites\ l} ref_l(q_k) \times acc_l(q_k) \right)$$

Où

$ref_l(q_k)$ est le coût (nombre d'accès à qui référence les deux attributs a_i et a_j) de la requête q_k dans le site l. $accl(q_k)$ est la fréquence de la requête q_k dans le site l.

3.3 Quelques algorithmes de fragmentation verticale

Dans cette section, on va détailler quelques algorithmes de fragmentation qui ont été proposés dans le contexte relationnel.

3.3.1 Algorithme de BEA (Bond Energy Algorithm)

L'algorithme BEA [18] est utilisé pour regrouper les attributs d'une relation basée sur les valeurs d'affinités des attributs dans la matrice d'affinité. Il prend en entrée la matrice d'affinité, il permute ses lignes et ses colonnes et il génère une matrice d'affinité regroupée (Clustered Affinity Matrix- CAM). La permutation est effectuée de manière à maximiser la mesure d'affinité globale (Afinity mesure-AM) :

$$AM = \sum_{i=0}^{n} \sum_{j=0}^{n} Aff_{ij} [Aff_{i,j-1} + Aff_{i,j+1} + Aff_{i-1,j} + Aff_{i+1,j}]$$

Et puisque la matrice d'affinité des d'attributs est symétrique, alors, la fonction pourrait être réduite à :

$$AM = \sum_{i=0}^{n} \sum_{j=0}^{n} Aff_{ij} [Aff_{i,j-1} + Aff_{i,j+1}]$$

Le lien (bond) entre deux attributs i et j est définit comme suit :

$$Bond_{i,j} = \sum_{z=1}^{n} Aff_{ij} [Aff_{z,i} + Aff_{z,j}]$$

La contribution nette à la mesure d'affinité globale de placement de l'attribut k entre i et j est :

$$Cont_{ikj} = 2Bond_{i,k} + 2Bond_{k,j} + 2Bond_{i,j}$$

Attributs	1	2	3	4	5	6	7	8	9	10
1	75	25	25	0	75	0	50	25	25	0
2	25	110	75	0	25	0	60	110	75	0
3	25	75	115	15	25	15	25	75	115	15
4	0	0	15	40	0	40	0	0	0	40
5	75	25	25	0	75	0	50	25	25	0
6	0	0	15	40	0	40	0	0	0	40
7	50	60	25	0	50	0	85	60	60	0
8	25	110	75	0	25	0	60	110	75	0
9	25	75	115	15	25	15	25	75	115	15
10	0	0	15	40	0	40	0	0	15	40

Table 3.1: Matrice d'affinité

La génération de la matrice CAM se fait en trois étapes [29] :

- Initialisation :

Placer et fixer une des colonnes arbitrairement de la matrice d'affinité dans la matrice CAM.

- Itération :

Choisir chacune des $(n - i)$ colonnes de la matrice d'affinité (où i est le nombre de colonnes déjà mis dans CAM) et essayer de les placer dans les $i + 1$ positions dans CAM.

Choisir le placement qui donne la plus grande contribution à la mesure d'affinité globale.

Continuer jusqu'à ce qu'il n'y aura plus de colonnes à placer.

- Le tri des lignes :

Une fois le tri des colonnes est déterminé, le placement des lignes devrait également être modifié de manière à ce que leurs positions relatives correspondent à la position relative des colonnes.

Le résultat de l'application de l'algorithme sur la matrice d'affinité du tableau 3.1 est donné comme suit :

Attributs	5	1	7	2	8	3	9	10	4	6
5	75	75	50	25	25	25	25	0	0	0
1	75	75	50	25	25	25	25	0	0	0
7	50	50	85	60	60	25	60	0	0	0
2	25	25	60	110	110	75	75	0	0	0
8	25	25	60	110	110	75	75	0	0	0
3	25	25	25	75	75	115	115	15	15	15
9	25	25	25	75	75	115	115	15	15	15
10	0	0	0	0	0	15	15	40	40	40
4	0	0	0	0	0	15	15	40	40	40
6	0	0	0	0	0	15	15	40	40	40

Table 3.2: La matrice d'affinité regroupée (CAM)

L'algorithme de BEA est jugé approprié pour les raisons suivantes :

– Il est spécialement conçu pour déterminer les groupes d'éléments similaires. Il rassemble les attributs qui ont les plus grandes valeurs d'affinité ensemble, et ceux qui ont des valeurs plus petites en même temps.

– Les derniers groupements sont insensibles à l'ordre dans lequel les éléments sont présentés à l'algorithme.
– La matrice d'affinité est symétrique, et elle permet ainsi la permutation des paires de lignes et de colonnes, ce qui réduit la complexité.
– A partir de la définition de l'affinité Aff_{ij} , la matrice initiale d'affinité est déjà semi block diagonale. Dans ce cas, chaque élément de la diagonale a une valeur plus grande de tout élément sur la même ligne ou la colonne.
– Le temps de calcul de l'algorithme est raisonnable : $O(n^2)$, où n est le nombre d'attributs.

L'inconvénient de cette approche de fragmentation est que l'algorithme détermine un ordre aux attributs, mais il ne permet pas aux concepteurs des bases de données de décider comment assembler les attributs pour former des fragments. Les similarités des paires d'attributs peuvent être insuffisantes si les similarités entre les grands groupes d'attributs ne sont pas prises en compte.

3.3.2 L'algorithme de fragmentation verticale binaire (BVP)

Navathe et al [20] ont étendu les résultats de Hoffer et Severance [11] par des algorithmes de groupements des attributs quantitativement en tenant compte de nombre des fragments avec des propriétés similaires. L'approche adoptée dans cet algorithme est la division plutôt que le groupement. L'idée de cette approche est que la solution optimale est beaucoup plus proche du groupe composé de tous les attributs (supposé être le point de départ) que les groupes qui sont des partitions à un seul attribut.

L'objectif de l'opération de partitionnement est de trouver les ensembles des attributs qui ont un accès unique. Cet algorithme utilise la matrice CAM pour partitionner un objet en deux fragments disjoins.

Supposons qu'un point X est fixé le long de la diagonale principale de la matrice CAM, comme montre la figure 3.1. Le point X définit deux blocs U (bloc supérieur) et L (bloc inférieur). Chaque bloc représente un fragment vertical donné par l'ensemble des attributs dans le bloc.

Soit A_t l'ensemble des attributs utilisés par la transaction t définie comme suit :

$$A_t = (I \mid q_{ti} > 0)$$

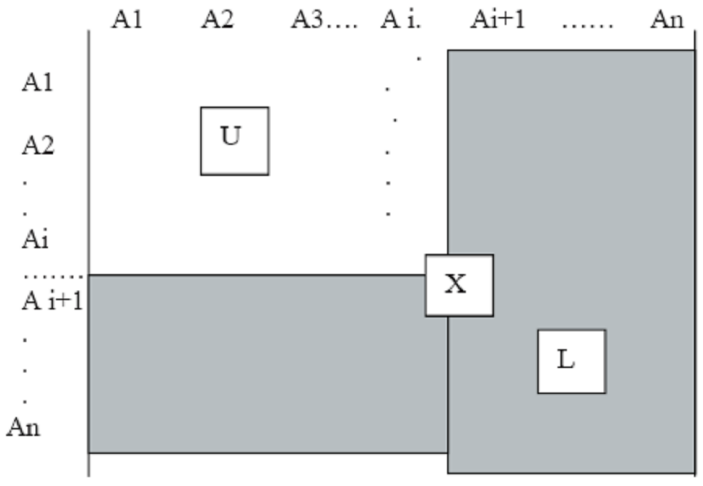

Figure 3.1: La matrice CAM

En utilisant A_t, on peut calculer les ensembles suivants:

T = (t | t c'est une transaction)
LT = (t | $A_t \in L$)
UT = (t | $A_t \in U$)
IT = T − (LT ∪ UT)

Attributes	5	1	7	2	8	3	9	10	4	6
5	75	75	50	25	25	25	25	0	0	0
1	75	75	50	25	25	25	25	0	0	0
7	50	50	85	60	60	25	60	0	0	0
2	25	25	60	110	110	75	75	0	0	0
8	25	25	60	110	110	75	75	0	0	0
3	25	25	25	75	75	115	115	15	15	15
9	25	25	25	75	75	115	115	15	15	15
10	0	0	0	0	0	15	15	40	40	40
4	0	0	0	0	0	15	15	40	40	40
6	0	0	0	0	0	15	15	40	40	40

Figure 3.2: La matrice CAM partitionnée

T représente l'ensemble de toutes les transactions. LT et UT représentent l'ensemble des transactions qui composent le partitionnement et qu'elles peuvent être entièrement exécutées en utilisant les attributs dans le bloc inférieur (L) ou le bloc supérieur (U) respectivement. IT représente l'ensemble des transactions nécessaire pour accéder à la fois aux deux fragments. On peut calculer :

$$CT = \sum t \in T\ qt$$

$$CL = \sum t \in LT\ qt$$

$$CU = \sum t \in UT\ qt$$

$$CI = \sum t \in IT\ qt$$

CT est le nombre total d'accès des transactions à l'objet considéré. CL et CU représentent le nombre total d'accès des transactions qui n'ont besoin que d'un fragment (L et U respectivement). CI est le nombre total d'accès des transactions qui ont besoin des deux fragments L et U. Au total, il y a (n − 1) positions possible pour le point X le long de la diagonale, où n est la taille de la matrice d'entrée (c'est à dire le nombre d'attributs).

Une partition disjointe est obtenue par la sélection du point X le long de la diagonale en maximisant la fonction objectif Z :

$$\text{Max } z = CL \ldots CU - CI$$

Le partitionnement qui correspond à la valeur maximale de la fonction z est accepté si z est positif, et il est rejeté sinon. La fonction objective précédente provient d'un jugement empirique de ce que devrait être considéré comme un *"bon"* partitionnement.

La fonction augmente dans CL et CU, et diminue dans CI. Pour une valeur donnée de CI, l'algorithme sélectionne CL et CU de telle sorte que le produit $CL \cdot CU$ est maximisé. Cela se traduit dans le choix des valeurs de CL et CU qui sont aussi proche que possible. Ainsi, la fonction z produit des fragments qui sont "équilibrés" avec la prise en compte de la charge des transactions.

Cet algorithme présente l'inconvénient de ne pas être en mesure de partitionner un objet en sélectionnant un bloc intérieur embarqué. Cet inconvénient peut être évité par l'utilisation de la procédure SHIFT , qui déplace la colonne la plus à gauche de la matrice d'affinité à l'extrême droite, et la ligne la plus haute de la matrice vers le bas. La procédure SHIFT est appelée au total n fois, de sorte que chaque bloc diagonal a la possibilité d'être apporter en haut à gauche dans la matrice. Lorsque la procédure SHIFT est utilisée, la complexité de l'algorithme augmente d'un facteur de n.

Les auteurs ont montré que l'utilisation de la procédure SHIFT améliore la solution du problème de partitionnement verticale binaire dans plusieurs cas. Il est possible de concevoir un partitionnement de n-façons mais le temps de

calcul est très grand. La solution consiste à appliquer récursivement l'algorithme de partitionnement binaire à chacun des fragments obtenus au cours de l'itération précédente.

L'algorithme BVP a quelques limitations :

– Il utilise la matrice d'affinité comme entrée, à cause des valeurs d'affinités qui sont des mesures des liens imaginaires entres les attributs, cette mesure ne reflète pas la proximité ou l'affinité dans le cas où plus de deux attributs sont impliqués.
– L'algorithme suppose qu'il y a toujours (n-1) possibilités pour partitionner une relation R, ignorons le fait qu'il se peut dans un cas où on considère la relation R entière comme un seul fragment qui peut être une solution optimale.

3.3.3 Algorithme de fragmentation basé sur la théorie des graphes

Une autre approche de fragmentation verticale basée sur des concepts de la théorie des graphes a été présentée par Navathe et ra [22]. Elle consiste à chercher la fragmentation verticale à partir de la matrice d'affinité qui est considérée comme un graphe complet non orienté, nommé *graphe d'affinité* où la valeur d'une arrête correspond à l'affinité entre deux sommets. Un sommet correspond à un attribut.

La figure suivante montre le graphe d'affinité correspondant à la matrice d'affinité 3.3. L'algorithme forme un arbre de parcours linéaire, c'est-à-dire un graphe sans cycles, connexe à (n −1) arrêtes où n est le nombre de sommets du graphe. L'algorithme génère simultanément tous les fragments, un cicle est considéré comme étant un fragment.

Définition et notations :

- A, B, C dénotes des nœuds.
- a, b, c correspond aux arrêtes.
- p(e) désigne la valeur d'affinité d'une arrête e.
- Un cycle primitif « primitive cycle »correspond à n'importe quel cycle dans le graphe

d'affinité.

- Un cycle d'affinité « vaffinity cycle »est un cycle primitif qui contient un nœud de cycle « cycle node ».
- L'arrête de l'achèvement du cycle dénote l'arrête à sélectionner qui va compléter le cycle.
- Un nœud de cycle « cycle node »c'est un nœud du cycle qui complète la dernière arrête sélectionnée.
- Une arrête complémentaire «completing edge », qui a été sélectionnée auparavant.
- Une ancienne arrête « former edge » dénote une arrête qui a été sélectionnée entre la dernière coupe et le nœud de cycle.
- Une arrête de cycle « cycle edge » est n'importe quelle arrête qui forme un cycle.
- Une extension du cycle« Extension du cycle» se réfère à un cycle étant prolongé par pivotement au niveau du nœud de cycle.

Attribute usage matrix											Type	Number of accesses per time period
Attributes / Transactions	1	2	3	4	5	6	7	8	9	10		
T1	1	0	0	0	1	0	1	0	0	0	R	Acc 1 = 25
T2	0	1	1	0	0	0	0	1	1	0	R	Acc 2 = 50
T3	0	0	0	1	0	1	0	0	0	1	R	Acc 3 = 25
T4	0	1	0	0	0	0	1	1	0	0	R	Acc 4 = 35
T5	1	1	1	0	1	0	1	1	1	0	U	Acc 5 = 25
T6	1	0	0	0	1	0	0	0	0	0	U	Acc 6 = 25
T7	0	0	1	0	0	0	0	0	1	0	U	Acc 7 = 25
T8	0	0	1	1	0	1	0	0	1	1	U	Acc 8 = 15

Attributes	1	2	3	4	5	6	7	8	9	10
1	75	25	25	0	75	0	50	25	25	0
2	25	110	75	0	25	0	60	110	75	0
3	25	75	115	15	25	15	25	75	115	15
4	0	0	15	40	0	40	0	0	15	40
5	75	25	25	0	75	0	50	25	25	0
6	0	0	15	40	0	40	0	0	15	40
7	50	60	25	0	50	0	85	60	25	0
8	25	110	75	0	25	0	60	110	75	0
9	25	75	115	15	25	15	25	75	115	15
10	0	0	15	40	0	40	0	0	15	40

Figure 3.3: Exemple de matrice d'usage et sa matrice d'affinité

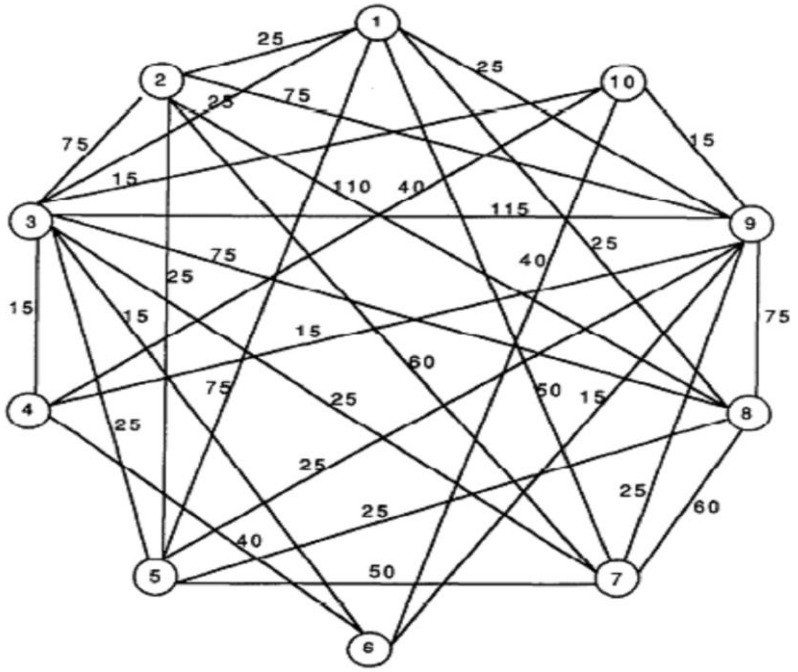

Figure 3.4: Le graphe d'affinité

L'algorithme de génération des fragments verticaux est résumé comme suit, chaque partition du graphe généré un fragment verticale.

Algorithme :

Données d'entré : la matrice d'affinité,

Résultats : un arbre,

Etape 1 : Construire le graphe d'affinité.

Etape 2 : Choisir un nœud de départ.

Etape 3 : Sélectionner un nœud qui satisfait les conditions suivantes :

- Il devrait être relié linéairement à l'arbre en cours de construction.

- Il devrait avoir la plus grande valeur parmi les choix possibles des arrêtes à chacune des feuilles de l'arbre (s'il y a plusieurs choix, alors n'importe quelle arrête peut être choisit). Cette itération est terminée lorsque tous les nœuds sont utilisés dans la construction de l'arbre.

Etape 4 : Si la prochaine arrête sélectionnée forme un cycle primitif :

(1) Si un nœud de cycle n'existe pas, vérifier s'il y a la "possibilité d'un cycle", et si la possibilité existe, marquer le cycle comme un cycle d'affinité. Considérer ce cycle comme une partition candidate. Aller à l'étape 3.

(2) Si le nœud de cycle existe déjà, éliminez l'arrête et passez à l'étape 3.

Etape 5 : Si la prochaine arrête sélectionnée ne forme pas un cycle primitif :

(1) S'il n'existe aucune arrête ancienne, vérifier si il y a la possibilité d'une extension de cycle par une nouvelle arrête, s'il n'y a pas de possibilité, supprimer l'arrête et considérer le cycle comme un fragment, aller à l'étape 3.

(2) Si l'ancienne arrête existe, changer le nœud de cycle.

Exemple :

L'exemple suivant est tiré de l'article [22], on utilise la même matrice d'affinité de la figure 3.3 et le graphe de la figure 3.4 après élimination des arcs de valeur 0 :

1. Choisit le nœud numéro 9 (étape 2) puis on applique l'algorithme,
2. Les arcs (9-3), (9-2) et (8-2) sont sélectionnés dans l'ordre (étape 3).
3. En ce moment, l'arc (8-9) ne peut pas former un cycle, donc l'arc (8-3) est sélectionné comme le prochain arc et il forme une partition de coordination (étape 4).
4. Ensuite, le processus est continu et l'arc 8-7 est sélectionné (étape 3). Avant, il y avait une partition de coordination, alors la possibilité de l'étendre est vérifié (étape 5.1).
5. Donc, le cycle est (9,3,2,8,9) parce que les valeurs des arrêtes (8-7) et (3-7) sont les deux inférieures à n'importe quelle nœud de cycle (étape 5.1).

L'algorithme génère trois cycles d'affinités avec l'arrête (3-4) et l'arrête (7-8). Il génère aussi les fragments (1,5,7) (2,3,8,9), (4,6,10) comme monte la figure suivante.

On remarque que les résultats sont similaires aux résultats de l'algorithme de [20]. Les auteurs ont montré que l'algorithme ne dépend pas du nœud de départ. Une des caractéristiques importantes de cet algorithme est que tous les fragments sont générés en une seule itération dans un temps égal à $O(n^2)$ qui est plus efficace que les approches précédentes (algorithme BEA, algorithme BVP).

On peut résumer les avantages majeurs de cette technique à travers les points suivants :

– Elle ne nécessite pas un partitionnement itératif binaire, les limites majeurs de la technique de partitionnement itératif binaire est que à chaque itération génère deux nouveaux problèmes se qui augmente la complexité de l'algorithme.

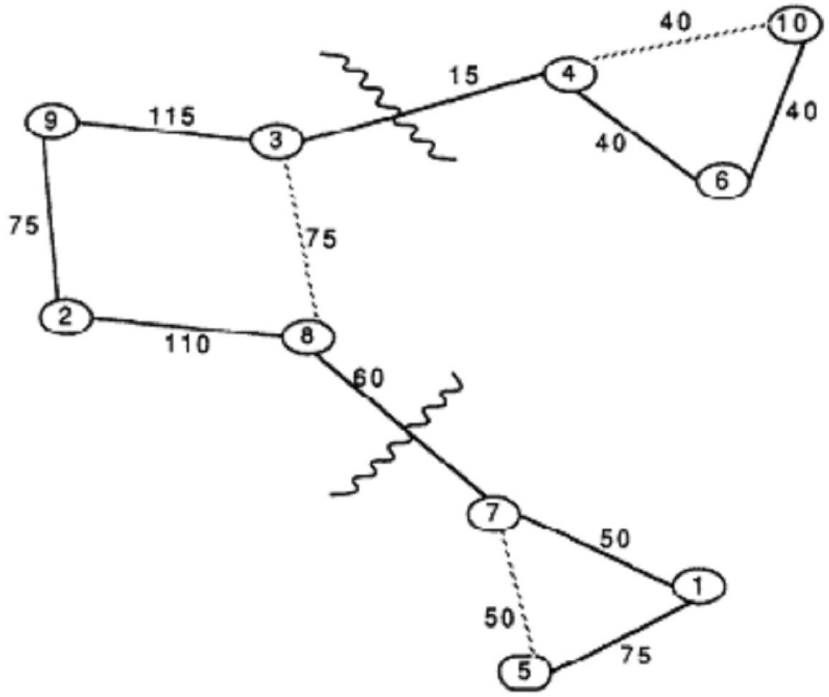

Figure 3.5: Schéma de fragmentation verticale

– La technique ne nécessite pas une fonction objective, donc la technique évite de conduire à des erreurs dans les étapes d'expérimentations.

– La méthode ne nécessité pas un algorithme complémentaire comme l'algorithme

SHIFT de [21].

– La complexité de l'algorithme est de $O(n^2)$, mieux que $O(n^2 \log n)$ de algorithme proposé dans [21].

3.3.4 Approche de fragmentation basée sur les algorithmes génétiques

La fragmentation verticale est un problème d'optimisation qui peut être résolu avec les algorithmes génétiques(AG), l'approche proposée dans [39] applique la contrainte de codage sous forme des chaînes RG (restricted growth-RG) [61] pour manipuler les chromosomes de sorte que les chromosomes redondantes seront exclus pendant le processus de l'AG.

Les auteurs de [39] ont proposé une nouvelle approche nommée Group oriented Restricted Growth String GA (GRGS-GA) qui intègre le croisement orienté groupe (group oriented crossover) et la mutation, ces deux techniques sont réalisables à l'aide des chaînes sous forme RG.

La fonction objective utilisée pour évaluer le schéma de fragmentation est empirique, c'est une version modifiée de l'évaluateur des partitions qui a été proposé par Chakravarthy et al [49]. Cet évaluateur utilise le critère de « square-error » couramment appliqué dans les stratégies de classification et qu'ils ont nommé SEPE(Square-Error Partition Evaluator).

Le Codage sous forme d'une chaine RG représente un groupe de solutions dans un tableau d'entiers dénoté par r[n], où n est le nombre d'attributs dans la relation [61]. Les éléments du tableau doivent être des entiers entre 1 et n.

Une chaine RG doit satisfaire la formule suivante :

$$r[i] = (\max(r[0], r[1], ..., r[i-1] + 1), 0 < i < n, r[0] = 1$$

Par exemple, $\{11231124\}$ est une chaine sous forme de RG, mais $\{44234421\}$ n'est pas, même s'ils mappent la même solution dans le schéma d'encodage des chaines.

- Le degré de la chaine RG r est la plus grande valeur dans r, notée d(r). Par exemple, considérons $r = \{11123221\}$, alors $d(r) = 3$.

- Le ième préfixe de la chaine RG, désigné par pRI, c'est la iéme sous chaine qui inclut les i premières valeurs de r.

- Le degré potentiel élevé hd(pRI) pour la position i dans la chaine de RG r est le degré le plus élevé possible pour le ième préfixe.

Par exemple, considérons $\mathrm{r} = \{1112321\}$, alors hd(pR5) = 3.

3.3.4.1 Les approches de fragmentation RD-GA et RGS-GA

L'approche RD-GA(RanDom integer string GA) est une approche simple et facile a implémenté qui structure chaque chromosome comme un vecteur d'entiers sans restriction.

L'approche RGS-GA est basée sur le codage sous forme des chaînes RG, elle nécessite que chaque chromosome devienne sous forme d'une chaine RG, ce qui permet d'éliminer les représentations redondantes des partitions solutions.

3.3.4.2 Opérations de création de nouveaux individus

Il existe plusieurs opérations pour la création de nouveaux individus :

La sélection : elle a été implémentée avec le principe de sélection par tournoi combiné avec la méthode d'élitisme.

Le croisement : Les deux approches RD-GA et RGS-GA utilisent la technique de croisement pour créer des nouveaux chromosomes.

La mutation : En plus de l'opération de mutation, une autre opération a été implémentée dans [39], c'est la mutation mono-point (où un seul gène est changé dans le chromosome dans chaque mutation).

Remarque : Dans le contexte de la fragmentation verticale, la mutation signifie qu'un attribut a été déplacé d'une partition à une autre.

3.3.4.3 Le rectifieur

Pour assurer que chaque chromosome est sous forme d'une chaîne RG, les auteurs ont introduit un rectificateur dont le but est d'ajuster les chromosomes sous forme RG et convertir les chromosomes obtenus après les applications de croisement et mutation.

La correction se fait à l'aide d'une fonction rectifiant qui scanne le

chromosome et la convertir en chaîne RG par l'ajustement des postions des gènes.

Par exemple, si on considère les deux chaines RG $r1 = \{11123213\}$ et $r2 = \{12134231\}$, et on suppose un point de croisement qui se produit après la position 4. Les deux chromosomes sont changés à $\{11124231\}$ et $\{12133213\}$, mais le premier chromosome n'est pas une chaine sous forme RG.

De même, une mutation dans la position 4 du chromosome r1 produira le chromosome $\{11133213\}$ qui n'est pas une chaine RG conforme.

Afin d'éviter de violer la contrainte de la chaine RG, les auteurs ont proposé des règles de sorte que la mutation de r1 dans la position 4 de l'exemple soit rejetée parce que le changement de l'élément à n'importe quel nombre autres que 2 causera une violation de la contrainte de la chaine RG.

Cependant, le croisement et la mutation restrictive limiteront la variété de descendance.

La solution proposée par les auteurs est qu'après chaque opération ils utilisent le rectifieur pour s'assurer que chaque chromosome est produit sous forme d'une chaine RG conforme. Le rectifieur met en application l'algorithme présenté dans [39] avec l'utilisation d'une table de hachage dans le processus pour stocker les valeurs des nombres entiers à employer dans la permutation des gènes.

3.3.4.4 L'approche GRGS-GA

L'approche nommée Group oriented Restricted Growth String GA (GRGS-GA) a apporté une amélioration par rapport à l'approche RGS-GA. Les auteurs ont conçus deux constructeurs différents pour le processus d'initialisation des chromosomes dans les AG basés sur les chaînes RG :

- le constructeur dispersé (Spare constructor - SC) est basé sur le constructeur utilisé dans la RD-GA. Après la génération d'une chaîne d'entiers, elle est converti en chaîne RG à l'aide de rectificateur.
- Le constructeur dense (Dense constructor - DC) génère les chromosomes, la contrainte RG est renforcée pour les premiers. Le processus de

rectification n'est pas nécessaire après la création aléatoire de tous les gènes d'un chromosome.

Ces deux constructeurs utilisent une fonction aléatoire pour la création d'un entier. Cependant, dans SC, chaque élément peut être rangé entre 1 à n, où n est fixé et il est égal au nombre maximum des fragments prévu par l'utilisateur. Dans DC, le premier élément est placé à 1 et la limite supérieure de chaque élément est incrémentée graduellement.

Pour la création de nouveaux individus, plusieurs opérations existent :

Le croisement : Le croisement orienté groupe de deux chromosomes passe par les étapes suivantes :

1. Unifier les id's des groupes (fragments) en ajustant l'id de l'un des parents pour que les deux parents utilisent un id de groupe unique.

2. Choisir au hasard un certain nombre d'id's de groupe pour chaque parent.

3. Injecter dans le premier parent les attributs des ids du groupe sélectionné dans le deuxième parent. Et enregistrer les groupes dans le premier parent qui a été modifiée (les groupes fragmentés).

4. Appliquer des stratégies spéciales visant à regrouper les attributs dans les groupes fragmentés pour obtenir la première descendance. Les attributs des groupes coupés doivent être attribués à nouveau suivants des stratégies, les auteurs ont proposé les deux stratégies suivantes dans le cadre de la fragmentation verticale :

L'union : elle permet d'inclure tous les attributs des groupes fragmentés dans un nouveau fragment.

Le fusionnement binaire : cette stratégie consiste à maintenir une liste des ids groupes fragmentés. Ensuite, deux ids sont sélectionnés de manière aléatoire de la liste, les attributs des deux ids sont fusionnés dans un seul groupe.

Mutation : Dans l'approche GRGS, les stratégies de mutation ont suivis trois directions : la création des nouveaux groupes, l'élimination d'un groupe existant et mélanger un petit nombre de gène entre les groupes.

Dans l'approche GRGS, les auteurs ont construit quatre opérateurs de mutation :

– fusionnement (merging)

– saut (jumping)

– fractionnement (splitting)

– l'échange (swapping).

L'expérimentation de l'approche proposée dans [39] a été réalisée dans trois cas. Dans le premier cas, les auteurs ont utilisé et comparer les trois approches RD, RGS et GRGS avec une matrice d'usage de 20 attributs. Le 2éme cas, ils considèrent une matrice d'usage générée aléatoirement de 100 attributs. Le 3éme cas, ont choisi une matrice d'usage de 300 attributs, ce dernier a été utilisé pour montrer la capacité et l'efficacité de l'approche
GRGS.

L'exemple de 20 attributs a été déjà traité par d'autres chercheurs, à savoir Navathe et al [20], Navathe et Ra [22] et Chakravarthy et autres [48]. Les méthodes proposées dans [20] et [48] ont données le même schéma de partitionnement optimal qui groupe 20 attributs dans quatre fragments.

Les nombre d'itération de tout l'ensemble des AGs pour cet exemple est 10. Chaque AG est exécuté 100 fois.

Le partitionnement mentionné ci-dessus de 4 fragments est évalué pour avoir une valeur de PE inférieur à 4.627 (si le partitionnement final dans chaque essai à une valeur de PE moins ou égale à 4.627, alors cet essai est considéré comme succès).

Le taux de succès dans l'approche RD, RGS et GRGS sont 65.0, 96.0 et 100% respectivement. Le nombre moyen de générations requises pour atteindre la solution optimale dans chaque AG est 56.6, 32.6, et 36.2 pour RD, RGS et GRGS respectivement.

On a remarqué que RD-GA est mauvais parmi les trois autres algorithmes en termes d'efficacité et vitesse de convergence.

Dans l'exemple de 100 attributs, le nombre d'itérations est donné à 50 pour

permettre à plus d'opérations de croisement et de mutation afin de créer une meilleure solution. Chaque AG est appliqué 10 fois sur la matrice d'utilisation de 100 attributs et les coûts de partitionnements finales réalisés par chaque essai sont présentés sur la figure suivante. Chaque essai de GRGS-GA trouve le partitionnement optimal connu quand la matrice d'usage est générée. Aucun essai de RD ou de RGS n'a atteint une telle solution optimale.

On remarque que quand le nombre d'attributs augmente, l'AG devient plus difficile à converger. L'avantage d'utiliser l'approche GRGS-GA est évident en face d'utilisation des deux autres approches.

Un autre exemple a été effectué par les auteurs, avec 400 attributs. Le même générateur de matrice d'usage est utilisé pour créer une matrice d'usage avec 400 attributs et 50 transactions. Le partitionnement optimal à l'intention d'avoir est 25 fragments. Ainsi, les 25 premières transactions sont consacrées aux attributs de chaque groupe, respectivement, et donner le nombre le plus élevés de fréquence d'accès. La probabilité de référence des transactions restantes à chacun des attributs est placée à 0.5. La valeur de PE du partitionnement optimal est 1.175.800.

Le nombre d'itérations est 100, les performances de l'approche RD et RGS se dégradent gravement. Ces deux approches convergent très lentement jusqu'à ce qu'elles s'arrêtent à environ 8000 générations. Par contre, l'approche GRGS-GA utilise le constructeur dispersé ou le constructeur dense trouve le partitionnement cible environ 5300 ou 3800 générations respectivement. Les résultats ont montré que le constructeur dense apporte un énorme impact sur la vitesse de convergence de GRGS-GA.

3.4 La fragmentation verticale dans les entrepôts de données

La fragmentation verticale représente un enjeu plus important dans les entrepôts que dans un contexte de base de données relationnelles ou objets. Cette importance est due au choix des tables (de dimensions ou des faits) à

fragmenter.

Les choix possibles sont les suivants :

– Fragmenter verticalement une ou les tables de dimension. Ce choix pourrait être intéressant pour le cas où la taille des tables de dimension est importante (le cas d'un schéma en étoile). Cette fragmentation pourrait réduire le coût des opérations de jointures en étoile.
– Partitionner seulement la table de faits. Celle-ci est composée des clés étrangères des tables de dimension et des mesures. La fragmentation verticale possible de la table de faits consiste à fragmenter l'ensemble des mesures en différents fragments. Les clés étrangères seront dupliquées dans chaque fragment vertical. Dans notre travail nous optons pour le premier choix.

Selon [84], le problème de fragmentation verticale dans le contexte d'entrepôt de don- nées peut être formalisé de la façon suivante :

Etant donné un ensemble de tables de dimension $D = \{D_1, D_2, \ldots, D_d\}$ et une table de faits F, un ensemble de requêtes OLAP fréquentes $Q = \{Q_1, Q_2, \ldots, Q_m\}$, où chaque requête $Q\,(1 \leq i \leq m)$ m possède une fréquence d'accès f_{Qi} , et un seuil M qui représente le nombre de fragments verticaux que l'administrateur souhaiterait avoir.

Le problème consiste à déterminer un ensemble de tables de dimension à fragmenter, et fragmenter une ou des tables de dimension de façon à ce que le coût total d'exécution des requêtes sur le schéma en étoile fragmenté soit réduit et que le nombre de fragments verticaux ne dépasse pas le seuil M.

Les entrepôts de données se prêtent bien à l'utilisation des techniques de partitionnement, car des requêtes complexes sur des grandes quantités de données peuvent être exécutées, D'après notre étude théorique sur la fragmentation verticale dans le domaine des entrepôts de données, on a remarqué que peu de travaux ont apporté une solution à la fragmentation verticale dans les entrepôts de données, bien que ces derniers contiennent de très grandes tables.

Wu et al. [1] dans leur article "research issues in data warehousing" ont recommandé l'utilisation de la fragmentation verticale dans les entrepôts de données pour améliorer l'évaluation des requêtes en évitant le balayage des grandes tables. Il est précisé que les algorithmes développés dans les bases de données réparties doivent être adaptés et réévaluer pour les entrepôts de données. Ils ont considéré le cas d'utilisation de la fragmentation verticale dans les entrepôts et ils ont proposé de fragmenter la table des faits verticalement.

On peut considérer d'autres cas d'utilisations de la fragmentation verticale dans les entrepôts de données, par exemple :

- Fragmenter verticalement des vues matérialisés
- Fragmenter verticalement des index
- Fragmenter totalement ou partiellement les tables de dimension en utilisant la fragmentation verticale et utiliser leurs schémas de fragmentation pour partitionner la table des faits.

3.5 Conclusion

Nous avons présenté dans ce chapitre une étude comparative de quelques approches qui traitent le problème de la FV. Nous avons comparé les algorithmes : BEA, de la fragmentation verticale binaire, l'algorithme basé sur la théorie des graphes et la génétique. Les travaux menés dans le contexte de La fragmentation verticale des données peuvent être divisés en deux familles [20] :

- Dans la première famille, une solution optimale du problème est tentée, généralement sous des hypothèses restrictives. Cette approche utilise généralement des méthodes mathématique afin d'obtenir le schéma de fragmentation verticale optimale.
- Dans la seconde, une approche heuristique est utilisée lorsque la taille de la fragmentation est grande. Cette famille des travaux utilise des algorithmes basés sur des méthodes approchées afin de trouver un meilleur schéma de fragmentation.

Notre travail sera appliqué sur un entrepôt de données relationnelles, l'objectif visé dans la deuxième partie de notre étude consiste à fournir un schéma de fragmentation optimale qui permet d'optimiser les performances des requêtes sur un entrepôt de données relationnelles. Cette technique d'optimisation repose sur la méthode de fragmentation verticale.

Nous allons présenter dans le chapitre suivant une approche basée sur les algorithmes génétiques et qui fournit une solution approchée à la FV dans les entrepôts des données.

CHAPITRE 4
Schéma de fragmentation verticale et calcul du coût

4.1 Introduction

Dans ce chapitre, nous allons présenter une formulation du problème de FV et une approche de sélection d'un schéma de FV. Nous donnons le modèle de coût que nous avons proposé, et nous décrivons l'algorithme proposé qui est basé sur les AG's pour la détermination d'un schéma de FV.

4.2 Formalisation du problème de fragmentation verticale

Nous formalisons le problème de sélection d'un schéma de fragmentation verticale comme étant un problème d'optimisation sous contraintes : étant donné un ensemble de requêtes $Q = \{q_1, q_2, \ldots, q_k\}$ exécutées sur un schéma d'un ED composé d'une table de dimension T fragmentée en m fragments $P = \{T_1, T_2, .., T_m\}$, un ensemble $D = \{D_1, D_2, .., D_n\}$ de n tables de dimension et une table de faits F, une contrainte de maintenance où M représente le nombre de fragments maximal de la table T fixé par l'administrateur. Notre objectif est de minimiser la fonction suivante :

$$\sum_{q \in Q} fq \times \big(cost(q, \{F, D, P\}) - cost(q, \{F, D, T\})\big) \qquad (4.2.1)$$

Où f_q représente la fréquence d'accès de la requête q. En respectant la contrainte de maintenance $m \leq M$. Ce problème d'optimisation consiste à partitionner une table T telle que le nombre de partitions de cette table soit borné par une constante M et le nombre d'opérations d'entrées/sorties (E/S) nécessaires pour l'exécution de toutes les requêtes soit minimal après la fragmentation. Il est définit comme suit :

Problème : Fragmentation Verticale d'une seule table

Instance :

- un ensemble de n tables de dimension $D = \{D_1, D_2, .., D_n\}$.
- une table de dimension $T = \{T_1, T_2, .., T_r\}$ de r attributs à fragmenter.
- une table de faits F.
- un entier positif M, où M est le nombre maximum de partitions qui peuvent être créées.

- un ensemble de requêtes $Q = \{q_1, q_2, .., q_k\}$ et pour chaque requête *qj, Cqj* est le nombre d'opérations d'E/S nécessaires pour son exécution sur le schéma non fragmenté et Lqj est le nombre d'opérations d'E/S nécessaires pour son exécution sur le schéma fragmenté.

Question : Est-ce que la table T peut être partitionnée au maximum en m fragments $\{T_1, T_2, .., T_m\}$ disjoints (sauf la clé primaire) avec $m \leq M$, tels que le nombre total des opérations d'E/S de toutes les requêtes sur un schéma fragmenté soit inférieur au total des coûts des opérations d'E/S sur le schéma non fragmenté :

$$\sum_{j=1}^{k} \mathrm{Lqj} \leq \sum_{\mathbf{j=1}}^{\mathbf{k}} \mathrm{Cqj}$$

Selon une partition donnée, le nombre des opérations d'E/S diminue puisque la taille en des tuples de chaque partition est inférieure à la taille des tuples de la table fragmentée. Par conséquent, le nombre de jointures peut augmenter s'il y a plusieurs attributs dans différents fragments utilisés dans la même requête.

4.3 Algorithme de sélection d'un SFV

En raison de la complexité du problème de FV, le développement des heuristiques pour la sélection d'une solution satisfaisante est recommandé. Pour cela, nous présentons un algorithme génétique pour la sélection d'un schéma de fragmentation presque optimal. Avant de détailler cet algorithme, nous présentons un codage pour représenter un schéma de fragmentation sur

lequel des opérations seront définies. Nous donnons par la suite le modèle de coût que nous avons utilisé pour évaluer le coût d'exécution des requêtes sur un schéma fragmenté ou non.

4.3.1 Codage d'un schéma de fragmentation

Le codage que nous avons proposé est inspiré de celui des chaines RG appliqué dans les algorithmes génétiques et le clustering [39],[61]. Nous représentons un ensemble de solutions (schéma de fragmentation) dans un tableau d'entiers dénoté par r[s] où s est le nombre d'attributs de la table sans compté la clé primaire. Une chaine RG doit satisfaire la formule suivante [39] :

$$r[i] < max(r[1], r[2], .. r[i-1] + 1) \qquad (4.3.1)$$

avec $r[1] = 1,\ 1 \leq i \leq s$.

Par exemple $\{11231124\}$ est une chaîne RG, mais $\{44234421\}$ n'est pas, même s'ils mappent la même solution dans le schéma d'encodage des chaines.

Dans notre cas, le vecteur r correspond à un schéma de fragmentation et r[i] désigne le numéro de fragment de l'attribut i. Pour garantir la contrainte de maintenance, les éléments du tableau doivent être entre 1 et M (nombre de fragments maximal).

Exemple 1 :

Soit $r1 = \{1221341324\}$ et $r2 = \{1122341343\}$ deux codages qui représentent les deux schémas de fragmentation $\{\{1,4,7\},\{2,3,9\},\{5,8\},\{6,10\}\}$ et $\{\{1,2,7\},\{3,4\},\{5,8,10\},\{6,9\}\}$.

Le degré d(r) d'une chaine RG r, est la plus grande valeur dans r.

Considérons $r = \{11123221\}$, alors $d(r) = 3$.

Le degré d'un individu correspond au nombre de fragments dans le schéma de fragmentation.

Exemple 2 :

Le codage du schéma de fragmentation de la table $CLient(\text{CID}, \text{Nom}, \text{Sexe}, \text{Age}, \text{Ville})$ est $r = \{1222\}$ qui représente un schéma de fragmentation composé des deux fragments $F1 = \{\text{CID}, \text{Nom}\}$ et $F2 = \{\text{CID}, \text{Sexe}, \text{Age}, \text{Ville}\}$. La clé primaire n'a pas été présentée dans le codage car elle est dupliquée sur tous les fragments

Pour évaluer la qualité d'un schéma de fragmentation, nous avons défini un modèle de coût qui estime le nombre d'opérations d'E/S nécessaires pour exécuter un ensemble de requêtes que nous allons le présenté par la suite.

4.3.2 Modèle de coût

Le modèle de coût que nous allons présenter permet d'estimer le nombre d'E/S nécessaires (exprimé en nombre de pages chargées) pour exécuter une requête définie sur un schéma de base de données. Dans ce modèle, nous ne prenons pas en compte le temps CPU qui est supposé négligeable par rapport au temps nécessaire pour effectuer les E/S. Etant donné un schéma défini par une table à fragmenter T et n tables, chaque requête q défini sur ce schéma comporte plusieurs prédicats de sélection et des jointures. Le modèle de coût utilise des données collectées sur le schéma de la base de données (taille des tables, taille d'un attribut, nombre de pages stockant une table, ...,etc.), le système physique (taille du buffer, taille de la page système, ...,etc.) et la fréquence d'accès des requêtes.

Le modèle prend en considération seulement les requêtes où la table à fragmenter T est impliquée. Nous donnons par la suite les principaux paramètres utilisés dans le modèle de coût.

$|T|$ nombre de bloques nécessaires pour stocker la table T

$||T||$ nombre de tuples de la table T

β taille du buffer en nombre de pages
ω_a taille moyenne en octet de l'attribut a
δa nombre de valeurs distincte de l'attribut a
fq fréquence d'accès de la requête q

Les requêtes prise en compte dans l'élaboration du modèle de coût sont des requêtes de jointure ayant la structure suivante :

SELECT [AS], FC(AA) FROM T1,T2,…,Tk
WHERE PJOIN AND
PSEL **GROUP BY** [ATTG]

Où

AS : l'ensemble des attributs retournés dans la réponse à la requête (il peut être vide pour les requêtes retournant un calcul).

FC : les fonctions de calcul (MIN, MAX, COUNT, SUM, AVG). AA : l'ensemble des attributs d'agrégation.

k : le nombre de tables utilisées par la requête.

PJOIN : un ensemble de prédicats de jointure entre les tables de dimension et/ou la table de fait.

PSEL : un ensemble de prédicats de sélection sous forme de conjonction. Chaque conjonction est constituée de plusieurs prédicats définis sur la même table.

ATTG : l'ensemble des attributs de groupement. Ces attributs peuvent être des attributs de faits et/ou de dimension.

4.4 Processus d'évaluation d'une requête

L'évaluation d'un schéma de fragmentation consiste à estimer le nombre d'E/S nécessaire pour l'exécution de chaque requête dans la charge de travail. Afin d'évaluer chaque requête $q_{i,}$ la requête passera par un processus de réécriture, cette phase consiste à réécrire la requête sur le schéma de l'entrepôt de données fragmenté.

Le processus de réécriture génère une requête q', qui sera l'entrée de processus d'évaluation. Dans notre modèle de coût, nous avons utilisé une heuristique pour les jointures qui consiste à réduire la taille des tables à joindre

autant que possible.

Cette approche consiste à appliquer les opérations de sélection et de projection au début. C'est-à-dire «Pousser» les sélections et les projections avant la jointure.

Pour évaluer le coût de la requête q', elle passera par un processus de parsing et la convertir à deux ensemble de sous requêtes :

– les requêtes de sélection/projection
– les requêtes de jointure

Soit un schéma R composé de deux tables :

Employees (sin INT, ename VARCHAR(20), rating INT, age REAL) Maintenances (sin INT, planeId INT, day DATE, descCode CHAR(10))

Soit la requête q :

```
SELECT ename FROM Employees, Maintenances
WHERE planeId = 100 AND rating > 5 AND Maintenances.sin =
Employees.sin ;
```

Le résultat de processus de parsing donnera les deux ensembles suivants

[SELECT ename , sin FROM Employees where rating > 5 , SELECT sin FROM Maintenances WHERE planeId = 100] et [JOIN Maintenances , Employees (Maintenances.sin,Employees.sin)].

4.4.1 Calcul du coût d'une requête

Le calcul de coût d'une requête est basé sur des estimations des coûts des opérations de sélection et projection [89]. Par exemple, une projection produit un tuple résultat pour chaque tuple argument, le seul changement est dans la taille des tuples en sortie.

$$Si\ S = \pi_a(R)\ alors\ \|S\| = \|R\|\ et\ |S| = \frac{\omega_a \times \|S\|}{\beta} \qquad (4.4.1)$$

Quand on effectue une sélection, en générale on réduit le nombre de tuples, bien que la taille des tuples reste la même. Dans le cas le plus simple, quand un attribut est égal à une constante, il y a une manière pour estimer la taille des résultats. Nous avons comme statistique le nombre de valeurs distincts de chaque attribut.

Soit S = $\sigma_{a=const}$(R), où a est un attribut de R et const est une constante. Nous avons utilisé la formule suivante pour estimer la taille de S :

$$\|S\| = \frac{\|R\|}{\delta_a} \qquad (4.4.2)$$

Pour les opérations ($<, \leq, >, \geq$), nous supposons que ces derniers retournent un tiers des tuples. D'où :

$$Si\ \ S = \sigma_{a\ oper\ const}(R)\ alors\ \|S\| = \frac{\|R\|}{3} \qquad (4.4.3)$$

Quand la condition de sélection C est une conjonction de condition C = (C_1 AND C_2 AND . . . AND C_n), nous traitons la sélection σ_C (R) comme une cascade de sélections simple. Notez que l'ordre dans lequel nous plaçons ces sélections n'a pas d'importance.

L'estimation de la taille de résultat est la taille du résultat initial multiplié par le facteur de sélectivité de chaque condition Ci. Le facteur est $\frac{1}{3}$ pour les opérations ($<, \leq, >, \geq$), 1 pour $\neq$; et $\frac{1}{\delta_a}$ pour tout attribut a dans R comparer à une constante.

Exemple 3

Soit $R(a, b. c)$ une relation, et $S = \sigma_{a=10\ AND\ b<20}(R)$. Aussi. Soit $\|R\| = 10\,000$, et $\delta_a = 50$. Alors, notre meilleure estimation de $\|S\|$ est $\frac{\|R\|}{3\times 50}$, ou 67. C'est-à-dire 1150 des tuples de R subsistent le filtre $a = 10$ et $\frac{1}{3}$ de ces derniers subsistent le filtre $b < 20$.

Dans notre modèle de coût, nous avons considéré que la jointure est implémentée par un algorithme de hachage. Si T = R Join S, l'opération de jointure par hachage nécessite $3 \times (|R| + |S|)$ opérations disque d'E/S. Pour déterminer l'ordre des jointures, nous utilisons la méthode de sélectivité minimum [87]. Cette technique repose sur l'hypothèse que les meilleures solutions sont généralement celles qui engendrent des résultats intermédiaires de petite taille.

4.4.1.1Calcul de coût sur un schéma non fragmenté

Pour exécuter une requête q, nous avons retenu le plan d'exécution suivant :

Soit une requête q respectant la syntaxe que nous avons définie dans la sous section 3.2 et comporte k jointures. Cette requête est divisée en sous requêtes q_{e1}, q_{e2}, ..., q_{ep} qui définissent les opérations élémentaire de sélection/projection et des sous requêtes de jointure q_{j1}, q_{j2}, ..., q_{jp}. Le plan d'exécution comporte les étapes suivantes :

1. Déterminer les sous requêtes de sélection/projection. Nous notons R_i le résultat de chaque sous requête.
2. Effectuer une sous requête de jointure par ordre de sélectivité minimum sur les résultats intermédiaires Ri. Nous supposons que l'ordre de jointure a été établi (en utilisant la méthode de sélectivité minimum).
3. Répéter l'étape 2 pour toutes les jointures présentées dans q. Le coût de la requête q sur un schéma $S = \{\mathrm{F}, \mathrm{T}, \mathrm{D1}, \mathrm{D2}, .., \mathrm{Dn}\}$ est la somme des coûts des sous requêtes :

$$Cost(q, S) = \sum_{x=1}^{p} C_e(q_{e_x}, S) + \sum_{y=1}^{p'} C_j\left(q_{j_y}, S\right) \qquad (4.4.4)$$

Où C_e représente le coût des sous requêtes de sélection/projection et C_j calcul le coût des sous requêtes de jointures. Nous supposons que tous les résultats intermédiaires des jointures tiennent en mémoire. Dans ce cas, l'optimiseur effectue la première jointure ensuite charge les résultats R_i une par une pour effectuer une jointure par hachage de ces dernières avec les résultats intermédiaires.

Le coût est calculé par la formule suivante :

$$Cost(q,S) = \sum_{x=1}^{p} C_e\left(q_{e_x}, S\right) + \sum_{y=1}^{p'} 3 \times |R_y| \qquad (4.4.5)$$

4.4.1.2 Calcul de coût sur un schéma fragmenté

Notre modèle de coût peut être utilisé pour estimer le coût d'une requête sur un schéma généré par le processus de fragmentation. Le nouveau schéma est composé de l'ensemble des tables de dimension $\{\mathrm{D1}, \mathrm{D2}, .., \mathrm{Dn}\}$ et l'ensemble de fragments $P = \{\mathrm{T1}, \mathrm{T2}, .., \mathrm{Tm}\}$ et une table de faits F.

Afin d'évaluer le coût d'une requête q, un processus de réécriture de la requête q est lancé pour générer une nouvelle requête q' par rapport au nouveau schéma.

Le coût de la requête q correspond au coût de la requête q'.

Exemple 4 :

Soit un schéma en étoile constitué de trois tables de dimension Client, Produit et Temps et une table de faits Ventes. Supposons que la table de dimension Client est fragmentée en deux fragments :

Client1 : $\pi_{\mathrm{CID,Nom,Prenom,Tel}}$(Client)

Client2 : $\pi_{\mathrm{CID,Ville}}$(Client)

Et Soit la requête q suivante qui calcule la somme des ventes effectuées par des clients algérois ayant acheté un produit le mois de juin :

SELECT Sum(montant) FROM Ventes V, Client C, Temps T WHERE V.CID=C.CID AND V.TID=T.TID AND C.Ville='ALGER' AND T.MOIS='JUIN'

Pour exécuter la requête q sur le nouveau schéma fragmenté, elle doit être réécrite. La nouvelle requête q' est donnée comme suite :

SELECT Sum(montant) FROM Ventes V, Client2 C, Temps T WHERE V.CID=C.CID AND V.TID=T.TID AND C.Ville='ALGER' AND T.MOIS='JUIN'

Nous présentons par la suite un algorithme de FV d'une table basé sur les algorithmes génétiques (AG's).

4.5 Algorithme génétique

La sélection, le croisement et la mutation sont considérés comme étant les opérations clés dans le développement d'AG. Nous avons utilisé la sélection par tournoi combiné avec l'élitisme. Nous sélectionnons au hasard n individus dans la population de base et nous choisissons les plus forts en coût pour la prochaine génération, ce qui élimine quelques individus plus faibles à chaque génération.

Pour la création de nouveaux individus, nous avons utilisé l'opérateur de croisement avec la stratégie suivante : Pour chaque locus, le fils n1 héritera le gène du parent dont la valeur est supérieure et le fils n2 héritera le gène du parent dont la valeur est inférieure entre les deux parents. Si la valeur des deux gènes des parents est identique, alors les deux fils prendront cette valeur.

Le croisement des deux individus $r1 = \{1221341324\}$ et $r2 = \{1122341343\}$ qui représentent les deux schémas : $\{\{1,4,7\},\{2,3,9\},\{5,8\},\{6,10\}\}$ et $\{\{1,2,7\},\{3,4\},\{5,8,10\},\{6,9\}\}$ respectivement donne les deux fils suivants :

$r1' = \{1222341344\}$ qui représente le schéma $\{\{1,7\},\{2,3,4\},\{5,8\},\{6,8,10\}\}$ et $r2' = \{1121331323\}$ qui définit le schéma$\{\{1,2,4,7\},\{3,9\},\{5,8\},\{5,6,8,10\}\}$.

Il existe plusieurs méthodes de mutation [86]. Dans notre travail nous avons utilisés trois stratégies de mutation :

La fusion : Deux numéros de fragments sont d'abord sélectionnés au hasard dans un individu, et les attributs dans les deux fragments sont fusionnés en un seul fragment.

La division : Cet opérateur choisit un numéro de fragments au hasard, puis, divise le groupe en deux petits groupes. La division est utile pour créer des nouveaux fragments dans le schéma.

Le déplacement : Cette stratégie de mutations met en œuvre la mutation d'un point, c'est à dire, un seul gène dans l'individu est modifié à chaque mutation. Dans le contexte de la FV. Le déplacement signifie qu'un attribut a changé de partition vers une autre.

Remarque :

L'application de ces trois opérations ne suffit pas lorsque les chaînes RG sont impliquées dans le processus. La restriction de chaîne RG nécessite de tester chaque individu et garantir qu'il s'agit d'une chaîne de RG. Pour cela, nous avons utilisé le rectifieur décrit dans le travail [39]. Le rectifieur permet d'ajuster un individu en chaîne RG. Aussi, il est important de convertir les nouveaux individus obtenus après l'application des opérations de mutation et le croisement en chaine RG.

Chaque individu (schéma de fragmentation) généré par notre AG est évalué en utilisant une fonction de fitness donnée par le modèle de coût décrit dans la section précédente.

La figure 4.1 représente l'architecture générale de fonctionnement de l'approche pro- posé pour la sélection d'un schéma de fragmentation verticale d'une table de dimension.

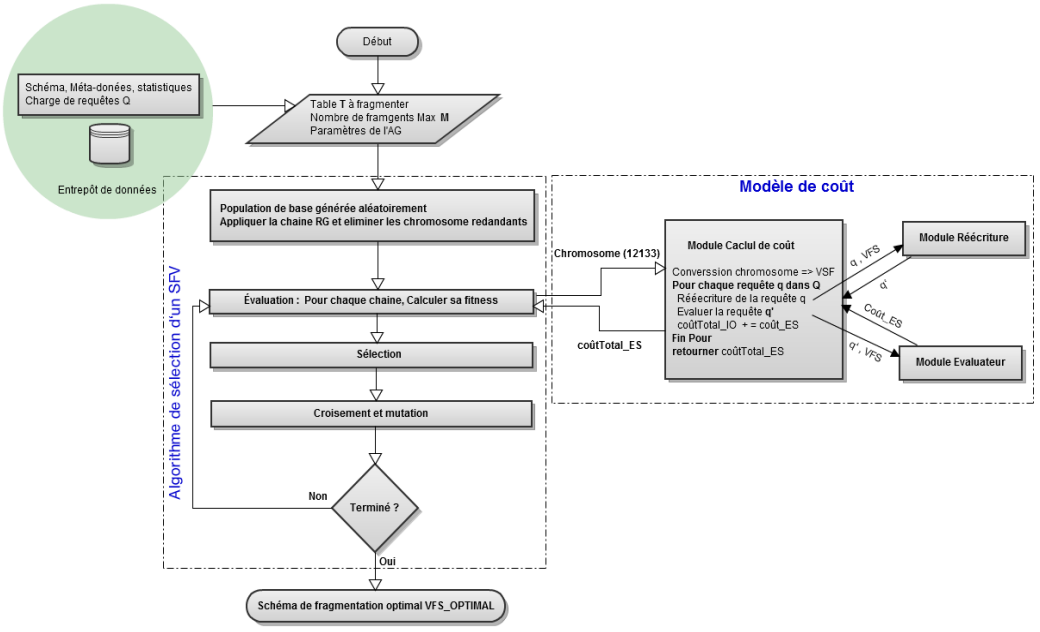

Figure 4.1: Approche de sélection d'un SFV basée sur un AG et un modèle de coût

L'AG se déroule comme suit :

Génération d'une population initiale pour l'AG

Suppression des chromosomes qui mappent le même schéma de fragmentation (contrainte RG)

POUR chaque itération FAIRE a) POUR chaque chromosome FAIRE

- Si le chromosome a déjà été évalué (nous utilisons une table de hachage pour stocker les chromosomes en chaine RG déjà évalués, si le chromosome existe dans cette table, sa implique qui l'a été déjà évalué, c'est une technique d'optimisation de temps de calcul).

- SI OUI retourner sa fitness,

- SINON :

POUR chaque requête q dans la charge de travail FAIRE

- Réécriture de la requête par rapport au nouveau schéma (le schéma et les fragments).

- Évaluation de la nouvelle requête q' (estimation en nombre d'E/S sur le nouveau schéma de l'entrepôt, c'est à dire :on prend en considération le schéma de fragmentation)

FIN FAIRE

Le coût (fitness) d'un chromosome est le total des coûts des requêtes.

FIN SINON

- Effectuer les FIN POUR

L'AG se termine et il renvoi le chromosome ayant un coût minimum

Conversion de chromosome en un schéma de fragmentation et génération du script SQL qui crée les tables (fragments).

Générer la liste de requêtes réécrit par rapport au nouveau schéma.

4.6 Conclusion

Nous avons parlé dans ce chapitre une approche de fragmentation verticale basée sur les algorithmes génétique. Nous avons essayé d'appliquer cette approche dans le contexte des entrepôts de données. Nous avons présenté notre formalisation du problème de fragmentation verticale, le codage qu'on a choisit pour modéliser une solution (Schéma de FV), ainsi qu'un modèle de coût pour évaluer la fitness d'un chromosome.

CHAPITRE 5
Implémentation et tests

5.1 Introduction

Nous allons présenter dans ce chapitre une implémentation de l'algorithme que nous avons proposé précédemment. Nous allons effectuer des tests sur des schémas des entrepôts de donnée générées à l'aide de l'outil DWEB [95].

Les résultats ont montré qu'il y a un gain en nombre d'entrés/sorties dans la majorité des requêtes après la fragmentation verticale de la table.

5.2 Implémentation

L'algorithme de sélection d'un schéma de fragmentation verticale (SFV) a été implémenté par le langage Java. Nous avons utilisé le package JGAP [93] qui est un Framework pour le développement des AGs et la programmation génétique en java, et le parseur sql General SQL Parser Java version (gsp) [94] qui fournit une analyse en profondeur de script SQL de diverses bases de données comme Oracle, SQL Server, DB2 et MySQL.

Le déroulement de programme est comme suit :

DEBUT

1. Génération d'une population initiale
2. Suppression des chromosomes qui mappent le même schéma de fragmentation (Contrainte RG)
3. Pour chaque itération :

– a) Application des opérations AG (mutation, croisement)

– b) Pour chaque chromosome (schéma de fragmentation de la table à fragmenter) :

Si le chromosome a été déjà évaluer (nous utilisons une table de hachage

pour stocker les chromosomes en chaine RG déjà évalués, si le chromosome existe dans cette table, ceci implique qu'il a été déjà évalué, c'est une technique d'optimisation de temps de calcul).

SI OUI retourner sa fitness,

SINON : Pour chaque requête dans la charge de travail :

- Réécriture de la requête par rapport au nouveau schéma (l'entrepôt + les fragments), ce qui signifie que la requête passera par un processus de réécriture.

Exemple : Soit la requête q1 :

```
SELECT a1, a2, b1, c1, c2,c3 FROM a, b, c
WHERE fk_c0= fk_a AND fk_c1 = pk_b
AND a1 > 1000 GROUP BY c3
```

Supposons que la table à fragmenter est a, le schéma de la fragmentation est le schéma (défini dans le chromosome) qui fragment la table en deux fragments :

frag_a1 [pk_frag_a1, a1, a3, a4] et frag_a2[pk_frag_a2 , a2].

La nouvelle requête q1' est :

```
SELECT a1, a2, b1, c1, c2, c3 FROM frag_a1, frag_a2, b, c
WHERE fk_c0 = pk_frag_a1 AND pk_frag_a1 = pk_frag_a2
AND fk_c1 = pk_b AND a1> 1000
GROUP BY c3
```

Nous avons traité les cas suivants dans la réécriture des requêtes :

– Les attributs sont dans le même fragment, dans ce cas, il n'y aura pas d'ajout des jointures.
– Les attributs sont dans des fragments différents (comme dans l'exemple), dans ce cas, nous ajoutons les jointures nécessaires.

- Évaluation de la nouvelle requête q1' en se basant sur des estimations des opérations de sélection/projection et de jointure, cette étape nécessite de

parser la requête q1' et de construire des sous requête de sélection dans une liste de requêtes, et une autre liste pour les jointures.

- Calculer l'estimation de chaque sous requête de sélection/projection, on a utilisé les métas données (nombre de tuples, taille moyenne des attributs, nombre de valeurs distincts) pour estimer le nombre d'E/S pour chaque sélection et projection, nous avons pris en considération les opérations =, >, =>, <= et pour chacune on effectue l'estimation de la taille du résultat de sélection/projection.

- À partir de la liste des jointures, nous cherchons la première jointure à effectuer et qui donne les résultats intermédiaires les plus petites (on se basant sur le calcul de coût de l'opération de jointure par hachage 3. (|R|+|S|)).

- Après, nous réitérons pour chaque résultat intermédiaire pour effectuer les jointures restent, nous utilisons ici la technique de sélectivité.

- le coût de la requête sera le coût total de chaque sous requête de sélection/projection et de jointure.

- Le coût (fitness) d'un chromosome est le total des coûts des requêtes. FIN SINON

4. L'AG se termine et il retourne le chromosome qui a un coût minimum.

5. Conversion de chromosome en un schéma de fragmentation et génération du script

SQL qui crée les tables (fragments).

6. L'utilisateur peut générer la liste de requêtes réécrites par rapport au nouveau schéma.

FIN.

Les tests sous Oracle sont effectués comme suit :

- Nous lançons le script pour créer les fragments
- Nous analysons les requêtes réécrites avec EXPLAIN PLAN et on récupère les valeurs IO_COST du plan et nous comparons avec celle des requêtes sur le schéma non fragmenté.

Remarque :

Nous avons effectué des tests sur plusieurs requêtes pour comparer le plan d'exécution (L'ordre des jointures) dans Oracle avec notre programme.

L'architecture d'implémentation que nous avons optée est donnée par la figure 5.1.

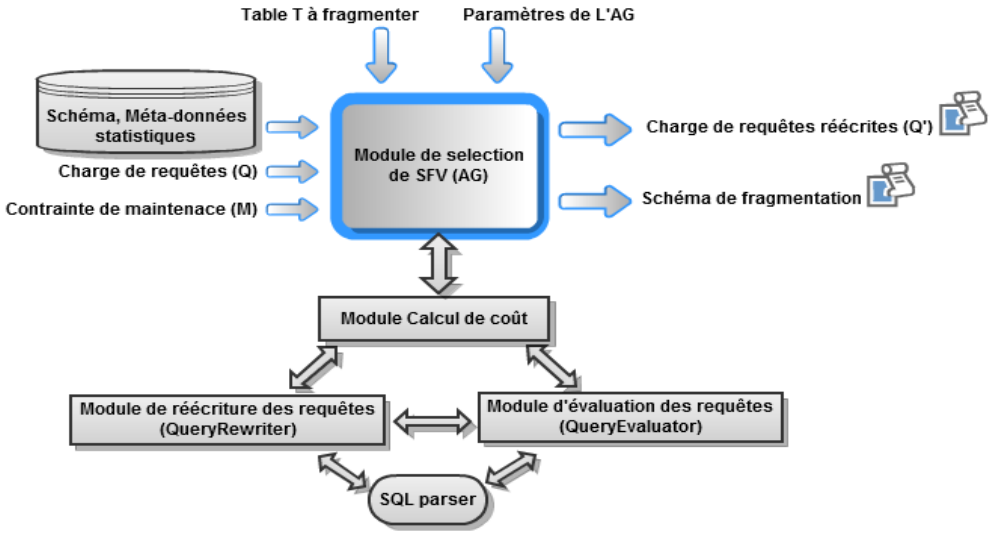

Figure 5.1: Architecture d'implémentation

On constate que notre programme nous a donné dans la plupart des cas le même ordre qu'Oracle.

La figure 5.2 représente l'écran de chargement des informations de l'entrepôt de don- nées et ces métadonnées à partir du SGBD.

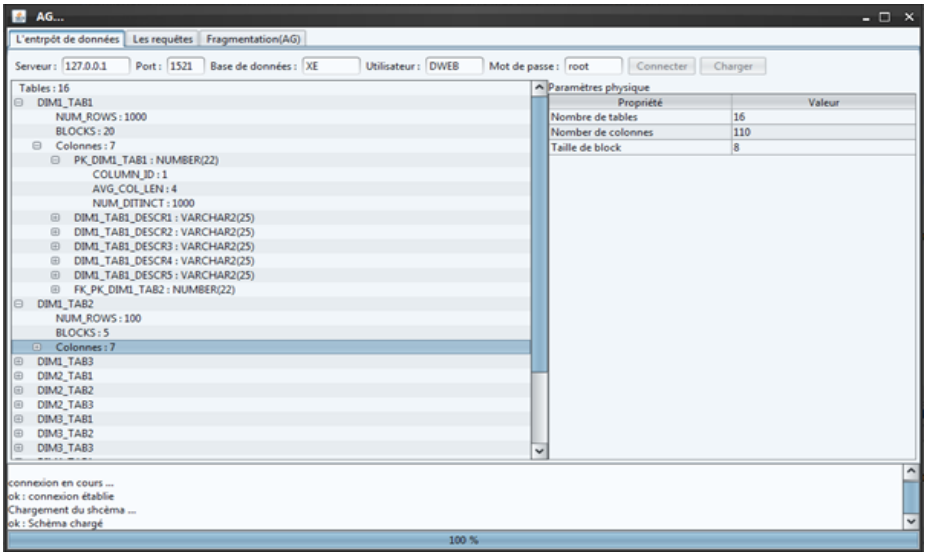

Figure 5.2: Chargement du schéma de l'ED et lecture des métadonnées

Nous avons chargé la liste des requêtes à partir d'un fichier bien structuré.

L'application donne le coût de chaque requête à l'aide des modules de calcul du coût, QueryEvaluator, et le SQL Parser. Ce qui montre la figure suivante :

Dans la figure 5.4, nous avons paramétré l'AG (Population initiale, Nombre d'essai, Nombre Maximum de fragment et la table à fragmenté). Ensuite, nous lançons l'AG.

Dans chaque itération, l'AG fournit l'évaluation optimale ainsi que le chromosome (Le schéma de fragmentation) optimale qui lui correspond.

Le schéma de fragmentation de la dernière itération est considéré comme le schéma de FV optimal pour la table choisit.

A la fin d'exécution, le programme donne le schéma de fragmentation de la table. Dans la figure 5.5, la table DIM_1_TAB1 va être fragmentée en deux fragments : FRAG0_DIM1_TAB1 et FRAG1_DIM1_TAB1.

Le programme permet la réécriture des requêtes chargé pour le nouveau schéma de l'entrepôt. Cette tâche de réécriture peut être lancée à la fin d'exécution de l'AG par l'utilisateur. Ce dernier, peut aussi générer le script SQL qui permet la création des nouvelles tables (Fragments).

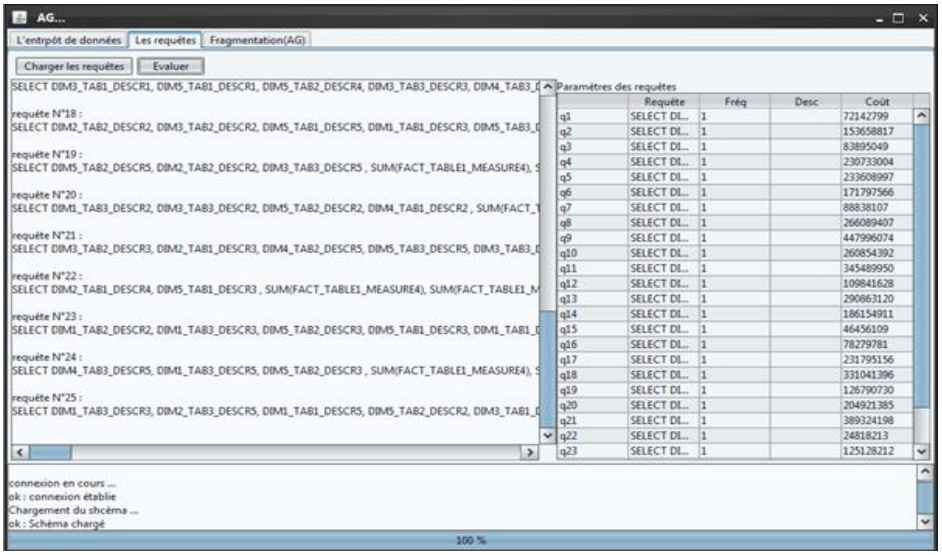

Figure 5.3: Chargement et évaluation des requêtes avant la FV

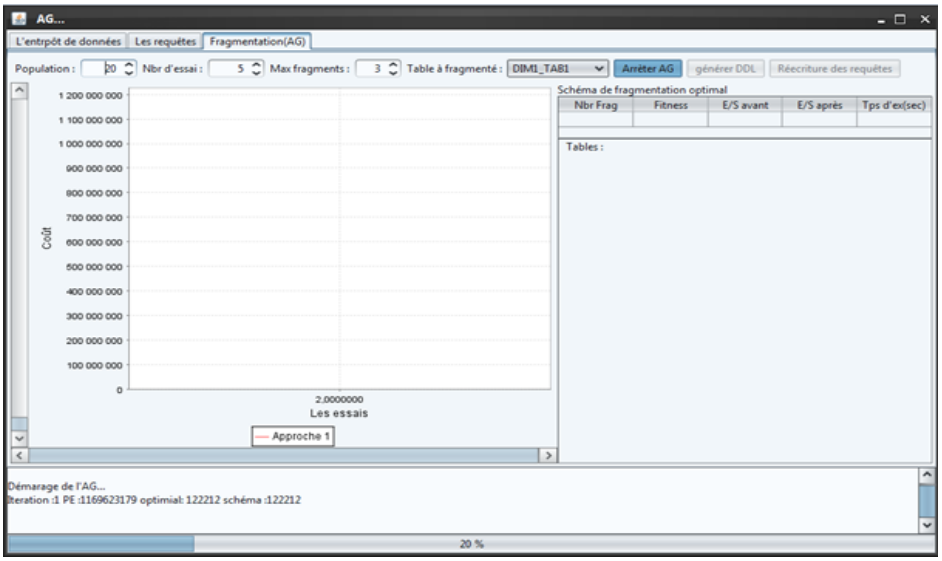

Figure 5.4: Paramétrage de l'AG et démarrage d'exécution

5.3 Etude expérimentale

Afin de tester notre approche de fragmentation, nous avons effectué une étude expérimentale sur le banc d'essais DWEB [95] et le SGBD Oracle 10G, avec une machine Intel Core2Duo et une mémoire de 2GO.

Le premier schéma est en étoile, il possède les tables suivantes :

- Table de faits FACT_TABLE(10000000 tuples),
- Quatre tables de dimension :
  - DIM1_TAB1(5000000 tuples)
  - DIM2_TAB1(1000 tuples)
  - DIM3_TAB1(1000 tuples)
  - DIM4_TAB1(1000 tuples)

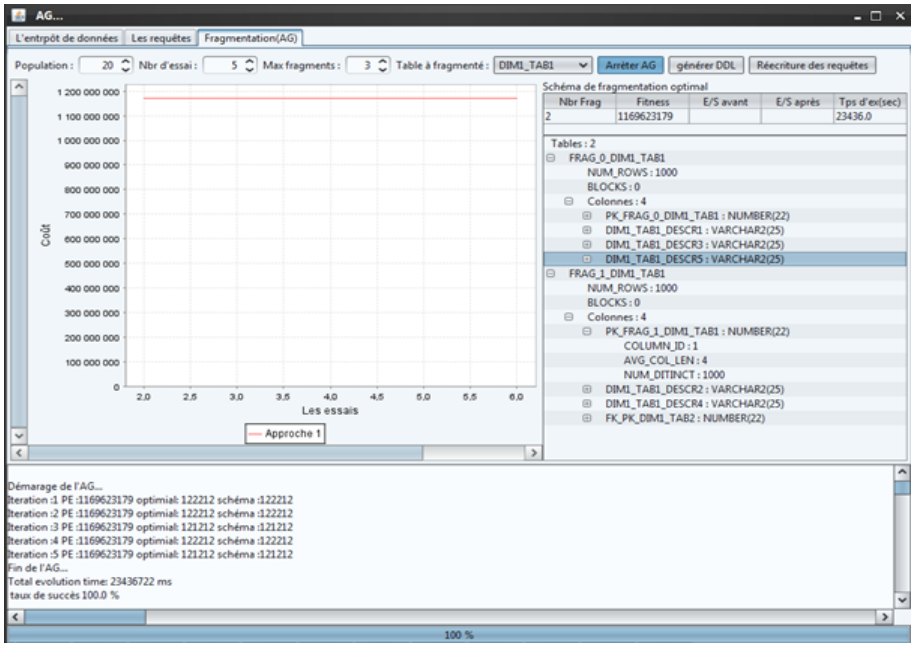

Figure 5.5: Fin d'exécution de l'AG et affichage du schéma de FV optimal

Le deuxième schéma est en flocon de neige, il contient les tables suivantes :

- table de faits FACT_TABLE(1000000 tuples),
- Quatre dimensions de trois niveaux :
  - DIMi_TAB1(5000 tuples),
  - DIMi_TAB2(100 tuples),
  - DIMi_TAB3(10 tuples)

Nous avons utilisé le modèle de coût (présenté dans la section 4.3.2) pour effectuer notre étude expérimentale sur la sélection d'un schéma de fragmentation verticale. Le coût représente le nombre d'entrées/sorties nécessaire pour exécuter la charge de requêtes. Nous avons validé cette évaluation sous Oracle 10G par une implémentation des structures sélectionnées par l'AG sur l'entrepôt, puis nous avons calculé le coût réel de la charge de requêtes en présence de ces structures.

Nous avons mené plusieurs expériences, nous voulons savoir l'intérêt de la fragmentation verticale dans la figure 5.6, l'axe des abscisses montre les différentes requêtes et l'axe des ordonnées montre le nombre d'opérations d'E/S.

Dans le premier test, la fragmentation a été effectuée sur la table DIM 5_TAB1(5000 tuples) du schéma en flocon de neige avec une charge de 10 et 25 requêtes choisis aléatoirement.

Les paramètres de l'AG sont (Population = 20. Nombre d'essai = 5. Nombre de fragments max = 3).

Le schéma de FV de la table DIM5_TAB1 trouvé par l'AG est composé de trois fragments :

F0_DIM5_TAB1 : f_PKF0_DIM5_TAB1, DIM5_TAB1_DESCR1, DIM5_TAB1_DESCR4, DIM5_TAB1_DESCR5

F1_DIM5_TAB1 : f_PKF1_DIM5_TAB1, DIM5_TAB1_DESCR2, DIM5_TAB1_DESCR4

F2_DIM5_TAB1 : f_PKF2_DIM5_TAB1, FKPK_DIM5_TAB2

Les figures 5.6 et 5.7 montrent que la fragmentation verticale est intéressante pour l'ensemble des requêtes. Les résultats de tests ont montré que la FV a donné un gain de 222 et 4506 opérations d'E/S pour une charge de 10 et 25 requêtes respectivement.

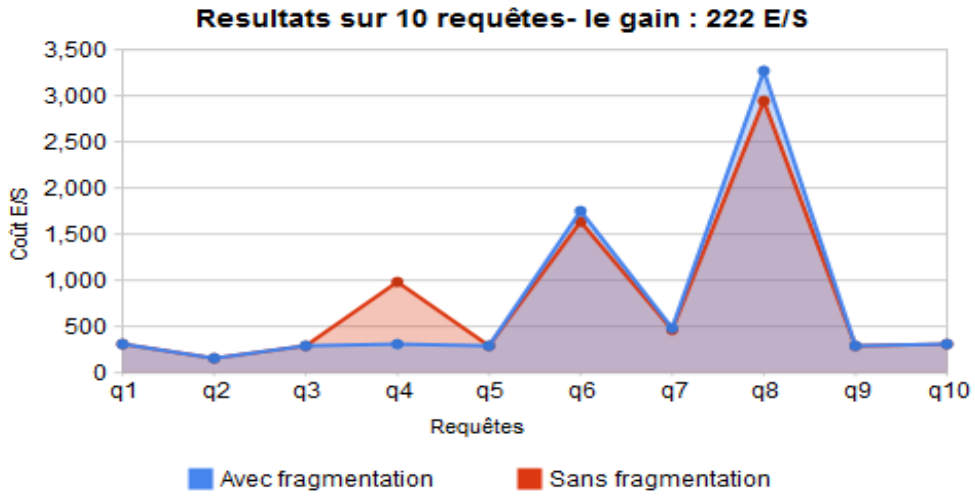

Figure 5.6: Coût d'E/S avec et sans FV, 10 requêtes

Dans le deuxième test, nous nous intéressons à savoir l'impact de la taille de la table à fragmenter sur les coûts d'E/S. Nous avons appliqué la FV sur la table DIM1_TAB1 du schéma en étoile avec les paramètres de l'AG (Population = 80, Nombre d'essai = 8, Nombre de fragments max = 4) et une charge de requête égal à 50.

Le schéma de FV de la table DIM1_TAB1 obtenu par l'AG est composé de trois fragments :

F0_DIM1_TAB1 : PK_F0_DIM1_TAB1 , DIM1_TAB1_DESCR1 , DIM1_TAB1_DESCR4, DIM1_TAB1_DESCR7
F1_DIM1_TAB1 : PK_F1_DIM1_TAB1, DIM1_TAB1_DESCR2 , DIM1_TAB1_DESCR3
F2_DIM1_TAB1 : PK_F2_DIM1_TAB1, DIM1_TAB1_DESCR5 , DIM1_TAB1_DESCR6

La figure 5.8 montre l'intérêt de la fragmentation verticale d'une table de dimension de grande taille. Nous remarquons que la FV pourrait être intéressante dans le cas où la taille des tables de dimension est importante (le cas d'un schéma en étoile).

Cette fragmentation pourrait réduire le coût des opérations de jointures en étoile.

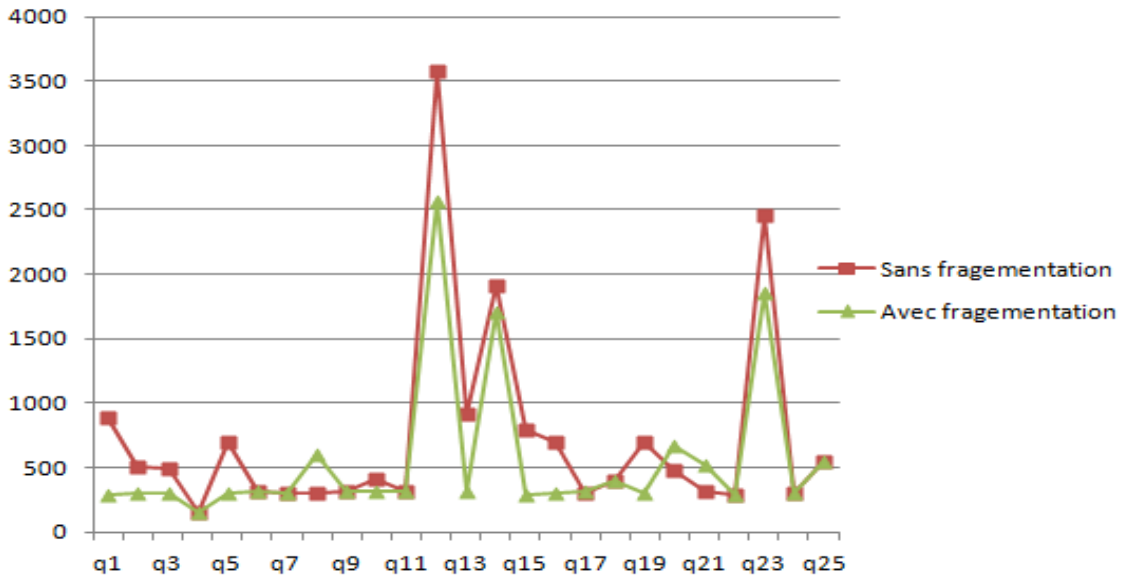

Figure 5.7: Coût d'E/S avec et sans FV, 25 requêtes

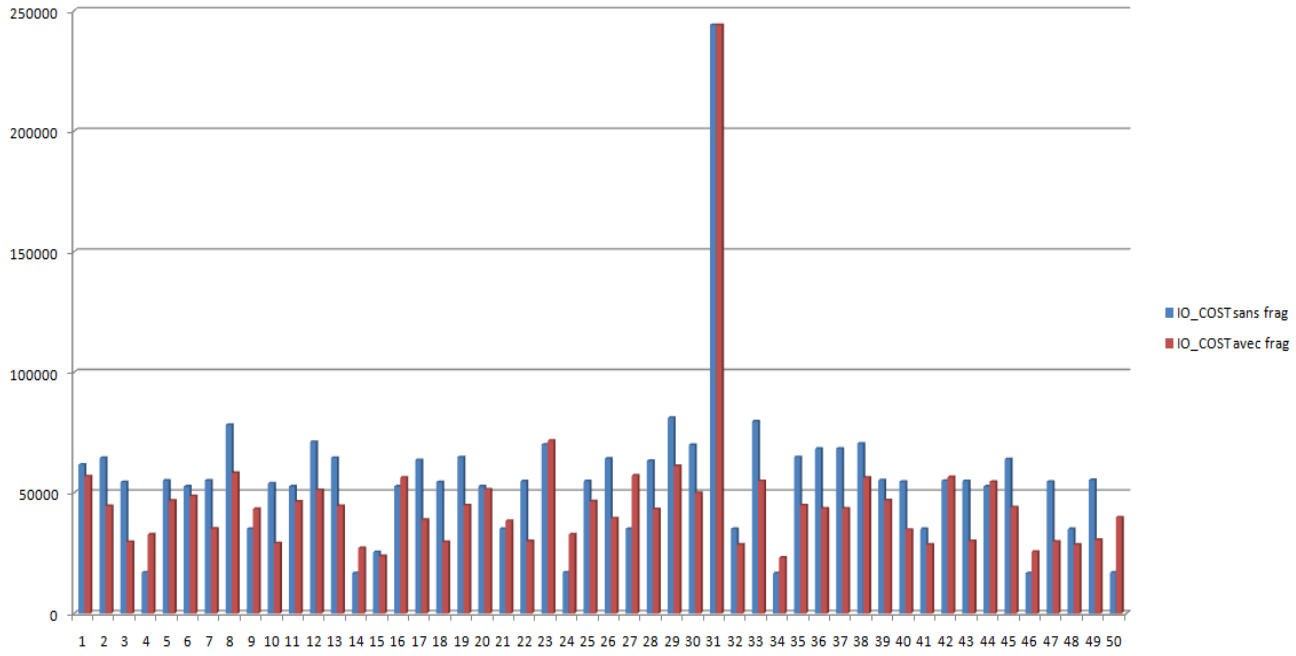

Figure 5.8: Le coût d'E/S avant et après la FV (gain = 491047 E/S)

5.4 Conclusion

Nous avons présenté dans ce chapitre une implémentation de notre algorithme pour la sélection d'un schéma de fragmentation verticale. L'implémentation a été développée en trois modules principaux : module de sélection d'un SFV, module d'évaluation des requêtes, module de réécriture des requêtes.

On a réalisé une étude expérimentale basée sur un modèle de coût calculant le nombre d'entrées/sorties nécessaire pour exécuter un ensemble de requêtes. Deux schémas ont été choisit pour les tests : Schéma en étoile et un schéma en flocon de neige. A la fin, nous avons présenté quelques résultats de FV

CONCLUSION ET PERSPECTIVES

Les systèmes décisionnels manipulent de très importants volumes de données stockées dans des entrepôts de données. Ces derniers sont alimentés par des données provenant de sources distribuées et hétérogènes. Les entrepôts de données sont très souvent modélisés par un schéma en étoile. Ce schéma est caractérisé par une table de faits de très grande taille (allant de quelques Giga-octets à plusieurs téraoctets) liée à un ensemble de tables de dimension de plus petite taille. Les requêtes définies sur un schéma en étoile (connues par requêtes de jointure en étoile) sont caractérisées par des opérations de sélection sur les tables de dimension, suivies de jointures avec la table des faits. Toute jointure doit passer par la table des faits, ce qui rend le coût d'exécution de ces requêtes très important. Sans technique d'optimisation, leur exécution peut prendre des heures, voire des jours.

Dans ce mémoire, nous avons considéré le problème de la fragmentation verticale dans les entrepôts de données qui est un problème d'optimisation. Nous avons remarqué que la plus part des algorithmes de fragmentation verticale [2, 10, 11, 20, 24] contiennent deux parties essentielles : une méthode d'optimisation et une fonction objectif. La résolution exacte du problème est difficile lorsque la taille du problème est grande, plusieurs algorithmes de FV utilisent des heuristiques pour la création des fragments d'une relation [9, 22, 11, 45], d'autres utilisent des méthodes mathématiques pour trouver le schéma de fragmentation optimal [2, 12, 14, 55].

Deux sortes de fonctions d'objectives utilisées pour des algorithmes de fragmentation verticales, qui sont 1) les fonctions de modèles de coût basées sur l'analyse d'accès des transactions sur le système de gestion de bases de données et 2) les fonctions fondées sur des hypothèses empiriques. La première forme de fonction objective est spécifique au système de gestion de bases de données fondamental tandis que la deuxième est plus générale et intuitif.

Nous avons remarqué grâce à des exemples traités que les résultats obtenus par les algorithmes de fragmentation verticale sont souvent différentes voire pour la même matrice d'affinité, ce qui montre que les fonctions d'objectifs de ces algorithmes sont aussi différentes.

Nous avons proposé un algorithme de sélection d'un schéma de fragmentation verticale d'une table de dimension basée sur les AG. Une implémentation de cet algorithme et des tests d'expérimentation ont été réalisés à l'aide du banc d'essai DWEB et le SGBD Oracle 10G. Les résultats obtenus montrent l'impact de la FV dans les entrepôts de données relationnels et le choix de la table fragmentée.

Comme perspectives, il serait intéressant d'adapter notre approche et la testée pour fragmenter verticalement plusieurs tables de dimension. Le choix de la table à fragmenter se fait d'une manière manuel. De plus, automatiser cette tâche en se basant sur l'usage des tables par rapport à la charge des requêtes.

Appliquer d'autres heuristiques (ou méta heuristique) pour la résolution du problème et comparer les résultats sur un banc d'essai important.

RÉFÉRENCES

[1] M-C. Wu and A. Buchmann. Research issues in data warehousing. In Daten bank- systeme in Bro, Technik und Wissenschaft(BTW'97), pages 61 82, March 1997.

[2] Inmon et Hackarton, Using the Data Warehouse. John Wiley & Sons, 304 p.,ISBN : 0471059668, 1994.

[3] Chu, W. W. and Ieong, I.T., "A Transaction-Based Approach to Vertical Partitio- ning for Relational Database Systems", IEEE Transactions on Software Engineering, 19-9, August 1993.

[4] Ladjel Bellatreche , Techniques d'optimisation des requêtes dans les data warehouses, LISI/ENSMA.

[5] M. Schkolnic. A Clustering Algorithm for Hierarchical Structures ACM TODS, Vol.1, No. 2, pp. 27-44. March 1977.

[6] D. Cornell, and P. Yu. A Vertical Partitioning Algorithm for Relational Databases. Proc. Third International Conference on Data Engineering, Feb. 1987, pp. 30-35. [7] H. Day. An Optimal Extracting from a Multiple File Data Storage System : An Ap- plication of Integer Programming Operations Research, 13, 482-494, 3(May 1956).

[8] P. Dearnley. Model of a Self-organizing Data Management System Computer Journal Vol. 17, No. 1, Feb. 1974.

[9] M. Eisner, and D. Severance. Mathematical techniques for efficient record segmentation in large shared databases. J. ACM 23, 4(Oct. 1976).

[10] M. Hammer, and B. Niamir. A heuristic approach to attribute partitioning. In Proceedings ACM SIGMOD Int. Conf. on Management of Data (Boston, Mass.,1979), ACM, New York.

[11] J. Hofer, and D. Severance. The Uses of Cluster Analysis in Physical Database Design . 1st International Conference on VLDB, Framingham,

MA, 1975, pp. 69 -86.

[12] J. Hoffer, An integer programming formulation of computer database design pro- blems. 1976.

[13] A. Jain, and R. Dubes. Algorithms for clustering Data. Prentice Hall Advanced

Reference Series, Englewood Cli_s, NJ, 1988.

[14] R. Kennedy. The Use of Access Frequencies in DataBase Organization, Ph.D Dis- sertation, The Wharton School, Univ. of Pennsylvania, 155 pp, May 1973 .

[15] R. Kennedy, A File Partitioning Model, Cal. Tech. Information Science Tech. Report 2, May 1972.

[16] B. Niamir. Attribute Partitioning in Self-Adaptive Relational Database System. Master's Thesis, M.I.T. Lab. for Computer Science, Jan. 1978.

[17] S. Lu, and K. Fu. A sentence-to-sentence clustering procedure for pattern analysis. IEEE Transactions on Systems, Man and Cybernetics SMC 8, 381-389.

[18] W. McCormick, P. Schweitzer, and T. White. Problem Decomposition and Data Reorganization by a Clustering technique Operations Research, 20 Sep. 1972.

[19] S. March, and D. Severance. The determination of efficient record segmentation and blocking factors for share data _les. ACM Trans. Database Syst. 2, 3(Sept.1977).

[20] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou. Vertical Partitioning Algorithm for Database Design ACM Transactions on Database Systems, Vol. 9,No. 4, Dec, 1984.

[21] S. March, and D. Severance. The determination of efficient record segmentation and blocking factors for share data files. ACM Trans. Database Syst. 2, 3(Sept. 1977).

[22] S. Navathe, and M. Ra. Vertical Partitioning for Database Design : A

Graphical Algorithm. ACM SIGMOD, Portland, June 1989.

[23] Navathe, S. B., Karlapalem, K., & Ra, M. A mixed fragmentation approach for initial distributed database design. Journal of Computers and Software Engineering, 3(4).1995.

[24] M. Osman. The Partitioning of a DataBase into Supplies Matching User's Queries Computer Center, University of Khartoum Sudan.

[25] S. Ceri, S. Pernici, and G. Weiderhold. Optimization Problems and Solution Methods in the Design of Data distribution. Information Sciences Vol 14, No. 3, p261-272, 1989.

[26] Y. Seppala. Definition of Extraction Files and Their Optimization by Zero-one programming. BIT, 7, 206-215, 3(1967).

[27] Cauvet, C., Rosenthal-Sabroux, C. : Ingénierie des systèmes d'information sous la direction de Corine Cauvet, Camille Rosenthal-Sabroux. Paris Hermès Science Publications, France, 2001.

[28] M. Stocker and A. Dearnley. Self-organizing Data Management Systems, Computer Journal. 16. 1973.

[29] N. Tamer, P. Valduriez. Principles of Distributed Database Systems. Prentice Hall Englewood Cli_s, New Jersey 07362.

[30] Marcel P. Manipulation de Données Multidimensionnelles et Langages de Règles, Thèse Doctorat de l'Institut des Sciences Appliquées de Lyon, 1998.

[31] C. Yue and K. Wong. On the Optimal Properties of a Partition Algorithm IBM Research Report, RC 3797, March 30, 1972

[32] C. Zahn. Graph-theoretical methods for detecting and describing Gestalt Clusters.IEEE Transactions on Computers C 20, 68-86.

[33] lsr.imag.fr/Les.Personnes/Herve.Martin/HTML/FenetrePrincipale.htm

[34] E. Malinowskil et S. Chakravarthy. Fragmentation Techniques for Distributing Object-Oriented Databases.

[35] M. T. Ozsu and P. Valduriez. Principles of Distributed Database Systems : Second Edition. Prentice Hall, 1999.

[36] A. Alsberg. Space and Time Savings through Large Database Compression and Dynamic Restructuring. Proceedings of the IEEE, Vol 63, No. 8, August 1975.

[37] P. O'Neil and D. Quass. Improved query performance with variant indexes. Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 38 49, May 1997.

[38] A. Datta, K. Ramamritham, and H. Thomas. Curio : A novel solution for Efficient storage and indexing in data warehouses. Proceedings of the International Conference on Very Large Databases, pages 730{733, September 1999.

[39] Jun Du & Reda Alhajj & Ken Barker, Genetic algorithms based approach to database vertical partition. 2006.

[40] Song, S.K. and Gorla, N., "A genetic Algorithm for Vertical Fragmentation and Access Path Selection," The Computer Journal, vol. 45, no. 1, 2000, pp 81-93.

[41] Erez Hartuv & Ron Shamir , "A Clustering Algorithm based on Graph Connectivity", School of Computer Science , Sackler Faculty of Exact Sciences , 1999.

[42] Cheng, C-H ; Lee, W-K ; Wong, K-F, "A Genetic Algorithm-Based Clustering Approach for Database Partitioning," IEEE Transactions on Systems, Man, and Cybernetics, 32(3), 2002, 215-230.

[43] Eltayeb Salih , An Optimized Scheme for Vertical Partitioning of a Distributed Database , IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.1, 2008.

[44] Inmon W.-H., Zachman John A., Geiger Jonathan G. : Data Stores, Entrepôts and the Zachman Framework, 1997.

[45] Ezeife and K. Barker. Vertical class fragmentation in a distributed based system. Technical Report 94-03, University of manitoba, 1994.

[46] M. Savonnet, Fragmentation et distribution de classes d'objets dans les bases de données Orientées Objets, Université de Dijon, 1995.

[47] J. Nachouki et H. Briand. A mixed approach for distributed object-oriented data bases design. In. ISCIS X The tenth international symposium on computer and information science, pages 209-216, 1995.

[48] S. Chakravarthy, J. Muthuraj, R. Varadarajan et S. B. Navathe. An Objective Function for Vertically Partitioning Relations in Distributed Data Base and it Analysis. Distributed and Parallel Databases. Université de Florida. 1993.

[49] J. Muthuraj , A formal approach to the vertical partitioning problem in distributed database design. thèse, Université de Florida, 1992.

[50] Theodoratos, D., Bouzeghoub, M. : A general framework for the view selection problem for entrepôt design and evolution. DOLAP 00, ACM Third International Workshop on Data Warehousing and OLAP, 2000.

[51] M-C. Wu and A. Buchmann. Research issues in data warehousing. in Datenbank- systeme in Bro, Technik und Wissenschaft(BTW'97). 1997.

[52] Patrick Valduriez. Join Indices. ACM tinsactions on Database Systems, 12(2) :218- 246, June 1987.

[53] A. Datta, K. Ramamritham, and H. Thomas. Curio : A novel solution for Effi- cient Storage and Indexing in Data Warehouses. Proceedings of the International Conference on Very Large Databases, pages 730{733, September 1999.

[54] Munneke, D., Wahlstrom, K., & Mohania, M. K, Fragmentation of Multidimensio- nal Databases. 10th Australasian Database Conference (ADC 99), Auckland, New Zealand (pp. 153{164), 1999.

[55] Golfarelli, M., Maio, D., et Rizzi, S, Vertical Fragmentation of Views in Relational Data Warehouses. Settimo Convegno Nazionale su Sistemi Evoluti Per Basi Di Dati (SEBD 99), Como, Italy (pp. 19{33). 1999.

[56] Furtado, C., Lima, A., Pacitti, E., Valduriez, P., & Mattoso, M, Physical and virtual partitioning in OLAP database cluster. In Proceedings of the 17th International Symposium on Computer Architecture and High Performance Computing (pp. 143-150), 2005.

[57] Labio, W. J., Quass, D., & Adelberg, B, Physical database design for data ware- houses. In Proceedings of the IEEE Conference on Data Engineering

(pp. 277-288),1997.

[58] M. Golfarelli, V. Maniezzo, S. Rizzi, Materialization of fragmented views in multidi- mensional databases, Data & Knowledge Engineering, Volume 49, Issue 3, pp.325-351, 2004.

[59] G. Velinov, M. Kon-Popovska, D. Gligoroski, Optimization of Relational Data Wa- rehouses, Proc. 4th European Conference on Intelligent Systems and Technologies, ECIT2006, Iasi, Romania, September 2006.

[60] Cornell, D.W. and Yu, P.S., "An Effective Approach to Vertical Partitioning for Physical Design of Relational Databases", IEEE Transactions on Software Enginee- ring, 16-2, Feb 1990.

[61] Ruskey, F, Simple combinatorial gray codes constructed by reversing sublists, Al- gorithms and Computation, Lecture Notes in Computer Science 762, pp.201–208, Berlin Heidelberg New York : Springer. 1993.

[62] S. Chaudhuri et V. Narsayya, Index merging, Proceedings of the International Conferance on Data Engineering(ICDE), 1999.

[63] Yu, P. S., Chen, M. S., Heiss, H. U., & Lee, S. (1992). Workload characterization of relational database environments. IEEE Transactions of Software Engineering,18(4), 347-355.

[64] http ://mathforum.org/advanced/robertd/bell.html

[65] http ://fr.wikipedia.org/wiki/Séparation_et_évaluation

[66] M. Babad. A record and file partitioning model. Commun. ACM 20,1(Jan 1977).

[67] A Fast Index for Semistructured Data , in 27th International Conference on Very Large Data Bases (VLDB 2001), Roma, Italy, pp. 341350, 2001.

[79] R. Kimball. The Data Warehouse Toolkit. John Wiley, 1996.

[80] Red Brick Systems. Star schema processing for complex queries. WhitePaper, July 1997.

[81] S. Chaudhuri and V. R. Narasayya. Selftuning database systems : A decade of progress. In VLDB, pages 3–14, 2007.

[82] K. Aouiche. Techniques de fouille de données pour l'optimisation automatique des performances des entrepôts de données. Ph.d. thesis, Université Lumière Lyon 2, December 2005.

[83] 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.

[84] Ziyati. E, L.Bellatreche et K. boukhalfa « La contribution des structures d'opti- misation non redondantes dans la conception physique des entrepôts de données » algiers, Alger), ISPS'2007.

[85] Bouchakri.R, Une approche dirigée par la classification des attributs pour fragmen- ter et indexer des entrepôts de données, Thèse de magister, ESI, 2010.

[86] J.Alliot et N.Durand, "Algorithmes genetiques", 2005.

[87] G. M. JM. Steinbrunn et A. Kemper, "Optimizing join orders", Techreport MIP9307, Faculty of Mathematic, Univ. of Passau, Germany, 1993.

[88] Y. Chen, F. Dehne, T. Eavis, and A. Rau-Chaplin, "Improved Data Partitioning For Building Large ROLAP Data Cubes in Parallel", International Journal of Data Warehousing and Mining, Volume 2, Number 1, pages 1-26, 2006.

[89] H. Garcia-Molina, J. D.Ullman, et J. Widom, J, "Database Systems : The Com- plete Book", Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

[90] K. Boukhalfa, De la conception physique aux outils d'administration et de tuning des entrepôts de données, Thèse de doctorat, Université de Poitiers, 2009.

[91] Bellatreche, L. Utilisation des index et de la fragmentation dans la conception logique et physique d'un entrepôt de données, Thèse de Doctorat en Informatique, Université de Clermond Ferrand. 2000.

[92] Messaoudi,M. Ladjel Belatreche, L et Oukid, K,S, Fragmentation verticale dans un Entrepôt de données, SETIT 2011, Tunisie, 2011.

[93] Java Genetic Algorithms Package http://jgap.sourceforge.net

[94] http://www.sqlparser.com/gspjava.php

[95] http://eric.univ-lyon2.fr/jdarmont/?page=logiciel

[96] M.Hui, ”Distribution design for complex value databases”. a dissertation presented in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Information Systems at Massey University,2007.