REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE AMAR TELIDJI – LAGHOUAT –

Faculté des **S**ciences

Département de **M**athématique et **I**nformatique

MÉMOIRE

Présenté par :

ARIOUAT Youcef

Pour l'obtention du diplôme de MAGISTER en Informatique

Option : **I**nformatique **R**épartie et **M**obile (**IRM**)

THÈME

Une approche basée sur l'extraction de motifs sous contraintes pour la sélection des Index de Jointure Binaires

Soutenu le : /2013, devant le jury formé de :

Mr. M. B. YAGOUBI	Professeur	Université de Laghouat	Président
Mr. A. MOUSSAOUI	Professeur	Université de Sétif	Examinateur
Mme. H. BOUZOUAD	Maître de conférences	Université de Laghouat	Examinatrice
Mr. Y. OUINTEN	Maître de conférences	Université de Laghouat	Encadreur
Mr. B. ZIANI	Maître assistant	Université de Laghouat	Co- Encadreur

N° d'ordre : ...../2013-M/INF

Résumé

Les requêtes analytiques définies sur les entrepôts de données sont complexes et couteuses en temps d'exécution car elles nécessitent plusieurs jointures exécutées sur un large volume de données. Les Index de Jointure Binaires (IJB) sont l'une des techniques d'optimisation les plus utiles pour réduire le coût d'exécution de ces requêtes en pré-calculant leurs jointures. Toutefois, la sélection d'une configuration appropriée d'IJB est un problème difficile à résoudre vue la complexité de l'espace de recherche à parcourir. Le problème est classé comme NP-complet, c'est pourquoi la plus part des travaux traitant ce problème se sont concentrés principalement sur la proposition de solutions d'élagage de l'espace de recherche par le biais de techniques de data mining ou des stratégies heuristiques. Le principal inconvénient de ces approches est que le processus de sélection des index s'effectue en deux étapes. *La génération d'un grand nombre d'index*, suivie d'une *phase d'élagage.*

Une alternative est de contraindre les données d'entrée plus tôt dans le processus de sélection, réduisant ainsi l'ensemble des index en sortie à ceux qui présentent un intérêt pour l'administrateur. Par exemple, pour sélectionner un ensemble d'index, l'administrateur peut mettre des limites sur le nombre d'attributs ou la cardinalité des attributs à inclure dans la configuration d'index qu'il cherche.

Dans ce travail, nous abordons le problème de sélection d'IJB en utilisant une approche d'extraction de motifs sous contraintes. Contrairement aux approches précédentes, la sélection est effectuée en une seule étape en introduisant des contraintes dans le processus de sélection. L'approche proposée est implémentée sous forme d'un outil d'aide à l'administration et évaluée en utilisant le benchmark APB-1. Les expérimentations menées ont montré que la configuration d'index générée en une seule étape permet d'avoir un important gain de performance et les tests comparatifs ont montré que les résultats obtenus sont comparables à ceux des approches de sélection d'IJB procédant en deux étapes.

Mots clés : Entrepôt de données, index de jointure binaire, extraction de motifs sous contraintes .

Abstract

Analytical queries defined on data warehouses are complex and very time consuming since they need several join operations that are very costly, especially when run on very large data volumes. Bitmap Join Indexes (BJI) are useful to pre-calculate these joins in order to reduce the execution cost. However, selecting a suitable configuration of BJI is a difficult problem to solve, since it needs to explore a large search space. The problem is classified as NP-complete; this is why most studies dealing with this problem mainly focused on proposing pruning solutions of the search space by the means of data mining techniques or heuristic strategies.

The main shortcoming of these approaches is that the indexes selection process is performed in two steps. The generation of a large number of indexes is followed by a pruning phase. An alternative is to constrain the input data earlier in the selection process thereby reducing the output size to a set of indexes that are of interest for the administrator. For example, to select a set of indexes, the administrator may put limits on the number of attributes or the cardinality of the attributes to be included in the indexes configuration he is seeking. In this work we addressed the bitmap join indexes selection problem using a constraint-based itemset mining Approach. Unlike previous approaches, the selection is performed in one step by introducing constraints in the selection process. The proposed approach is evaluated using APB-1 benchmark. The proposed approach is implemented as an aid administration tool and evaluated using benchmark APB-1. The experiments conducted have shown that the index configuration generated in one step allowed a significant improvement of the execution time of the workload and the comparative tests have shown that the results obtained are comparable to those obtained by other approaches for selecting IJB proceeding in two steps.

Keywords: Data Warehouse, Bitmap join index, Constraint-based itemset mining.

ملخص

الاستعلامات التحليلية المعرفة على مستودعات البيانات هي معقدة ومكلفة في وقت التنفيذ لأنها تتطلب عدة عمليات ضم مكلفة جدا، وخصوصا عندما يتم تنفيذها على كمية كبيرة من البيانات. فهارس الضم الثنائية (IJB) مفيدة جدا لخفض تكلفة الآداء عن طريق الحساب القبلي لعمليات الضم. ومع ذلك، فإن اختيار التكوين المناسب من IJB هي مشكلة صعبة التعقيد وتصنف NP ، لذلك معظم الاعمال التي تعامت مع هذه المشكلة ركزت أساسا على تشذيب فضاء البحث عن طريق تقنيات التنقيب في البيانات أو استراتيجيات الاستدلال. والعيب الرئيسي لهذه المقاربات هو أن عملية اختيار الفهارس تتم في خطوتين: 1- توليد عدد كبير من الفهارس، 2- تليها مرحلة من التشذيب.

والبديل الذي نراه، هو تقييد البيانات في وقت سابق في عملية توليد الفهارس للحد من حجم الناتج الى مجموعة من الفهارس التي تهم المسؤول عن مستودعات البيانات. على سبيل المثال، يمكن للمسؤول تعيين حدود على عدد السمات في كل فهرس أو تعداد كل سمة تندرج في تكوين الفهارس التي يريدها.

في هذا الإطار، نقترح التصدي لمشكلة اختيار IJB باستخدام مقاربة تستعمل "**التنقيب في البيانات في ظل القيود**". على عكس المقاربات السابقة، يتم الاختيار في خطوة واحدة عن طريق إدخال قيود في عملية الاختيار.

تم تجسيد المقاربة المقترحة كأداة مساعدة لإدارة مستودعات البيانات وتقييمها باستخدام مقياس الأداء APB-1. وقد أظهرت التجارب التي أجريت أن تكوين الفهارس المتولدة في خطوة واحدة يمكن أن يكون لها مكاسب كبيرة في الآداء. كما أن اختبارات المقارنة أظهرت أن النتائج تماثل بل وأحيانا أحسن من تلك الناتجة عن المقاربات التي تعتمد على مرحلتين في اختيار الفهارس IJB.

كلمات البحث :مستودع البيانات، فهارس الضم الثنائية، التنقيب في البيانات، التنقيب في البيانات في ظل القيود.

Remerciements

En tout premier lieu, nous remercions Allah le tout puissant, à la sagesse et au savoir infinis, " *Gloire à Toi ! Nous n'avons de savoir que ce que Tu nous as appris. Certes c'est Toi l'Omniscient, le Sage.* " (Sourate al-Baqarah, verset 32). Et nous remercions Le Prophète Mohammed (SAW) qui nous a montré la lumière de la foi.

Je remercie sincèrement Mr. Youcef OUINTEN, Maître de conférences à l'Université de Laghouat et Mr. Benameur ZIANI, Maître assistant à l'Université de Laghouat pour leur encadrement, leur collaboration et surtout pour leur soutien sans faille. Je les remercie d'avoir eu confiance en ce mémoire, ainsi que pour leur très grande disponibilité. Je voudrais particulièrement les remercier pour leurs remarques très pertinentes, leurs qualités humaines et scientifiques qui m'ont permis de mener à bien et avec grand plaisir et liberté ce travail, Qu'ils veuillent bien donc trouver ici le témoignage de ma profonde reconnaissance.

Je souhaite également adresser mes remerciements à l'ensemble des membres du jury qui me font le grand honneur d'avoir accepté de juger mon travail.

Mes remerciements vont aussi à mes collègues et responsables de l'Université de Djelfa pour leur soutien moral et informationnel,

J'oublie pas mes collègues de l'école doctorale IRM avec qui j'ai partagé des moments inoubliable et à qui je souhaite beaucoup de réussite. Et en particulier je remercie Mr. Benmlouka Ahmed de m'avoir accompagner et encourager tout au long de ce travail.

Mes remerciements sont particulièrement adressés à mes amis : El'khier, Ilies, Oussama, Abdou, Adel, Wahab et Abdelhafid. Vous êtes plus que des amis, vous êtes des frères avec le sens profond de l'amour fraternel qui est devenu notre parole. Les mots ne suffisent pas pour exprimer l'intensité de l'affection et de la reconnais-

sance que j'ai pour vous.

Et surtout,
Merci à mes parents (Ma Mère, Mon Père). Je leur serai éternellement reconnaissant de m'avoir assurer de leur confiance, de leurs encouragements et de leur soutien sans limite. Merci aussi à mes frères et sœurs pour tout l'aide qu'ils m'ont apporté, c'est un exemple de la famille à suivre.

Enfin, Merci à ceux que je n'ai pu citer mais qui ont toutes mes amitiés et mes remerciements.

Dédicace

Je dédie ce mémoire pour l'âme de mon très cher frère et ami, le défunt « Ali MERZOUG ».

Cher ami, J'aurai souhaité que vous soyez à mes côtés aujourd'hui mais le Bon Dieu a choisi autrement.

Vous êtes parti trop tôt, je me souviendrai de vous toute ma vie.

Ton souvenir restera à jamais gravé dans mon cœur.

Ta générosité et ta sympathie me serviront de fil conduit.

Je prie Dieu qu'il vous fasse beaucoup de miséricorde, de clémence et que la terre vous soit légère.

Que Le Prophète Mouhamed soit votre tuteur.

Que Dieu vous accorde le paradis le plus haut... Amin

Table des matières

Table des figures

Liste des tableaux

Introduction

Contexte et problématique

Issus de l'industrie pour servir de base à l'analyse des données, *les entrepôts de données* ("*data warehouses*") sont devenu aujourd'hui des outils incontournables dans plusieurs domaines (universités, compagnies aériennes, banques, compagnies d'assurances ...). Un entrepôt de données peut être vu comme une base de données qui a pour but de regrouper et d'homogénéiser de grands ensembles de données, provenant de sources hétérogènes pour des fins d'*analyse* et d'*aide à la décision*.

Les entrepôts de données sont souvent modélisés selon un *schéma en étoile* dans lequel une table centrale de faits très volumineuse (peut atteindre des Terra Octets) et contenant les faits à analyser, référence par des clés étrangères des tables dites de dimensions qui représentent les axes d'analyse. Les données stockées dans un entrepôt modélisé selon ce schéma sont interrogées par des requêtes décisionnelles appelées requêtes de jointure en étoile. Ces requêtes sont complexes et nécessitent un temps d'exécution qui peut atteindre plusieurs heures ou jours à cause de la volumétrie des données et des multitudes opérations de jointures en étoiles entre la table de fait et les tables de dimensions. Un tel temps de réponse est inacceptable par un analyste ou un décideur, il est donc crucial de faire appel à des structures et des techniques d'optimisation pour le réduire.

Parmi les structures d'optimisation, les *Index de Jointure Binaires* ont montré leur efficacité à réduire le temps d'exécution des requêtes décisionnelles car ils précalculent les résultats d'exécution des jointures contenues dans ces requêtes et ils optimisent les sélections en évaluant la conjonction ou la disjonction des prédicats contenus dans ces requêtes à travers des opérations logiques. Ils représentent aussi l'avantage de consommer un minimum d'espace de stockage à cause de leur représentation binaire. Cependant, vu le nombre important des tables de dimensions et des attributs susceptible d'être indexés, la sélection des index à créer est un problème connu NP-complet. De ce fait, il n'existe pas d'algorithme qui propose une solution optimale en un temps raisonnable. Plusieurs travaux de recherche utilisent des heuristiques ou des techniques de datamining pour réduire la complexité du

problème et proposer des solutions proches de la solution optimale. Généralement ces approches procèdent en deux étapes : *sélection d'une configuration initiale* dans le but de réduire la complexité du problème, suivie d'une deuxième étape d'élagage qui permet la *sélection d'une configuration finale d'index*.

Objectifs

Nous proposons dans ce mémoire une approche de sélection des index de jointure binaires basée sur une technique de datamining appelée *l'extraction de motifs sous contraintes*. Plusieurs travaux de recherche ont traité le problème de sélection d'index en utilisant des techniques de datamining ; cependant, ces travaux ne prennent pas en compte un élément clé dans tout processus d'extraction de connaissances à partir des bases de données qui est l'*expertise humaine*. En effet, les processus d'extraction des connaissances sont principalement interactifs, l'utilisateur doit intervenir du bout en bout de ces processus pour préparer les données, choisir la bonne technique d'extraction, le paramétrage des extracteurs, la validation des résultats, pour enfin décider d'utiliser de celles-ci ou d'itérer à nouveau. La stratégie de sélection d'index que nous proposons intègre donc cette connaissance, elle privilégie l'interactivité de l'administrateur avec le système d'extraction de motifs pour guider le processus de sélection à travers un ensemble de contraintes. Ces contraintes sont utilisées pendant le processus de sélection pour ne générer qu'un ensemble d'index *apriori* pertinents (qui respectent les exigences de l'administrateur) en une seule étape et éviter ainsi que ce processus soit réalisé en deux étapes : La génération d'un grand nombre d'index candidats, suivie d'une deuxième phase d'élagage imposant ces contraintes à posteriori.

Organisation du mémoire

Le mémoire est organisé en quatre chapitres :

- Le *premier chapitre* introduit le concept des entrepôts de données (définition, spécificités par rapport aux bases de données, architecture et différentes implémentations,...) ainsi que quelques problèmes liés à la conception physique d'un tel système. Il aborde par la suite la présentation des techniques d'optimisation des requêtes les plus utilisées dans ce contexte et met l'accent sur l'utilité des différentes techniques d'indexation ainsi que leurs utilités dans la réduction du temps de réponse des requêtes.
- Le *deuxième chapitre*, présente le problème de sélection d'index dans le contexte des entrepôts de données (définition, formalisation du problème, complexité, ...etc). Il introduit ensuite un état de l'art présentant la démarche générale

de sélection d'index dans la littérature ainsi que les principaux travaux qui ont abordés ce problème. Enfin, il fait une analyse de ces travaux pour identifier leurs principaux verrous. Ces derniers vont constituer nos motivations pour proposer une nouvelle approche de sélection d'index.

- Le *troisième chapitre*, sera consacré à la présentation de notre approche. Nous commençons par exposer le paradigme de l'extraction de motifs sous contraintes, nous discutons les raisons pour lesquelles nous avons choisi ce paradigme, puis nous introduisons l'outil MUSIC-DFS permettant ce type d'extraction. Après avoir défini tous les outils nécessaires pour exposer notre approche, le reste du chapitre sera dédié aux détails de cette approche (principe, différentes étapes, types de contraintes, modèle d'évaluation,..)
- Le *quatrième chapitre* présente l'outil *CIStool (Constrained Index Selection tool)* que nous avons développé afin d'assister l'administrateur dans ses tâches d'optimisation de la conception physique et de tuning. Les principales fonctionnalités de l'outil sont présentées ; puis, il est utilisé pour réaliser une série d'expérimentations afin d'évaluer notre approche et de la comparer avec d'autres approches de sélection d'index.
- Enfin, une *conclusion* rappelle la problématique étudiée, les résultats obtenus et esquisse diverses perspectives de prolongement de nos travaux.
- Deux *annexes* sont rajoutées en fin du mémoire. La première donne plus de détails sur le fonctionnement de l'outil MUSIC-DFS. Tandis que la deuxième, présente la charge de requêtes utilisée dans la partie expérimentale.

Chapitre I

Techniques d'optimisation dans les entrepôts de données relationnels

1 Introduction

Aujourd'hui, les bases de données des entreprises contiennent d'extraordinaires quantités de données collectées et stockées dans des bases de données de grande taille. Cette quantité *surabondante* de données entraîne une confusion entre les données brutes et l'information, entre les faits et la connaissance. Les données sont aussi *volatiles* et *non organisées* pour la prise de décision, et souvent *éparpillées* dans de multiples systèmes *hétérogènes*. Il devient donc fondamental de *rassembler* et *d'homogénéiser* les données dans une base unique, spécialement conçue pour l'analyse des indicateurs nécessaires aux *prises de décisions* : un ENTREPÔT DE DONNÉES (DATA WAREHOUSE). L'objet de l'entrepôt de données est de transformer les données déjà présentes dans et à l'extérieur de l'entreprise, en connaissance, afin de permettre la judicieuse décision au bon moment.

Les entrepôts de données ont la particularité d'être volumineuses vu qu'ils possèdent une structure scalable permettant l'ajout continuel de nouvelles données et sources de données. Aussi, le processus d'analyse de données s'appuie souvent sur des requêtes qui regroupent et filtrent des données de différentes formes, et compte tenu de la nature interactive des entrepôts de données, la nécessité d'avoir un *temps de réponse rapide* est un objectif critique pour l'utilisateur et/ou le décideur. Afin de faire face aux complexes et fastidieuses requêtes d'aide à la décision, il y a un besoin de techniques efficaces et sophistiquées de conceptions logiques et physiques et une bonne maitrise des structures physiques (index, vues matérialisées, etc.)

Dans ce chapitre, nous nous attacherons à présenter le concept général d'un entrepôt de donnée, les techniques d'optimisation les plus utilisées dans ce contexte,

les problèmes liés à chaque technique et l'utilité de chacune d'elle dans la réduction du temps de réponse des requêtes décisionnelles. Dans la dernière section nous nous concentrons sur les différentes techniques d'indexation utilisées dans le contexte des entrepôts de donnée.

2 L'entrepôt de donnée (ED)

Le concept d'entrepôt de données a pris forme au début des années 90 [1], il est devenu depuis la clé de voûte de ce que l'on appelle l'informatique décisionnelle.

Un entrepôt de donnée peut être vu comme une base de données spécifique, entièrement dédiées aux décideurs. Ces bases de données thématiques constituent une véritable interface entre les décideurs et les diverses sources de données et permettent un accès simplifié aux données en masquant l'hétérogénéité des sources.

L'entreposage de données est une technologie habilitante pour les applications d'analyse de données, il est aujourd'hui omni présent dans tous les secteurs d'activités (économique, médical, militaire, les services Web, la bio-informatique ou autre). Dans le domaine de la vente par exemple, on aimerait déterminer les produits à succès, les modes, les habitudes d'achat des consommateurs globalement ou par secteur géographique. Dans le domaine des services tels que les banques, les entreprises de télécommunication, les assurances et mutualités, on aimerait pouvoir classifier les clients, détecter des fraudes ou les clients qui risquent d'être infidèles, etc.

Une définition aux entrepôts de données a été proposée par Inmon en 1992 [1] : *«Un entrepôt de données est une collection de données orientées sujet, intégrées, historisées et non volatiles, utilisée comme support de gestion d'un processus d'aide à la décision.»* .

Cette définition dégage les principales caractéristiques des ED que nous allons détailler dans la section suivante.

2.1 Caractéristiques

Les données d'un entrepôt respectent les caractéristiques suivantes :

- *Orientées sujet :* à l'inverse des données des systèmes d'informations d'une entreprise qui sont organisées selon les applications d'une entreprise (orientées applications), Les données d'un Entrepôt de données sont organisées autour des thèmes /sujets qui ont un intérêt majeur pour l'entreprise par exemple : Production, Vente, Marketing,... (figure I.1[14]). Cette orientation par thème va permettre à l'entreprise de réaliser des analyses sur ces différentes activités.

Cette orientation permet également de faire des analyses par itération, sujet après sujet et de développer son système décisionnel progressivement. L'intégration dans une structure unique est indispensable pour éviter aux données concernées par plusieurs sujets d'être dupliquées.

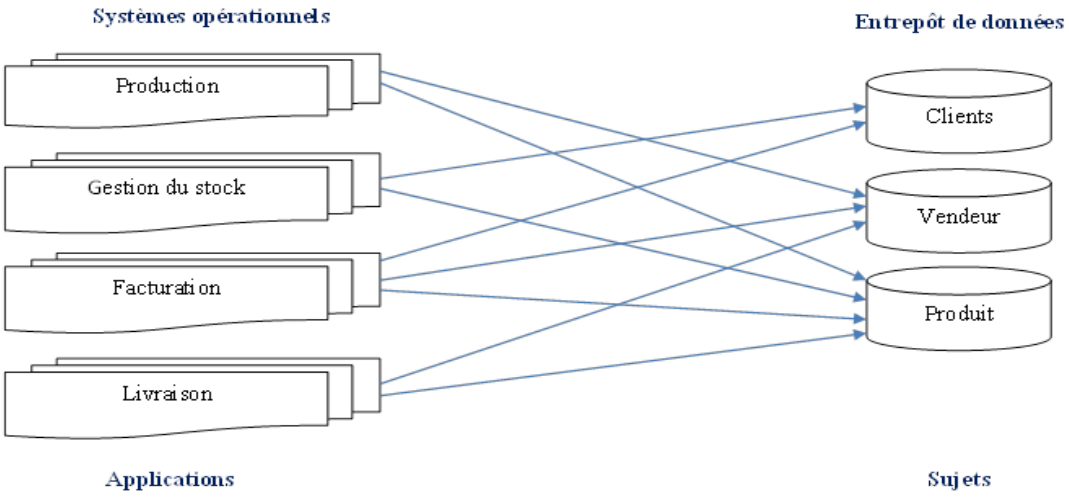

FIGURE I.1: Données orientés sujet dans un entrepôt de données

- *Intégrées* : Les données proviennent de plusieurs sources éventuellement hétérogènes. Cette hétérogénéité donne lieu à des conflits des modèles, des schémas, syntaxiques, sémantique (conflits de représentation, de noms, de contexte et de mesure) entre les données des sources. L'intégration consiste à résoudre ces problèmes et mettre les données en forme et les unifiées afin d'avoir un état cohérent.
- *Historisées* : La prise en compte de l'évolution des données est primordiale pour la prise de décision et notamment les prédictions. Dans un entrepôt de données, la donnée ne doit jamais être mise à jour, pour cela un référentiel temps doit être associé aux données afin de permettre l'identification de valeurs précises dans la durée. Cette caractéristique donne la possibilité de suivre une donnée dans le temps pour analyser ses variations.
- *Non volatiles* : Les données chargées dans l'entrepôt sont surtout utilisées en mode consultation, elles peuvent être interrogées mais ne sont ni modifiées, ni supprimées (sauf dans les cas de rafraichissement de l'entrepôt) et ce afin de conserver la traçabilité des informations et des décisions prises,
- *Support de gestion d'un processus d'aide à la décision :* Les bases de données des systèmes existants de type (*OLTP : "On-Line Transaction Processing"*) n'impliquent que quelques données, occasionnent des changements dans la base de données et requièrent une réponse presque instantanée. Mais elles ne sont

pas appropriées comme support d'analyses, car les données pertinentes pour faire des analyses se trouvent disséminées entre plusieurs bases. Les traitements mis en œuvre pour l'aide à la décision utilisent des processus d'analyse en ligne de données (*OLAP : "On-Line Analytical Processing"*) qui impliquent la lecture de nombreuses données mais n'entrainent pas de changement dans la base de données et ne requièrent pas une réponse instantanée. [2]

2.2 Base de données Vs Entrepôt de données

Les bases de données sont utilisées dans les entreprises pour organiser les importants volumes d'informations contenus dans leurs systèmes opérationnels. Ces données sont gérées selon des processus transactionnels en ligne (OLTP : "On-Line Transactional Processing"). Ils sont destinés à l'insertion, modification et suppression des enregistrements d'une base de données conçue en troisième forme normale. Un entrepôt est conçu pour supporter des opérations d'analyse utiles à la prise de décision. Ces opérations décisionnelles utilisent des processus d'analyse en ligne de données (OLAP : "On-Line Analytical Processing"). Ces processus répondent aux besoins spécifiques des analyses d'information. [3]

Le tableau I.1 résume les différences entre les systèmes de gestion de bases de données et les entrepôts de données [4] :

	Bases de données	*Entrepôts de données*
Raison d'être	Supporter les opérations courantes de l'entreprise	Supporter des décisions gestionnaires
Données	Détaillées	Détaillées et Résumées
	Orientées application	Orientées sujet
	A jours	Historiques
	Dynamiques	Statiques
	Des giga-octets	Plutôt des téra-octets
Utilisateurs / Utilisation	Employés de bureau	Analystes Décideurs
	Nombreux	Peu Nombreux
	Accès Concurrents	Non Concurrents
	Requêtes de Mises à jour	Interrogations
	Requêtes prédéfinies	Requêtes 'One Use'
	Réponses immédiates	Réponses moins rapides

TABLE I.1: Comparaison entre les systèmes OLTP et les entrepôts de données

2.3 Modèle de données pour les entrepôts

Les données à analyser doivent apparaître sous une forme facilitant les prises de décision. Une structuration des données selon plusieurs axes d'analyse (telles que le temps, la localisation géographique, une nomenclature de produits,...) est donc primordiale. La modélisation traditionnelle sous la forme de relations est inadéquate pour supporter efficacement ces analyses (appelées analyses multidimensionnelles). Pour cela, les données ne sont pas modélisées sous forme tabulaire mais sous forme d'hyper-cubes (cubes de données ou encore data cubes). C'est ce qu'on appelle : *la modélisation multidimensionnelle.*

La figure I.2 illustre un exemple de cube de données. Il représente la répartition de ventes suivant plusieurs axes d'analyse : temps, localité ou région et catégorie.

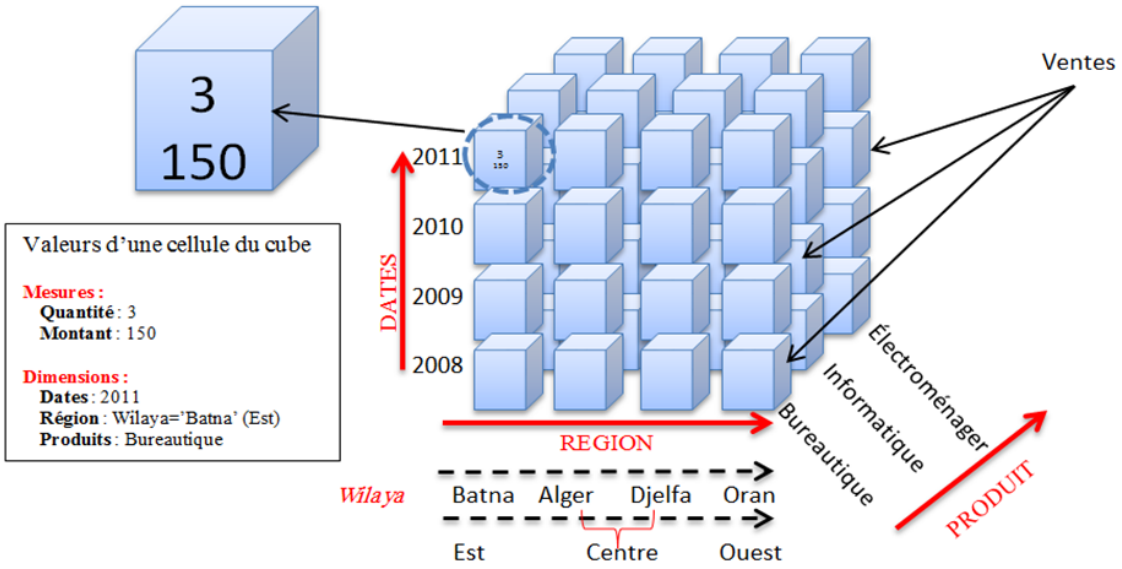

FIGURE I.2: Exemple de cube de données

Cette modélisation multidimensionnelle a donné naissance aux concepts de *fait* et de *dimension.* [6]

FAIT : le fait représente le sujet analysé, il est formé de *mesures* correspondant aux informations de l'activité analysée. Les mesures d'un fait sont numériques et généralement valorisées de manière continue [6]. Le fait que les mesures sont numériques cela permet de résumer un grand nombre d'enregistrements en quelques enregistrements (on peut les additionner, les dénombrer ou bien calculer le minimum, le maximum ou la moyenne). Par exemple, dans le fait « Ventes », nous pouvons avoir la mesure "Quantité de produits vendus par wilaya".

DIMENSION : Une dimension représente un *contexte d'analyse* d'un fait, par exemple la gestion des « Ventes » peut être analysée selon les dimensions : Produit, Région, et Dates. Les dimensions sont composées d'attributs, appelés *paramètres*, agencés de manière hiérarchique, qui modélisent les différents niveaux de détails des axes d'analyse. Par exemple, pour la dimension Dates, nous pouvons avoir la hiérarchie suivante : année, semestre, trimestre, mois, semaine et jour.

2.4 Les implémentations des modèles multidimensionnels

Selon la manière dont le cube est stocké, Il existe trois manières principales pour construire un système basé sur un modèle multidimensionnel : l'approche MOLAP, l'approche ROLAP et une troisième Hybride.

2.4.1 L'approche MOLAP

Les systèmes MOLAP (Multidimensional On-Line Analytical Processing) implémentent le cube sous forme d'un tableau multidimensionnel en utilisant un SGBD multidimensionnel spécialisé. Chaque dimension de ce tableau est associée à une dimension du cube.

Ces systèmes sont généralement plus performants puisqu'ils demandent un précalcul de toutes les agrégations possibles ce qui permet d'offrir un temps de réponse quasi-immédiat à l'utilisateur. En plus, les données, à l'intérieur de la base de données multidimensionnelle, sont organisées d'une manière similaire à la manière dont elles seront utilisées lors de l'analyse, c'est-à-dire sous forme de matrices et donc un accès direct aux données. Par contre, cette structure est valable pour une situation peu évolutive et ne dépassant pas les quelques giga-octets.

2.4.2 L'approche ROLAP

Les systèmes ROLAP (Relational On-Line Analytical Processing) stockent les données du cube en utilisant un SGBD relationnel. Le modèle multidimensionnel est alors traduit de la manière suivante :

- Chaque fait correspond à une table, appelé *table de fait*,
- Chaque dimension correspond à une table, appelée *table de dimension*.

Ainsi, la table de fait est constituée des attributs représentant les mesures d'activités (par exemple les quantités vendues) et les attributs clés étrangers de chacune des tables de dimension. Les tables de dimension contiennent les paramètres et une clé primaire permettant de réaliser des jointures avec la table de fait [7].

L'avantage de ce type de systèmes est qu'ils permettent l'exploitation des capacités d'un standard bien établi et maîtrisé « le relationnel », ce qui réduit le coût de mise en œuvre et facilite leur intégration dans les SGBD relationnels existants. Ils permettent aussi le stockage de grandes quantités de données. Néanmoins, vu la taille des tables de Faits, ils peuvent présenter un temps de réponse élevé aux requêtes [8].

2.4.3 L'approche HOLAP

Dans cette approche, un serveur HOLAP (Hybrid On-Line Analytical Processing) accède à deux bases de données différentes, la première maintient par un SGBD multidimensionnel MOLAP les données agrégées (fréquemment utilisées) et la deuxième, les données détaillées de base dans un SGBD relationnel ROLAP. Cette approche est particulièrement utile lorsque la majeure partie des requêtes touchent des données agrégées et que les données détaillées représentent un volume important [8].

Dans ce mémoire, nous nous intéressons aux entrepôts de données modélisé selon un modèle multidimensionnel ROLAP, nous décrivons dans ce qui suit la structure de données multidimensionnelles la plus utilisée dans ce modèle.

2.5 Schéma relationnel en étoile

Pour rendre les SGBDs relationnelles plus utiles pour les applications OLAP, de nouvelles fonctions leurs sont rajoutés. Ces caractéristiques, dites super-relationnelles permettent de fournir des temps d'accès rapides aux applications OLAP, les données sont organisées selon un schéma en étoile (star). Ce modèle représente visuellement une étoile, les mesures d'intérêt pour l'OLAP sont stockées dans la table des faits (e.g., ventes, stock). Pour chaque dimension du modèle multidimensionnel il existe une table (e.g., région, produit, temps). Cette table stocke les informations relatives aux dimensions. La figure I.3 illustre ce modèle [9].

Les requêtes généralement appliquées sur ce schéma sont appelées « *requêtes de jointure en étoile* », et ont les propriétés suivantes [10] :

- il y a des jointures multiples entre la table des faits et les tables de dimension.
- il n'y a pas de jointure entre les tables de dimensions.
- plusieurs prédicats de sélection peuvent être appliqués sur les attributs descriptifs de chaque table de dimension impliquée dans une opération de jointure.

Ce schéma est le plus populaire puisqu'il :

- Offre une performance dans la rapidité d'analyse des données en réduisant au maximum le nombre de tables et le nombre de joints entre les tables.
- Il offre aussi un maximum de flexibilité des dimensions puisque chaque dimension est représentée par une seule table.
- Les modifications des dimensions deviennent plus simples. Sa simplicité peut lui permettre de faire des analyses complexes sans grande difficulté.

Par contre, il y a de la redondance dans les tables de dimensions à cause de la non-normalisation des données [6].

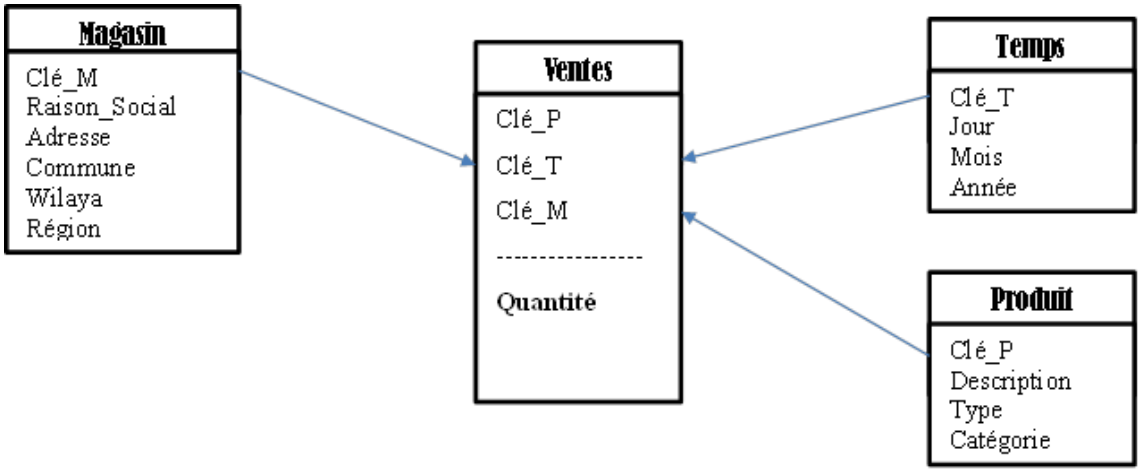

FIGURE I.3: Exemple de modélisation en étoile

Variantes du schéma en étoile

Il existe deux variantes de ce modèle, le schéma en flocon de neige et le schéma en constellation de faits :

1. Dans le schéma en *flocons de neige (snowflake schema)* nous éclatons les tables de dimension en sous-tables selon la hiérarchie de cette dimension. Ce qui peut être vu comme une normalisation des tables de dimensions. Ce schéma évite les redondances d'information ce qui permet de gagner de l'espace disque et facilite l'alimentation. Mais peut altérer les performances de l'entrepôt car il nécessite des jointures lors des agrégats sur les dimensions.

2. Le schéma en *constellation de faits* consiste à fusionner plusieurs modèles en étoile qui utilisent des dimensions Communes [3]. Un modèle en constellation comprend donc plusieurs faits et des dimensions communes ou non.

3 Techniques d'optimisation des entrepôts de données

Le processus d'analyse est réalisé à l'aide de requêtes complexes comportant de multiples jointures entre la table de faits et les tables de dimension adjacentes et des opérations d'agrégation sur des tables très volumineuses. Un temps de réponse qui peut durer des heures voir des jours n'est pas acceptable vu la nature interactive des entrepôts de données et la nécessité d'un temps de réponse acceptable pour prendre des décisions à temps. Donc, il y a un besoin urgent de techniques d'optimisation efficaces et sophistiqués lors de la conception physique. [12] Sans technique d'optimisation de requêtes, l'interrogation d'un entrepôt de données serait complexe.

Les travaux concernant l'optimisation des performances des requêtes décisionnelles classent les structures d'optimisation en deux catégories [7] [11] [13] :

1. *Les techniques non redondantes :* ne dupliquent pas les données mais permettent de réorganiser leur représentation physique [14] : *la fragmentation horizontale et le traitement parallèle.*
2. *Les techniques redondantes :* dont l'utilisation produit une duplication des données : *les index, les vues matérialisées et la fragmentation verticale.*

La figure I.4 illustre une classification des principales techniques d'optimisation.

3.1 Les techniques d'optimisation non-redondantes

Dans les techniques d'optimisation non redondantes il n y a pas de duplication de données et du fait elles ne nécessitent ni coût de stockage ni coût de maintenance.

3.1.1 La fragmentation horizontale

Dans le contexte des entrepôts de données relationnels, la fragmentation horizontale FH consiste à segmenter les tables, les index et les vues matérialisées en plusieurs ensembles de fragments (partitions) de même structure que la table initiale. Chaque fragment est défini par un ensemble de prédicats de sélection, physiquement stockés et consultés séparément [15].

L'objectif de la fragmentation des tables de l'entrepôt de données est d'isoler les parties de l'entrepôt ayant des probabilités d'accès différentes. En effet, le fait de regrouper les données correspondant à un ensemble de requêtes similaires permet de limiter les temps de recherche et d'améliorer la gestion des mémoires cache en fonction de la fréquence d'accès.

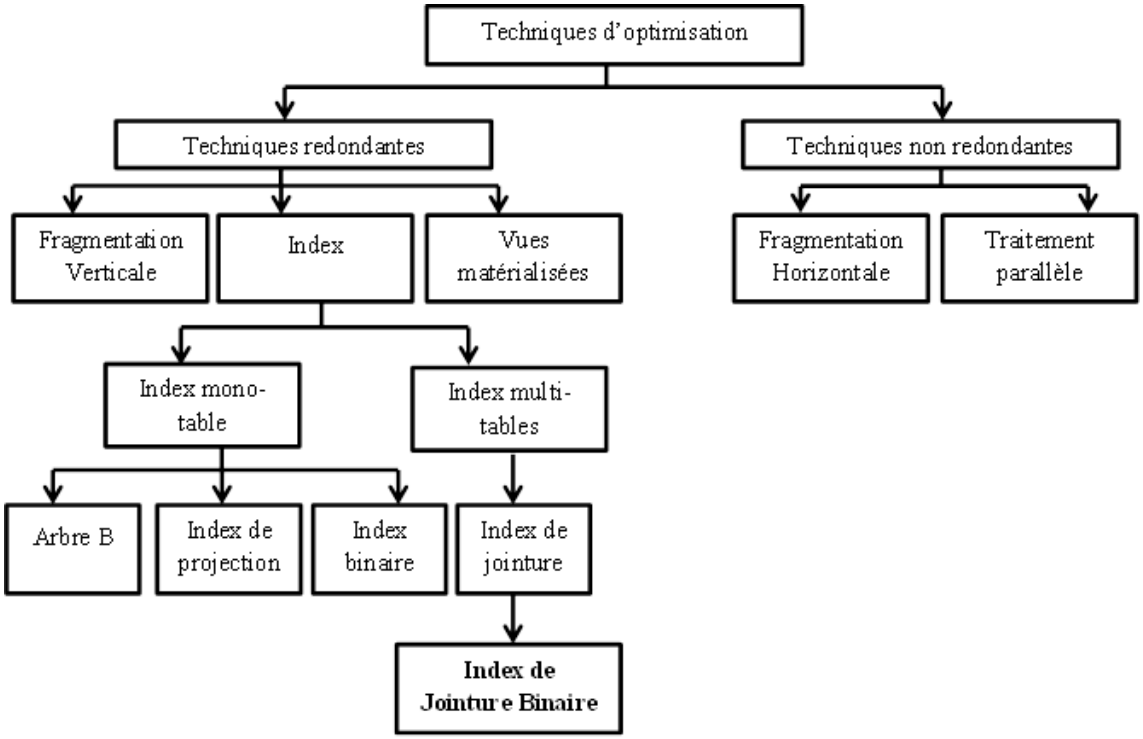

FIGURE I.4: Classification des techniques d'optimisation

Types de fragmentation horizontale

Deux principaux types de fragmentation horizontale existent : la FH primaire et la FH dérivée.

1. *Fragmentation horizontale primaire FHP :* (appelée aussi FH mono-table [28]), Dans la ce type de fragmentation une table est partitionnée en utilisant la conjonction de prédicats de sélection définis sur ses propres attributs. La FHP peut être utilisé pour optimiser les sélections, en particulier lors que les clés de partitionnement correspondent avec les attributs de sélection. Dans le contexte d'entrepôt de données, la FHP est bien adapté pour fragmenter les tables de dimension.
2. *Fragmentation horizontale dérivée FHD :* (appelée aussi FH table-dépendent [28]), La fragmentation horizontale dérivée s'effectue avec des prédicats de sélection définis sur une autre table (i.e. suivant la fragmentation horizontale primaire d'une une autre table). Par exemple, une table de faits peut être fragmentée en se basant sur les schémas de fragmentation des tables de dimension. Cette fragmentation est possible s'il existe une clé étrangère qui pointe de la première table à la seconde (relation père-fils). [16][14] La FHD optimise les sélections et les jointures en même temps. Par conséquent, elle est bien adaptée pour les requêtes de jointure en étoile. [14][19][29]

3.1.2 Le traitement parallèle

Le parallélisme est l'une des solutions pertinentes pour faire face aux grandes masses de données gérées par l'entrepôt et les requêtes décisionnelles complexes. Plutôt que de compter sur un seul processeur régulier, les systèmes parallèles exploitent des microprocesseurs rapides et peu coûteux pour optimiser l'enchaînement d'un ensemble d'opérations éventuellement interdépendantes dans le cadre d'une même requête.

Il existe trois architectures largement utilisés pour le parallélisme du traitement : (a) *mémoire partagée*, (b) *disque partagé* et (c) *ne rien partager*. La différence se situe essentiellement au niveau du partage de la mémoire. Pour l'une ou l'autre des architectures, on peut paralléliser des requêtes grâce aux CPU mais aussi grâce aux disques.

Une fois l'architecture choisie, une fragmentation (horizontale primaire, horizontale dérivé ou verticale) est nécessaire pour pouvoir partager les données sur les différents processeurs. Puis, un processus d'allocation des données met les fragments générés sur les nœuds de la machine parallèle.

Pour pouvoir être exécuté sur une telle architecture, les requêtes décisionnelles sont réécrites sur les fragments de données, et pendant leur exécution, l'équilibrage de la charge du travail sur les nœuds doit être vérifié [53].

3.2 Les techniques d'optimisation redondantes

Les structures redondantes comme les index de jointure, les vues matérialisées et la fragmentation verticale causent une redondance des données qui sont présentes à la fois dans les tables et dans les structures d'optimisation [14]. Ils sont plus efficace pour accélérer les opérations de jointure entre plusieurs tables, par contre leurs inconvénients est qu'ils nécessitent un espace de stockage supplémentaire pour leur implémentation et il y a un surcout de maintenance [12].

3.2.1 La fragmentation verticale

La fragmentation verticale est une technique d'optimisation redondante qui consiste à dissocier les différentes attributs ou colonnes d'une relation et à les enregistrer dans des espaces physiques séparés. Pour cela, des projections dont appliquées sur la relation afin de la diviser en plusieurs sous-relations contenant chacune un sous-ensemble des attributs (colonnes) non-clé ainsi que la clé primaire de la relation initiale qui permet la reconstitution de la relation initiale par opération de jointure.

Il y a un certain nombre de raisons pour effectuer une fragmentation verticale, par exemple :

- La fragmentation verticale accélère les opérations de projection et ce, en ne chargeant que les sous relation dont les attributs figurent dans la requête. Sachant qu'une requête n'a pratiquement jamais besoin de tous les attributs des tables auxquelles elle s'adresse. Elle est particulièrement intéressante si on l'applique aux tables de faits.
- Une ligne avec moins d'attributs prend moins d'espace et du fait, les mises à jour et l'exécution des requêtes sont plus performantes car le tampon système peut contenir plus de lignes à la fois et donc le nombre d E/S diminue.
- La décomposition d'une relation verticalement permet également un certain nombre de transactions d'être exécutées simultanément. Par exemple, dans une même entreprise, la direction des ressources humaines peut conserver les informations telles que le nom, l'adresse et le grade avec le numéro de client comme clé. Tandis que le service du paie maintient le numéro du compte bancaire client et les informations de crédit, avec comme clé le même numéro de client.

L'inconvénient principal de La fragmentation verticale est qu'elle requiert des jointures supplémentaires lorsqu'une requête accède à plusieurs fragments [16].

3.2.2 Les vues matérialisées

Une vue matérialisée ("MaterializedView" ou "VM") est une requête nommée, stockée sous forme d'une table contenant les résultats d'exécution de cette requête. Dans les entrepôts de données, les vues matérialisées sont utilisées pour pré-calculer et stocker des données agrégées telles que par exemple la somme des ventes. Ils peuvent également être utilisées pour pré-calculer des opérations coûteuses en temps d'exécution telles que les jointures avec ou sans agrégations. Ceci élimine le surcout associé à ces opérations pour les requêtes les plus fréquentes qui nécessitent seulement un accès aux vues matérialisées au lieu des données sources dont la taille est significativement plus importante que celle des vues matérialisées.

Cependant, la manipulation des vues matérialisées présente deux problématiques importantes : la *sélection des vues matérialisées* et la *maintenance des vues matérialisées* [11][14][16][17]

3.2.3 Les index

Un index est une structure de données utilisée et entretenue par le système de gestion de base de données (SGBD) pour lui permettre de rechercher rapidement les

informations nécessaires à une requête sans parcourir toutes les données. Il permet à partir d'une clé d'index de trouver l'emplacement physique des n-uplets recherchés.

Un index de base de données fonctionne comme un index dans le dos d'un livre. Un index d'un livre associe les informations d'intérêt avec un numéro de page. Pour localiser des informations dans un livre, il est généralement beaucoup plus rapide d'inspecter l'index en premier, de déterminer le numéro de la page, puis tournez vers la page souhaitée. S'il n'y avait pas d'index, on aura à inspecter chaque page du livre pour trouver les informations.

Pour un entrepôt de données l'utilisation d'un index simplifie et accélère les opérations de recherche, de tri, de jointure ou d'agrégation effectuées par le SGBD. Par exemple, l'utilisation des index peut optimiser les requêtes qui requière d'ordonner les tuples selon un ou plusieurs attributs comme les clauses GROUP BY et ORDER BY. Si l'attribut de jointure entre deux ou plusieurs tables est indexé, il suffit juste de calculer la jointure en utilisant les index sans pour autant accéder aux tables ce qui permet de réduire considérablement le nombre d'opérations d'entrée sortie.

De même que pour les vues matérialisées, au niveau de la conception physique, l'administrateur de l'entrepôt de données doit sélectionner des index afin d'accélérer l'exécution des requêtes. La sélection d'index est difficile car leur nombre est exponentiel en nombre total d'attributs dans l'entrepôt. Ce problème est connu sous le nom de problème de sélection d'index (PSI). Dans ce mémoire nous nous intéressons à ce problème et il sera traité en détails dans le prochain chapitre.

Dans la section suivante, nous décrivons quelques techniques d'indexation qui ont été étudiées/utilisées pour des fins académique/industrielles dans le contexte de bases de données ou entrepôts de données et nous montrons l'intérêt de chaque techniques d'indexation pour les entrepôts de données.

4 Techniques d'indexation

Il existe de nombreuses techniques d'indexation, chaque technique est appropriée pour une situation particulière. Nous allons utiliser l'exemple de la figure I.5 pour expliquer ces techniques d'indexation tout au long de cette section. La figure illustre l'exemple d'un schéma en étoile avec une table de faits centrale appelée VENTE et deux tables de dimension appelés PRODUITS et CLIENTS.

4.1 Index B-arbre

L'index B-arbre est l'index par défaut pour la plupart des SGBD relationnelles. Le niveau le plus haut de l'index est appelé : racine. Le niveau le plus bas est

Produit TABLE

Produit ID	Poids	Taille	Package-Type
P10	10	10	A
P11	50	10	B
P12	50	10	A
P13	50	10	C
P14	30	10	A
P15	50	10	B
P16	50	10	D
P17	5	10	H
P18	50	10	I
P19	50	10	E
P20	40	10	I
P21	50	10	F
P22	50	10	G
P23	40	10	J
P24	50	10	G
P25	10	10	F

Client TABLE

Client_ ID	Sexe	Ville	Région
C101	F	Alger	Centre
C102	F	Alger	Centre
C103	M	Djelfa	Centre
C104	M	Alger	Centre
C105	F	Batna	Est
C106	F	Djelfa	Centre
C107	M	Alger	Centre
C108	F	Annaba	Est
C109	M	Alger	Centre
C110	F	Oran	Ouest

Vente TABLE

Produit ID	Client_ ID	Total_ventes
P10	C105	100
P11	C102	100
P15	C105	500
P10	C107	10
P10	C106	100
P10	C101	900
P11	C105	100
P10	C109	20
P11	C109	100
P10	C102	400
P13	C105	100

FIGURE I.5: Exemple d'un schéma en étoile avec une table de faits VENTE et deux tables de dimension PRODUITS et CLIENTS

appelé : feuille. Tous les autres niveaux entre les deux sont appelés branches. La racine et les branches contiennent des entrées qui pointent vers le prochain niveau de l'indice. Les nœuds feuilles composées de la clé d'index et des pointeurs pointant vers l'emplacement physique (Row-ID) dans lequel les enregistrements correspondants sont stockés. Un index B-arbre pour package_type de la table PRODUIT est montré dans la figure I.6.

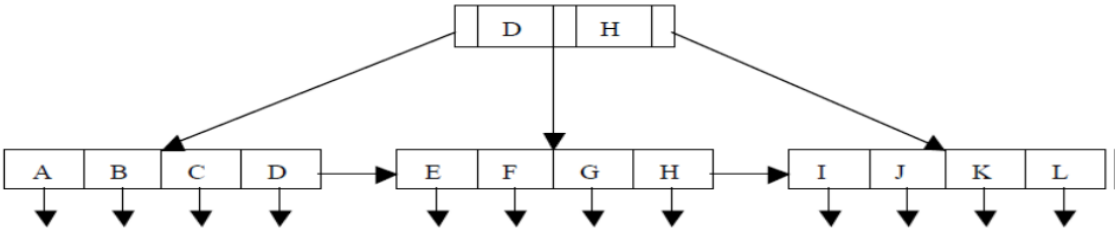

FIGURE I.6: Index B-arbre pour package_type de la table PRODUIT

L'index B-Arbre est apprécié dans les applications d'entrepôt de données pour la colonne cardinalité élevée tels que les noms car l'utilisation de l'espace de l'index est indépendant de la cardinalité de la colonne. Cependant, l'index B-Arbre possède des caractéristiques qui le rendent un mauvais choix pour les requêtes des ED. Tout d'abord, un index B-Arbre n'est d'aucune utilité pour les données de cardinalité faible, comme la colonne sexe, car il réduit de très peu le nombre d'E/S et peut utiliser plus d'espace que la colonne indexée. Deuxièmement, chaque index B-arbre est indépendant et ne peut donc pas fonctionner avec l'autre au niveau des index avant d'aller à la source de donnée. Enfin, l'index B-Arbre récupère les données du résultat ordonnées par les valeurs clés qui ont des Row-IDs non ordonnées et donc, plus d'opérations d'E/S et des défauts de page sont générées.

4.2 Index de projection

Un index de projection sur une colonne indexée A dans un tableau T stocke toutes les valeurs de A dans le même ordre qu'ils apparaissent dans T. La figure I.7-a montre l'index de projection sur Package_Type de la table PRODUIT.

Généralement, les requêtes sur un entrepôt de données récupèrent un sous-ensemble de données des colonnes de la table ; le fait d'avoir un index de projection sur ces colonnes réduit énormément le coût d'interrogation, car une seule opération E/S peut apporter plus de valeurs dans la mémoire.

4.3 Index bitmap

Les index bitmap sont l'une des méthodes d'indexation les plus efficaces disponibles pour accélérer les requêtes multidimensionnelles. Les requêtes sont évaluées de manière performante par des opérations logiques (par exemple les opérations ET, OU, XOR, NOT) qui sont bien pris en charge par le matériel informatique. Pour un attribut avec c valeurs distinctes, l'index bitmap génère c bitmaps avec N bits chacun, où N est le nombre d'enregistrements (lignes) dans l'ensemble de données. Chaque bit dans un bitmap est mis à "1" si l'attribut dans l'enregistrement est d'une valeur spécifique, sinon le bit est mis à "0". La figure I.7-b montre un exemple de l'index Bitmap sur la colonne Package_Type de la table PRODUIT.

Package_Type
A
B
L
C
.
.
G

(a) Projection

B_A	B_B	B_A	B_C	B_D	B_E	B_F	B_G	B_H	B_I	B_J	B_K	B_L
1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	0
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
0	0	0	0	0	0	0	1	0	0	0	0	0

(b) Index Bitmap

FIGURE I.7: Un exemple d'un index de projection et un index bitmap pour Package_type de la table PRODUIT

Pour répondre à une requête, les vecteurs de bitmap des valeurs spécifiées dans le prédicat de condition sont lus dans la mémoire. S'il y a plus d'un vecteur bitmap lus, une opération booléenne est effectuée sur eux avant d'accéder aux données. Cependant, le problème se produit si l'index bitmap est construit sur une colonne de cardinalité élevée ce qui nécessite alors plus d'espace et de temps de traitement des requêtes. Pour réduire la taille d'un index binaire, un ensemble de techniques de

compression a été proposé [25]. Une variante de l'index binaire a été aussi proposée pour réduire sa taille, *l'index binaire encodé* [26]. Dans ce type d'index, chaque valeur de l'attribut indexé A est encodée en utilisant un nombre de bits. Une table de correspondance permet d'établir la correspondance entre la valeur de A et sa représentation encodée.

4.4 Index de jointure

Un index de jointure est une structure qui contient des couples de Row-IDs (identifiant de n-uplet) de tuples de deux relations dont on veut faciliter la jointure. On enregistre dans un index de jointure les couples de Row-IDs des tuples qui vérifient un critère de jointure donné. Cet index peut être vu comme une jointure pré-calculée [16].

Pour les systèmes OLTP qui possèdent souvent des jointures entre deux tables, ce type d'index est souhaitable. Mais, ces index sont limités pour les entrepôts de données modélisés par un schéma en étoile car ses requêtes sont caractérisées par plusieurs jointures entre la table des faits et plusieurs tables de dimension [16]. Pour résoudre ce problème, les auteurs dans [27] ont proposé un nouvel index appelé index de jointure en étoile (star join index), adapté à ce type de schéma. Un index de jointure en étoile contient toutes les combinaisons possibles entre l'identifiant de la table des faits et les clés étrangères des tables de dimension.

L'index est implémenté en utilisant l'une des deux représentations : *Row-id* ou *bitmap*, en fonction de la cardinalité de la colonne indexée. Une représentation *Row-id* est utilisée pour les données avec une cardinalité élevée. Tandis qu'une représentation *bitmap*, qui est appelé *index de jointure binaire (IJB)*, est utilisée avec les données de faible cardinalité.

4.5 Index de jointure binaire

Un Index de jointure binaire est un index bitmap décrit via une requête de jointure. Il est défini sur une table de base et stocke les identificateurs de ligne de la table de base avec les colonnes indexées pour les tables jointes.

Par exemple, l'index de jointure binaire sur la colonne « sexe » peut être construit en utilisant la colonne sexe dans la table CLIENT et la clé étrangère client-ID de la table VENTE. Notez que la table de VENTE ne contient pas la colonne sexe. L'index de jointure binaire sur la colonne sexe est créé en affectant un 1 au bit correspondant à un client-ID dont le sexe est «M» dans la table VENTE. Sinon, le bit est mis à 0 comme le montre la figure I.8.

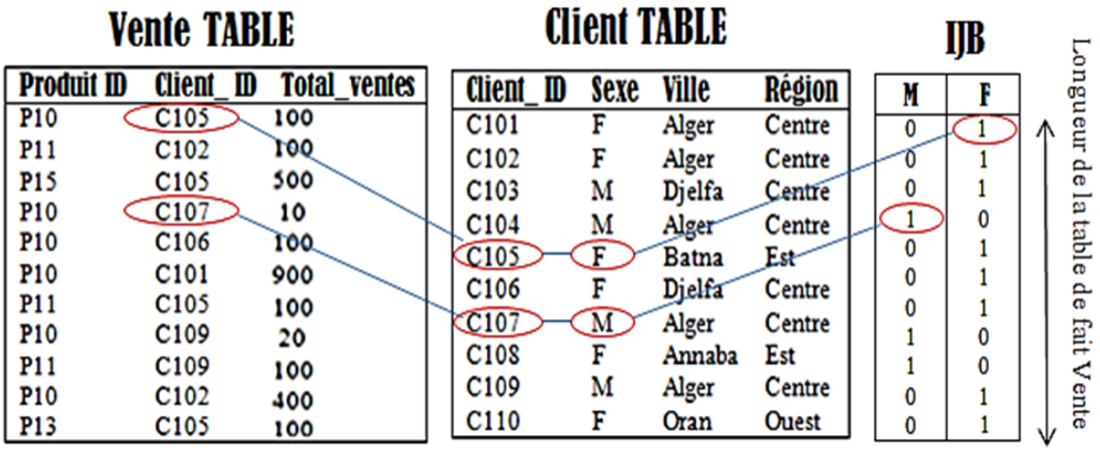

FIGURE I.8: Exemple d'un Index de Jointure Binaire sur la colonne Genre de la table CLIENT

Ainsi, l'index bitmap de jointure est construit en faisant une jointure grâce aux clés étrangères et peut porter sur n'importe quel champ. Les index bitmap sont par ailleurs une structure intéressante pour optimiser les recherches sur les données.

Les index de jointure binaire permettant d'optimiser à la fois les jointures en étoile et les opérations de sélections définies sur les tables de dimensions.

Exemple : Pour illustrer l'utilisation et l'intérêt des index de jointure binaire, considérons la requête Q suivante que l'administrateur souhaite optimiser : [16]

SELECT Count(*)
FROM CUSTOMERS C, SALES S
WHERE C.Gender ='M' AND C.CID=S.CID

Pour ce faire, l'administrateur crée un index de jointure binaire entre la table de faits VENTES et la table de dimension CLIENTS sur la base de l'attribut sexe *comme suit :*

CREATE BITMAP INDEX SC_IDX
ON SALES(CUSTOMER.Gender)
FROM SALES S, CUSTOMERS C
WHERE S.CID= C.CID

Pour exécuter la requête ci-dessus, l'optimiseur de requêtes accède seulement aux bitmap dont le bit correspondant au sexe masculin = '1' et compte le nombre d'occurrences, sans accéder aux tables VENTES et CLIENTS et faire leur jointure.

4.6 Résumé des techniques d'indexation existantes

Le tableau I.2 résume les principales caractéristiques des techniques d'indexation.

Tech. d'index.	*Caractéristiques*	*Avantages*	*Inconvénients*
Index B-Arbre	index stocké sous forme d'arbre (tree) ordonné et équilibré (feuilles au même niveau). La recherche consiste à déterminer l'intervalle pour connaître le niveau inférieur et ainsi de suite jusqu'à atteindre la feuille qui stockera, près de sa valeur, les Row-ID des tuples recherchés	• Il accélère les requêtes connues. • Il est bien adapté pour les cardinalités élevées. • L'espace requis est indépendant de la cardinalité de la colonne indexée. • Il est relativement peu coûteux quand on met à jour la colonne indexée.	• inefficace avec les données de cardinalité faible • Les index ne peuvent être combinés avant de récupérer les données.
Index de Projection	L'index est créé en enregistrant les valeurs réelles de la colonne (s) de tableau indexé.	• Il accélère les performances lorsque quelques colonnes de la table sont récupérées.	• utilisé uniquement pour récupérer des données brutes (c.-à-d liste de colonne en sélection)
Index Binaire Pure	Un tableau de bits est utilisée pour représenter chaque valeur de colonne unique de chaque ligne d'une table, définissant les bits correspondant à la ligne soit ON (valeur 1) ou OFF (valeur 0)	• bien adapté pour les colonnes de cardinalité faible. • utilise des opérations binaires. • Les index peuvent être combinés avant de récupérer les données brutes. • utilise un faible espace • fonctionne bien avec machine parallèle. • facile à construire. • Il accomplit efficacement les fonctions scalaires (par exemple COUNT). • Il est facile d'ajouter une nouvelle valeur indexée. • Il est adapté pour OLAP.	• inefficace avec les données de cardinalité élevée. • très coûteux lorsqu'on met à jour la colonne d'index.
Index de Jointure Binaire	index binaire décrit via une requête de jointure. construit par la restriction d'une colonne sur la table de dimension dans la table de faits.	• Avantages des index Binaires pures • prend en charge les requêtes en étoiles	• Inconvénients des index Binaires pures • Nombre d'index possibles.

TABLE I.2: Principales caractéristiques des techniques d'indexation

Les techniques d'indexation classiques (les B-arbres, les index binaires, les index de projection, etc.) ont montré leurs performances dans les bases de données traditionnelles, mais elles ne sont pas adaptées pour le contexte des entrepôts de données relationnels modélisés selon un schéma en étoile. Cela est dû aux requêtes exécutées sur ce type de schéma (dites requêtes de jointure en étoile), et ce, à cause des opérations de jointure très couteuses surtout quand elles sont exécutées sur un large volume de données comme dans le cas des entrepôts. Les index de jointure binaire permettent de réduire la complexité de ces requêtes en pré-calculant les jointures contenues dans celles-ci. Par contre, le choix d'une configuration d'index est une tâche difficile à cause du nombre important d'attributs et de tables qui peuvent être indexé. Ce problème est connu sous le nom de problème de sélection d'index et il est connu NP-Complet.

Pour cela, nous passerons en revue dans le chapitre suivant le problème de sélection d'index dans les entrepôts de données. Puis, nous présentons les principaux travaux de sélection des index. Pour enfin analyser ces travaux et introduire notre technique de sélection d'index.

Chapitre II

Problème de sélection des index de jointure binaires dans les ED

1 Introduction

L'accès aux données d'un entrepôt est d'autant plus lent que la base de données est volumineuse et les opérations de jointure exigent souvent que ce parcours doit être effectué de façon répétitive [17]. Un parcours séquentiel des données est une opération lente et pénalisante pour l'exécution des requêtes. L'administrateur emploie en général des *index* pour rechercher rapidement les informations nécessaires à une requête sans parcourir toutes les données. La création d'un index permet d'améliorer considérablement le temps d'accès aux données en créant des chemins d'accès directs.

Parmi les techniques d'indexation, les *index de jointure binaire* IJB [35] sont particulièrement adaptées au contexte des entrepôts de données car non seulement ils rendent efficaces les opérations logiques (*And, Or, Not*) et de comptage (*Count*) sur les bitmaps, mais ils permettent aussi de pré-calculer les jointures au moment de la création des index et elles ne sont donc pas calculées lors de l'exécution des requêtes. De plus, l'espace nécessaire au stockage des bitmaps est réduit en raison de leur représentation binaire et la compression éventuelle surtout quand les attributs indexés sont de faible et moyenne cardinalité, ce qui est en général le cas dans les dimensions d'un entrepôt [18].

Sélectionner un ensemble d'index qui permettent de réduire le temps d'exécution des requêtes définies sur l'entrepôt est une tâche difficile à cause du nombre important d'attributs et de tables qui peuvent être indexé. En plus, les index peuvent être définis sur plusieurs attributs issus de plusieurs tables. Le problème de sélection d'index est connu NP-Complet [36] [37] [38] et du fait, il n'existe pas d'algorithmes

qui proposent une solution optimale en un temps fini. Plusieurs travaux de recherche proposent plutôt des solutions proches de la solution optimale en utilisant des algorithmes gloutons, des heuristiques et des techniques de Datamining afin de réduire la complexité du problème.

Dans ce chapitre, nous nous attacherons à présenter le problème de sélection des IJB dans le contexte des entrepôts de données ainsi qu'une synthèse des principaux travaux qui ont traité ce problème. Nous finissons avec une analyse de ces travaux, ce qui va nous permettre d'identifier leurs principaux verrous. Ces derniers vont constituer nos motifs pour élaborer une nouvelle approche de sélection d'IJB dans les entrepôts de données.

2 Sélection automatique des index de jointure binaires

Un Index de Jointure Binaire (IJB) un est un index bitmap décrit via une requête de jointure .C'est une structure d'indexation qui porte sur de multiples tables et qui permet d'augmenter les performances des jointures de ces tables. Dans le contexte des entrepôts de données, un IJB permet de pré-calculer la jointure entre la table des faits et une ou plusieurs tables de dimension ce qui permet d'optimiser à la fois les jointures en étoile et les opérations de sélections définies sur les tables de dimensions. La plupart des SGBD commerciaux supportent les IJB.

Un IJB est défini sur un ou plusieurs attributs non clés des tables de dimensions avec une faible et moyenne cardinalité, appelés attributs indexables. Un attribut est indexable s'il est utilisé dans une requête décisionnelle par un prédicat de sélection. Vu le nombre important d'attributs indexables et vu le nombre de tables de dimension et le nombre d'attributs non clés de chaque table, la sélection d'une configuration d'IJB est généralement une tâche difficile (complexité algorithmique) comparée à d'autres types d'index [11].

2.1 Problème de sélection automatique d'index

Le problème de sélection d'index consiste à sélectionner une configuration d'index minimisant le coût d'exécution d'un ensemble de requêtes décisionnelles (appelé charge de requêtes) supposé représentatif de l'ensemble des requêtes adressées au système. Cette sélection peut être réalisée sous certaines contraintes, comme l'espace de stockage alloué aux index à sélectionner [17].

Le problème est formalisé de la manière suivante [48] :
Étant donné :

1. Un entrepôt de données relationnel modélisé par un schéma en étoile ayant un ensemble de tables de dimension $D = \{D_1,\ldots,D_d\}$ et une table des faits F.
2. Un ensemble représentatif de requêtes décisionnelles $Q = \{Q_1 \ \ldots \ Q_m \ \}$ avec leurs fréquences $Freq = \{ \ Freq_1, \ \ldots, \ Freq_m \ \}$.
3. Un espace de stockage S allouée par l'administrateur aux index à sélectionner.

Alors, il faut trouver une configuration d'index CI tel que :

1. Le coût d'exécution des requêtes en utilisant cette configuration $COUT\ (Q/CI)$ soit minimal.
2. L'espace alloué pour le stockage de ces index ne dépasse pas S : $\sum_{I \in CI} taille(I) \leq S$

Le coût d'exécution de la charge des requêtes Q est exprimé en nombre d'entrées-sorties nécessaires pour l'exécuter.

Il est calculé par la formule :

$$COUT\,(Q) = \sum_{i=1}^{m} COUT\ (Q_i)\ \times Freq_i$$

2.2 Complexité

Soit $A = A_1, A_2, ..., A_n$ un ensemble d'attributs indexables candidats pour la sélection d'une configuration d'IJB. Le nombre d'IJB possibles que nous devons considérer pour sélectionner *un seul IJB* est donné par la formule [11] :

$$IJB_{simple} = \sum_{k=1}^{n} \binom{n}{k} = \ 2^n - 1$$

Pour choisir plus d'un index, le nombre total de configurations possibles de [11] :

$$IJB_{multi} = \ \sum_{k=1}^{2^n-1} \binom{2^n-1}{k} = \ 2^{2^n-1} - 1$$

Ainsi, si le nombre d'attributs indexables est égal à 5 (n = 5), alors le nombre d'IJB possibles est égal à $2^{31} - 1 = 2.147.483.647!!!$

3 État de l'art

Nous abordons dans ce qui suit un état de l'art sur les travaux de sélection d'index. Nous commençons par présenter la démarche générale que la plus part des travaux suivent pour traiter ce problème, nous citons brièvement quelques travaux effectués dans le contexte des bases de données. Puis avec plus de détails, nous présentons quelques travaux étroitement liés à notre contexte.

3.1 Démarche générale de sélection d'index dans les ED

Généralement, les algorithmes proposés pour la sélection d'index comprennent deux étapes : (1) *Sélection de l'ensemble d'index candidats*, (2) *Construction d'une configuration d'index*.

3.1.1 Sélection de l'ensemble d'index candidats

Au cours de cette première étape, un ensemble des index candidats est construit à partir d'une charge de requêtes représentant les requêtes adressées au SGBD durant une période de temps. Cette charge est extraite soit directement à partir du journal des transactions du SGBD ou bien grâce à une application externe.

La sélection des attributs candidats à l'indexation peut se faire de deux façons, *manuellement* ou *automatiquement* :

1. Manuellement par l'administrateur en fonction de son expertise [39], le choix est alors subjectif car il dépend du degré d'expertise de l'administrateur, mais il devient assez difficile à réaliser lorsque le nombre de requêtes dans la charge de requêtes est très grand.
2. L'identification des attributs candidats peut se faire automatiquement en utilisant un analyseur de requêtes, afin d'identifier les attributs intéressants à indexer, qui sont principalement présents dans les clauses WHERE, GROUP BY, ORDER BY, HAVING et UPDATE des requêtes SQL [18] [40] [41].

La plupart des travaux de sélection d'index utilisent la sélection automatique des attributs candidats, l'objectif étant de faciliter les tâches manuelles de l'administrateur du système. [17][18][40][41][42]

Réduction du nombre d'index candidats

La principale difficulté dans la sélection d'index est le grand nombre d'attributs de dimension qui peuvent participer dans le processus de sélection (plusieurs dizaines ou

centaines d'attributs). Il est donc primordial de réduire le nombre d'index candidats afin de réduire l'espace de recherche de l'algorithme qui sélectionne la configuration finale d'index. Pour réduire le nombre d'index candidats, plusieurs méthodes sont utilisées [14] :

- Estimation de l'apport de chaque index dans l'optimisation de la charge et écarter les index qui n'apportent pas un grand bénéfice [45].
- Suppression des index qui causent le dépassement de l'espace total réservé au stockage des index [52].
- Suppression des index ne pouvant pas être crées ou dont la clause de création présente une anomalie [46].
- *Utilisation de techniques de Datamining :* L'idée principale de ces approches [12] [43] [46] [47] [49] est de calculer l'ensemble des groupes fréquents des attributs indexables plutôt que toutes les combinaisons possibles. Les groupes générés dans cette étape sont ensuite utilisées dans la seconde étape d'élagage pour sélectionner la configuration finale d'index.

3.1.2 Construction d'une configuration finale d'index

La configuration d'index candidats obtenue peut ne pas respecter la contrainte de stockage. Aussi, le nombre d'index candidats peut être important et certains SGBD n'autorisent qu'un nombre d'index prédéfini par table [50]. Il faut donc réduire le nombre d'index à générer.

Les méthodes utilisées pour construire une configuration finale d'index à partir des index candidats peuvent être classés en :

1. Méthodes algorithmiques simples (gloutonnes) ascendante ou descendante.
2. Méthodes heuristiques

(a) Les méthodes gloutonnes

Ces méthodes peuvent être classées en deux catégories : méthodes *ascendantes* et méthodes *descendantes*.

1. *Les méthodes gloutonnes ascendantes* démarrent à partir d'un ensemble vide d'index candidats et ajoutent progressivement des index qui minimisent le coût de la charge. Ce processus s'arrête lorsque le coût cesse de diminuer ou si la contrainte de l'espace de stockage est violée.
2. Au contraire, *les méthodes gloutonnes descendantes* considèrent l'ensemble des index candidats comme point de départ. Ensuite, à chaque itération, les index sont élagués jusqu'à ce que le coût de la charge ne diminue plus. Il est à noter

que la plus part des travaux de sélection d'une configuration finale d'index emploient la construction ascendante [12][42][40][41][46]

(b) Les méthodes heuristiques

Le principe de ces approches est de trouver le meilleur sous-ensemble d'index candidats qui fournit le plus de bénéfice pour la charge de requêtes et qui respecte la contrainte d'espace de stockage. Le problème est vu comme un problème d'optimisation où l'on cherche à minimiser une fonction objectif calculée par un modèle de coût qui estime le coût d'exécution de la charge de requêtes si l'ensemble d'index en cours est implémenté.

1. *Méthodes assimilant le problème de sélection au problème d'optimisation du sac à dos :* Le problème de sélection d'index a été assimilé dans certains travaux [40][45][51][52] au problème du sac à dos. Les index sont assimilés à des objets qui peuvent être ou ne pas être mis dans le sac, la taille des index correspond aux poids des objets. L'amélioration dans le temps d'exécution estimé qu'un index contribue à toutes les requêtes qui l'exploitent correspond au bénéfice de l'objet correspondant. L'espace disque alloué au stockage de la configuration d'index finale correspond à la taille du sac à dos.

2. *Méthodes dérivées des algorithmes génétiques :* Le principe de ces méthodes [82] est d'imiter la sélection naturelle afin produire une population (configuration d'index) ayant une meilleure adaptabilité (convergence vers l'optimum).

3.1.3 Modèles de coût

Pour évaluer le coût d'un index et éliminer les index les moins avantageux (ayant un coût plus élevé), deux méthodes sont utilisées pour estimer le coût d'une configuration d'index.

1. La première [12][14][17][43] consiste à mettre en œuvre un modèle de coût mathématique permettant d'estimer le coût d'utilisation d'un index. Les avantages de cette méthode est sa rapidité et qu'elle est complètement indépendante du SGBD, mais la simplicité de ces modèles qui ne reflètent pas exactement la réalité d'exécution des requêtes est son grand inconvénient.

2. La deuxième [40][42] consiste à faire appel à l'optimiseur de requêtes du SGBD pour estimer ce coût en se basant sur les estimations réelles de la tailles des tables, des données chargées, de la taille des tuples etc. il est donc plus fiable

que le premier, mais son principal inconvénient est qu'il rend la technique de sélection dépendante d'un SGBD donné. De plus, il est gourmand en temps de calcul à cause des appels fréquents de l'optimiseur [18].

3.2 Travaux de sélection d'index dans les bases de données

Le problème de sélection d'index a été initialement étudié dans le contexte des bases de données. Les premiers travaux qui ont traité ce problème datent depuis les années 70 et il a été démontré NP-Complet en 1978 [38]. De ce fait, ces travaux ont proposé des solutions permettant de trouver un ensemble d'index proche de l'optimal sans évaluer le coût de tous ces ensembles.

3.2.1 Travaux de Chaudhuri et al. 1997

Dans le cadre du projet AutoAdmin lancé par Microsoft pour l'auto administration des bases de données. Les auteurs dans [45] ont proposé un outil de sélection d'index mono et multi-attributs IST (Index Selection Tool) qui prend en entrée une charge de requêtes et une contrainte d'espace de stockage et il procède en quatre étapes :

1. Dans un premier temps, une configuration d'index mono-attribut est créée à partir de la charge.
2. Puis, un algorithme glouton va utiliser les estimations du cout obtenu par l'optimiseur de requêtes pour en choisir les meilleurs index de cette configuration.
3. Des index sur deux attributs sont construits à partir des index mono-attributs en utilisant la même démarche que l'étape 2. Le processus est réitéré pour créer des index de taille de plus en plus supérieure.
4. La dernière étape construit consiste à faire l'énumération des configurations et choisir les k meilleurs index parmi les n choisis à l'étape précédente à l'aide d'un algorithme glouton et en faisant appel à l'optimiseur.

Les auteurs ont étendu l'optimiseur de requêtes de l'SGBD Microsoft SQL server par un module dénommé "*what-if* " pour simuler la présence des index qui n'existent pas et ont également étendu le module d'optimiseur des coûts, de tel sorte à ce qu'étant donné une requête Q et une configuration C, le coût de Q lorsque le modèle physique est la configuration C, peut être calculé.

L'explosion du nombre d'index généré à l'étape 3 et les appels fréquents de l'optimiseur constituent les principaux inconvénients de cette approche.

3.2.2 Travaux de Valentin et al. 2000

Les auteurs dans [40] proposent une extension de l'optimiseur de requêtes DB2 Advisor d'IBM. L'approche proposé prend en entrée une charge de requêtes et procède en trois étapes :

1. La première étape permet de générer des index dites virtuels qui constituent un ensemble d'index candidats pour chaque requête.
2. L'optimiseur génère ensuite le plan d'exécution optimal de chaque requête et recommande une configuration d'index.
3. Dans la dernière étape, le problème est modélisé comme un problème du sac à dos et résolu par une heuristique qui permet de sélectionner les index dans un ordre décroissant du ratio gain/espace de stockage.

3.2.3 Travaux de Feldman et al. 2003

Les auteurs dans [52] proposent un outil de sélection d'index nommé DINNER qui exploite dans un premier temps une base de connaissances pour constituer son modèle de cout. Puis, il procède à la représentation des requêtes d'une charge sous forme de graphe dans lequel chaque chemin représente un plan d'exécution d'une requête en utilisant ou non des index. Une dernière étape consiste à utiliser des heuristiques pour éliminer et unifier les nœuds inutiles et élaguer les mauvaises solutions.

3.2.4 Travaux de Kratica et al. 2003

Les auteurs dans [82] proposent un algorithme génétique dans lequel le temps de traitement des requêtes est considéré comme fonction objectif. Une population initiale d'individus est construite à partir des index candidats (chaque index est assimilé à un individu). Pour choisir l'ensemble d'individus qui vont survivre à la prochaine génération, une configuration constituée des meilleurs individus (index) est construite par les opérations de croisement, mutation et sélection génétique.

3.3 Travaux de sélection d'index dans les ED.

Nous présentons dans cette section les principaux travaux sur le problème de sélection d'index dans le contexte des entrepôts de données, classifiés en deux catégories :

a) Approches basées sur des *heuristiques*.

b) Approches basées sur des *techniques de datamining*.

A- Approches basées sur des heuristiques

3.3.1 Travaux de Golfareli et al. 2002

Dans leurs travaux, Golfareli et al [41] proposent une approche heuristique pour la sélection des index de type liste de valeurs et les index bitmap dans les entrepôts de données relationnels implémentés en schéma en étoile.

Dans le but de déterminer un ensemble d'index qui minimisent le cout d'exécution de la charge de requête en respectant la contrainte de l'espace de stockage, les auteurs définissent un modèle d'optimiseur qui crée un plan d'exécution pour chaque requête et un modèle de cout pour comparer les différentes solutions. Un ensemble d'index candidats potentiellement utiles est déterminé puis un algorithme glouton choisis progressivement à partir de ceux-ci les index les plus bénéfiques tenant compte de la contrainte de l'espace de stockage.

L'algorithme proposé par Golfarelli et al. procède en trois étapes distinctes.

1. La première consiste à initialiser :
   - *l'ensemble des index candidats C* : pour chaque attribut indexable, l'index le plus bénéfique (selon son cout calculé) est rajouté à C.
   - *l'ensemble des index optimaux O :* Pour chaque table de dimension un index de type liste de valeurs construit sur sa clé primaire est inséré dans O
   - *l'espace de stockage S* nécessaire pour stocker les index à sélectionner.
2. La deuxième étape consiste à sélectionner à partir de C d'une manière gloutonne les index apportant le meilleur bénéfice et les rajouter à O. Si après un ajout, il arrive que tous les attributs composant la clé primaire d'une table de faits soient indexés, alors ces index sont transformés en un seul index multi-attributs construit sur la clé primaire de la table de faits.
3. Dans la troisième étape, des index primaires pour les tables de faits restantes sont choisis.

3.3.2 Travaux de Boukhalfa et al. 2010

Dans [48], les auteurs proposent une approche algorithmique pour la sélection des IJB mono-attribut et multi-attributs basée sur une heuristique d'élimination d'attributs en se basant sur des critères comme la fréquence, la cardinalité des attributs, etc.

Deux algorithmes ont été proposés pour la sélection des IJB, le premier pour la sélection des *IJB mono-attributs* et le deuxième pour les *IJB multi-attributs*. Un modèle de coût basé sur celui proposé dans [12] permettant d'estimer la taille des

IJB ainsi que le coût d'exécution des requêtes en présence d'une configuration d'IJB est utilisé pour évaluer la qualité des configurations d'index obtenues. La figure II.1 représente l'architecture générale de l'approche proposée.

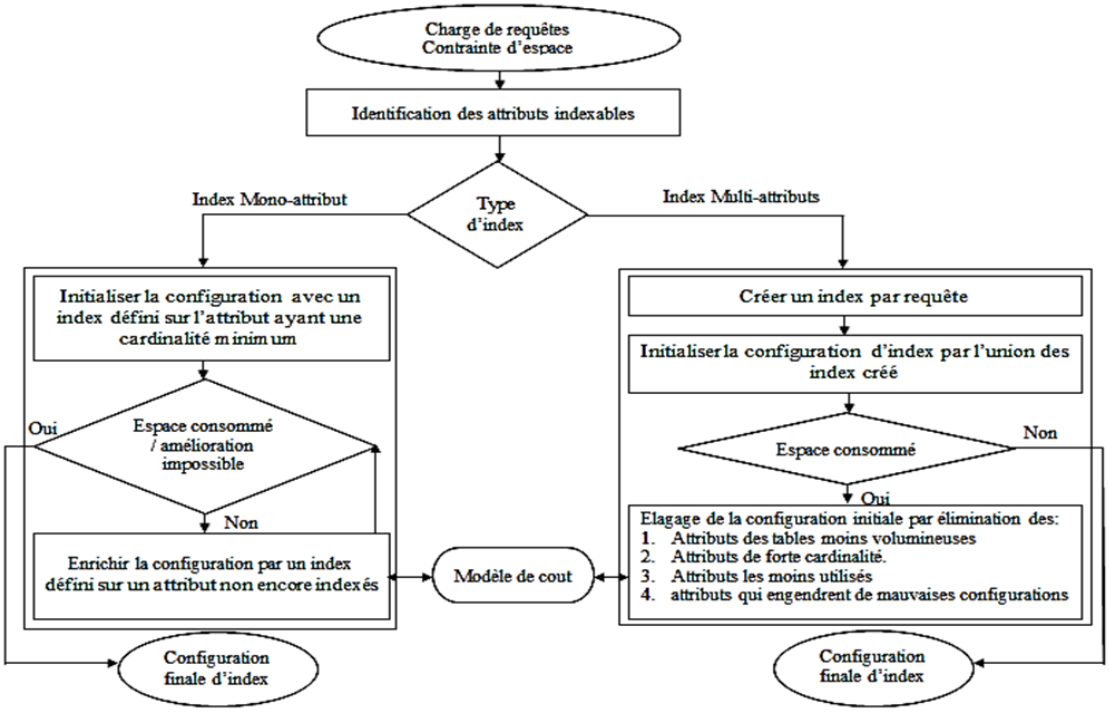

FIGURE II.1: Architecture générale de l'approche de Boukhalfa et al.

Algorithme de sélection d'une configuration d'index mono-attributs

La sélection d'une configuration d'index mono-attributs se déroule en trois étapes :

1. Identification des attributs de faible et de moyenne cardinalité contenus dans les clauses WHERE des requêtes de la charge.
2. Une configuration initiale est construite avec un index mono-attribut défini sur l'attribut ayant une cardinalité minimum.
3. La configuration initiale est enrichie par l'ajout de nouveaux index définis sur d'autres attributs non encore indexés jusqu'à ce qu'aucune amélioration n'est possible et/ou l'espace de stockage est consommé.

Algorithme de sélection d'une configuration d'index multi-attributs

La sélection d'une configuration d'index multi-attributs passe par quatre étapes :

1. *l'identification des attributs indexables :* c'est la même étape que pour la sélection d'une configuration d'index mono-attributs,

2. *la construction d'une configuration par requête :* Pour chaque requête, un index permettant de pré-calculer toutes les jointures de la requête est sélectionné. C'est le scénario idéal permettant de réduire considérablement le coût d'exécution de la requête mais peut violer la contrainte de stockage.
3. *la construction d'une configuration initiale* : Cette configuration initiale est construite en effectuant l'union des index obtenus de l'étape précédente et en supprimant les index partagés par plusieurs requêtes.
4. *la construction d'une configuration finale :* si la configuration initiale respecte la contrainte de stockage, elle est retenue, sinon certains attributs doivent être éliminés. Pour choisir les attributs à éliminer les auteurs ont considéré les quatre stratégies suivantes :

   a) *Elimination des attributs de forte cardinalité :* Dans cette stratégie, l'attribut de forte cardinalité est éliminé de tous les index constituant la configuration initiale jusqu'à ce que la contrainte de stockage soit respectée.

   b) *Elimination des attributs appartenant aux tables moins volumineuses :* les jointures entre la table des faits et les tables de dimension les plus volumineuses sont les plus couteuses et sont donc plus prioritaires. Pour cela, les attributs appartenant aux tables moins volumineuses sont éliminés à commencer avec ceux ayant une forte cardinalité.

   c) *Elimination des attributs les moins utilisés* : cette stratégie suppose qu'il est judicieux de créer des index sur les attributs les plus utilisés afin de satisfaire plus de requêtes. Par conséquent, les attributs les moins utilisés sont éliminés en premier.

   d) *Elimination des attributs apportant moins de réduction de coût :* cette stratégie utilise un modèle de coût pour éliminer les attributs qui engendrent de mauvaises configurations. Pour cela, pour chaque attribut candidat, on l'élimine, on calcule le cout de la charge après élimination pour garder à la fin ceux engendrant une configuration apportant la meilleure réduction de cout.

B- Approches basées sur des techniques de Datamining

3.3.3 Travaux d'Aouiche 2005

Dans [17] l'auteur part de l'hypothèse qu'un index est considéré intéressant, s'il est construit sur un attribut ou un groupe d'attributs les plus fréquemment utilisés

dans l'ensemble des requêtes d'une charge. Pour maitre en évidence cette corrélation, l'auteur utilise une technique de Datamining (recherche des motifs fréquents) pour générer l'ensemble d'attributs (index mono-attribut) ou groupe d'attributs (index multi-attributs) les plus fréquents et donc plus pertinents au lieu de calculer toutes les combinaisons possibles d'index. L'ensemble d'index candidats généré est ensuite utilisé par un algorithme glouton pour la sélection des index finaux grâce à un modèle de cout.

Pour réduire d'avantage le nombre d'index candidats qui peut être énorme (2^k, où k est le nombre d'attributs), l'auteur a fait recours aux motifs fréquents fermés qui représentent l'ensemble maximal d'items communs à un ensemble de motifs et donc ils sont beaucoup moins nombreux. L'algorithme utilisé pour faire l'extraction des motifs fermés est Close [55].

La démarche de sélection automatique d'Index de Jointure Binaire proposé par l'auteur et dont les principales étapes sont représentées dans la figure II.2 , se déroule en cinq étapes :

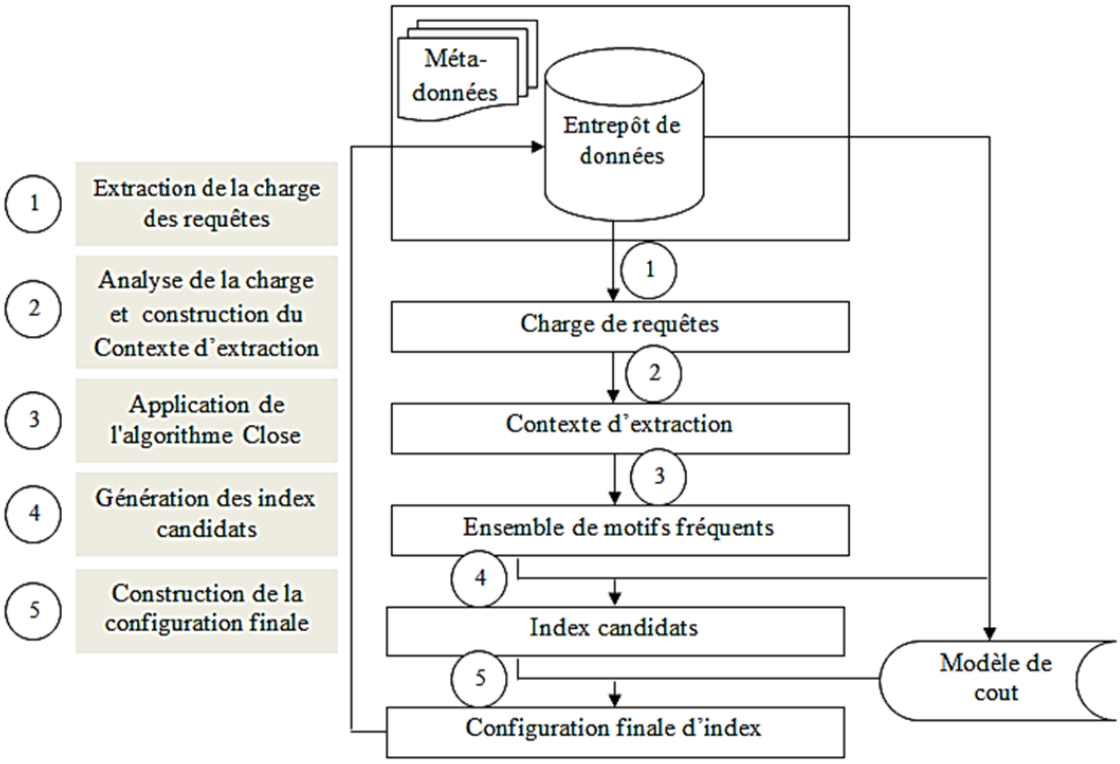

FIGURE II.2: Les principales étapes de l'approche de Aouiche

1. *Extraction de la charge de requêtes :* Durant cette étape, une charge de requêtes représentée par un ensemble de requêtes décisionnelles adressées au SGBD est extraite du journal des transactions.

2. *Analyse de la charge et Construction du contexte de recherche des motifs fréquents* : Durant cette étape, un analyseur syntaxique traite la charge de requête pour en extraire un ensemble d'attributs candidats parmi ceux présents dans les clauses '' WHERE" des requêtes d'interrogation. Ces attributs sont utilisés pour construire une matrice requêtes-attributs où les lignes représentent les requêtes de la charge et les colonnes représentent les attributs candidats.
3. *Application de l'algorithme 'Close' sur ce contexte* : La matrice résultat de l'étape précédente est exploitée par l'algorithme Close et donne lieu à un ensemble de motifs fréquents fermés. Chaque motif extrait est composé d'un ensemble d'attributs de l'entrepôt de données dont le support est supérieur ou égal à un seuil choisis par l'auteur.
4. *Génération des index candidats :* Durant cette étape et à l'aide du schéma de l'entrepôt de données (clés étrangères de la table de faits, clés primaires des tables dimensions, etc.) chaque motif extrait dans l'étape précédente va être analysé afin de vérifier s'il permet de générer un index de jointure binaire candidats ou non et un ensemble d'index candidats est créé.
5. *Construction de la configuration finale d'index :* Une approche ascendante d'élagage basée sur une recherche gloutonne dans l'ensemble des index candidats est choisie par l'auteur pour la construction de la configuration finale d'index.

Un modèle de cout mathématique qui calcule la taille de l'index ainsi que le cout d'exécution de la charge des requêtes (coût d'accès aux données à travers cet index) est utilisé pour définir trois fonctions objectifs : 1) Profit, 2) Ration profit/espace et 3) Hybride.

- *La fonction objectif Profit :* privilégie les index apportant le plus de profit lors de l'exécution des requêtes. Elle est utilisée quand l'espace de stockage n'est pas limité.
- *La fonction objectif Ration profit/espace :* privilégie les index apportant le plus de profit tout en occupant un minimum d'espace de stockage. Elle est utile quand l'espace de stockage est faible.
- *La fonction objectif Hybride* : combine les deux premières afin de sélectionner dans un premier temps les index apportant le plus de profit et de ne conserver dans un deuxième temps que ceux qui occupent le moins d'espace de stockage lorsque celui-ci devient faible. Elle est intéressante quand cet espace est moyennement grand

Cette approche est parmi les rares travaux qui ont traité le problème de sélection des IJB dans le contexte d'entrepôts de données modélisés par un schéma en étoile [11]. Son architecture modulaire lui permet d'être facilement adaptable à d'autres techniques d'indexation (il suffit d'adapter le module de génération des index et d'appliquer les modèles de coût correspondant à ces index.). Elle tire sa force des capacités des techniques de fouille de données pour extraire des connaissances utiles des données et présente l'avantage de générer des index mono et multi-attributs en un seul passage. Elle évite donc d'exécuter plusieurs itérations pour générer des index multi attributs comme dans [45]. Cependant, se baser sur la fréquence des attributs comme seule paramètres pour choisir les index les plus avantageux a été démontré insuffisant dans [12].

3.3.4 Travaux de Bellatreche et al. 2007

Les auteurs de [12] se sont basés sur les travaux qui ont été faites sur la fragmentation verticale [60] où il a été démontré que l'unique paramètre de fréquence n'est pas suffisant pour avoir un bon schéma de fragmentation. Ils ont constaté que dans l'approche de Aouiche on peut éliminer des index construit sur des attributs non fréquemment utilisés mais qui appartiennent à des tables de dimension volumineuses, or que le coût des opérations de jointure dépend fortement de la taille des tables jointes.

Ces constatation ont permet aux auteurs de juger que la fréquence d'utilisation d'un index par un maximum de requêtes comme seul paramètre pour générer les motifs fréquents fermés ne serait pas efficace pour sélectionner un ensemble d'index candidats. Les auteurs ont montré la limite de l'approche d'Aouiche à travers un exemple (voir [12]) et ont proposé d'inclure d'autres paramètres dans la génération des motifs fréquents comme la taille des tables de dimension, la longueur des tuples, la taille de la page système, etc.

L'approche proposée procède en deux grandes étapes : (1) génération des index candidats et (2) la sélection de la configuration finale d'index.

1- Génération des index candidats :

Comme pour l'approche d'Aouiche, l'approche de Bellatrche et al, commence par la construction du contexte de recherche des motifs fréquents à l'aide des attributs candidats. Mais pour générer les motifs fréquents fermés, les auteurs utilisent deux adaptations de deux algorithmes de recherche de motif fréquents fermés : Close et Charm appelés *DynaClose* et *DynaCharm* qui au lieu d'utiliser le paramètre

de fréquence comme seul paramètre d'élagage, ils utilisent une fonction de finesse appelée *Fitness* définit sur les attributs non-clés des motifs comme suit :

$$Fitness\,(X) = \frac{1}{n} \times (\sum_{i=1}^{n} \alpha_i \times sup_i)$$

Où : n représente le nombre d'attributs non clés A_i dans le motif X. sup_i représente le support de $A_i$et α_i est un paramètre pénalisant les attributs appartenant aux tables de dimension les moins volumineuses et défini par l'équation suivante : $\alpha_i = \frac{|D_i|}{|F|}$
Où : $|F|$, $|D_i|$ représentent respectivement le nombre de pages système nécessaires pour le stockage de la table des Faits F et la table de dimension D_i qui contient A_i. Pour l'élagage de l'espace de recherche des index, un motif X est retenu lorsque son support est supérieur ou égal à un seuil minfit calculé en utilisant un support minimal donné minsup : $minfit = \frac{minsup}{|F|} \times \sum_{j=1}^{d} \frac{|D_j|}{d}$

Où d représente le nombre de tables de dimension. Les motifs fréquents fermés générés par les algorithmes *DynaClose* et *DynaCharm* sont ensuite purifiés pour éviter des index non cohérents.

2- Sélection de la configuration finale d'index :

Une fois les motifs récupérés et purifiés, un algorithme glouton basé sur un modèle de cout mathématique adapté de celui d'Aouiche [17] est utilisé pour la création de la configuration finale d'index sous contrainte de l'espace stockage disponible. L'algorithme commence par une configuration comportant un index défini sur un attribut de plus faible cardinalité, recalcule le cout et améliore de manière itérative la configuration initiale en considérant d'autres attributs jusqu'à ce qu'aucune réduction supplémentaire de coût total de traitement des requêtes n'est possible ou la contrainte de stockage est violée.

3.3.5 Travaux de Ziani et al. 2010

Les auteurs [47] proposent d'exploiter les motifs fréquents maximaux qui sont moins nombreux que les motifs fréquents fermés afin de réduire l'espace de recherche des index candidats.

En effet, Le contexte d'extraction construit à partir de la charge des requêtes est souvent dense à cause de la corrélation des requêtes due à la corrélation de données interrogées [12]. Dans ce type de contexte, la génération de tous les motifs fréquents fermés n'est pas triviale et souffre des mêmes problèmes que les motifs fréquents. Un algorithme de recherche basé sur la recherche des motifs fermés peut avoir un

nombre de motifs fermés exponentiellement plus grand que l'ensemble des motifs fréquents maximaux générés.

Par définition, *Un Motif Fréquent Maximal est un motif dont tous les sur-ensembles sont non fréquents.* Il est donc clair que l'ensemble des motifs fréquents maximaux est inclus dans l'ensemble des motifs fréquents. Cette particularité va permettre de générer des index moins nombreux et plus pertinents[47]. Une autre particularité des motifs fréquents maximaux est que l'ensemble des motifs fréquents peut être généré directement à partir de l'ensemble des motifs fréquents maximaux. Elle préserve donc tous les attributs fréquents (candidats à l'indexation).

Plusieurs algorithmes ont été proposé pour la recherche des motifs fréquents maximaux. Dans leurs travaux, les auteurs ont choisis d'utiliser leur propre implémentation Java de l'algorithme FP-Max [61]. Ce dernier est présenté dans [65] comme étant le plus performant.

3.4 Synthèse

Dans ce chapitre, nous avons abordé le problème de sélection d'index et présenté quelques travaux qui existent dans la littérature traitant ce problème. Les approches étudiées procèdent généralement en deux grandes étapes : (1) *Sélection de l'ensemble d'index candidats.* (2) *Construction d'une configuration finale d'index.*

La première étape permet à partir d'une charge de requêtes de générer manuellement ou automatiquement un ensemble d'index candidats. Certains travaux utilisent des techniques de datamining pour réduire l'espace de recherche des index candidats.

La deuxième étape construit progressivement une configuration finale d'index, et ce à travers l'élagage des index candidats afin de respecter les contraintes du temps d'exécution et le coût de stockage. L'élagage est souvent fait par des algorithmes approximatifs (gloutons ascendant ou descendant) ou par des heuristiques (recuit simulé, algorithmes génétiques, ou par élimination basées sur des critères comme : des attributs de forte cardinalité, des attributs appartenant aux tables moins volumineuses, des attributs les moins utilisés, etc.

Le tableau II.1 présente une classification des principaux travaux suivant plusieurs critères qui sont :

- Le contexte sur lequel ont été appliqués (BDD ou ED).
- Le type d'index sélectionné.
- La méthode de sélection de l'ensemble d'index candidats.
- La méthode de construction de la configuration finale d'index.

– Le modèle de coût utilisée pour évaluer une configuration.

Critères ——————- Travaux	Contexte	Type d'index sélectionné	Sélection des index candidats	Construction de la configuration finale	Modèle de coût
Chaudhuri et al. [45]- 1997	BDD	Génériques sur une table	Automatique	Glouton (Ascendant)	Optimiseur + module what-if
Valentin et al. [40]- 2000	BDD	Génériques sur une table	Automatique	Heuristique (sac à dos)	Optimiseur
Feldman et al. [52]- 2003	BDD	Génériques sur une table	Automatique	Heuristique (éliminer et unifier les nœuds d'un graphe)	Mathématique
Kratica et al. [82]- 2003	BDD	Génériques sur une table	Manuelle	Heuristique (Génétique)	Mathématique
Golfareli et al. [41]- 2002	ED	liste de valeurs et bitmap	Automatique	Glouton (Ascendant)	Optimiseur
Boukhalfa et al. [48]- 2010	ED	IJB	Automatique (index par requête)	Heuristique (Elimination d'attribut)	Mathématique
Aouiche et al [17] - 2005	ED	IJB	Automatique par Datamining (motifs fermés)	Glouton (Ascendant)	Mathématique
Bellatreche et al. [12]- 2007	ED	IJB	Automatique par Datamining (motifs fermés + adaptation)	Glouton (Ascendant)	Mathématique
Ziani et al. [47]- 2010	ED	IJB	Automatique par Datamining (motifs maximaux)	Datamining	Mathématique

TABLE II.1: Synthèse des travaux sur la sélection d'index

3.5 Analyse et proposition

Vu le nombre important d'attributs indexables, le nombre d'index issus de la première étape de Sélection des index candidats peut être énorme, ce qui influe sur le processus du choix d'une configuration finale dans la deuxième étape. Il est donc primordial de faire appel à des techniques de réduction de l'espace de recherche des index candidats.

Les techniques d'élagage basées sur des algorithmes de datamining ont montré leur efficacité à réduire l'espace de recherche des index en calculant l'ensemble de motifs fréquents (fermés dans [12] [17] et maximaux dans [47]) au lieu de calculer toutes les combinaisons possibles d'attributs. Mais, en analysant les travaux existants, nous remarquons que :

- Le critère de la fréquence d'utilisation d'un index par un maximum de requêtes dans [17] n'est pas suffisant pour sélectionner les index les plus optimaux, et même le critère de la taille des tables de dimension rajouté au processus de génération des motifs fréquents fermés dans [12] n'est pas le seul. D'autres critères comme la cardinalité des attributs par exemple sont cruciaux et peuvent être intégrés au processus de sélection.
- Dans toutes les approches précédentes, le processus de sélection est purement automatique, l'administrateur n'a pas de moyen pour guider ce processus, ni pour exploiter son expertise pour spécifier ses préférences sur les index à extraire. Par exemple, l'administrateur peut exiger une cardinalité minimale aux attributs à retenir, la présence ou non de certains attributs ou encore la taille maximale / minimale (nombre d'attributs) des index recherchés, etc.
- Un autre inconvénient de ces approches, est que le processus de sélection des index s'effectue en deux étapes. La génération d'un grand nombre d'index, suivie d'une phase d'élagage appliquant des contraintes supplémentaires. Or que nous pensons qu'il serait préférable d'inclure ces contraintes plus tôt dans le processus de sélection, ce qui permet d'une part de diminuer le nombre d'index candidats générés et d'une autre part d'éviter cette deuxième phase d'élagage.

Les problèmes que nous venons de décrire, nous ont motivé à proposer une nouvelle démarche de sélection des IJB permettant de :

- Tirer profit de l'efficacité des techniques basées sur des algorithmes de datamining pour réduire l'espace de recherche des index.
- Permettre à l'administrateur d'exploiter son expertise en spécifiant ses préférences (sous forme de contraintes imposées au processus de sélection) sur les index à générer et les inclure au processus d'extraction. Ce qui permet de ne générer que des index apriori pertinents pour lui.
- Éviter la phase deuxième phase d'élagage. La différence de notre approche par rapport aux autres approches basées sur des techniques de datamining, est que notre approche cherche la collection de tous les motifs satisfaisant des prédicats appelés contraintes. Le but est d'utiliser efficacement les contraintes spécifiées par l'administrateur pour diminuer le nombre d'index générés, diminuer les

temps d'extraction en élaguant le plus tôt possible l'espace de recherche et de ne générer que les index qui sont a priori pertinents pour lui.

Les détails de cette approche seront présentés dans le chapitre suivant.

Chapitre III

Nouvelle approche pour la Sélection des index de jointure binaires basée sur l'Extraction de Motifs sous Contraintes

1 Introduction

Nous présentons dans ce chapitre notre approche de sélection d'IJB. Elle propose de profiter de l'expertise de l'administrateur pour guider le processus de sélection d'index et de prendre en compte ces préférences sur les index à générer pour réduire leur nombre. Cette approche prend en entrée les préférences de l'administrateur sous forme de contraintes et utilise une méthode de datamining appelée Extraction de Motifs sous Contraintes (EMC) pour générer des index qui sont a priori pertinents (respectent le contraintes de l'administrateur). Ces mêmes contraintes servent à minimiser l'utilisation des ressources (mémoire et espace de stockage) et permettent d'accélérer le processus de génération d'index en éliminant les index non désirables pendant le processus et non pas dans une deuxième phase à postériori.

Ce chapitre est organisé en deux grandes sections. Dans la première, nous présentons le paradigme d'Extraction de Motifs sous Contraintes et nous introduisant l'algorithme MUSIC-DFS permettant ce type d'extraction. Dans la deuxième section, nous détaillons notre stratégie de sélection d'index basée sur ce paradigme.

2 Paradigme d'extraction de motifs sous contraintes

Avant de présenter le paradigme d'extraction de motifs sous contraintes, nous commençons par les bases de l'extraction de motifs.

2.1 La recherche de motifs fréquents

Le concept d'extraction de motifs fréquents a été introduit la première fois en 1993 par Agrawal et al. Dans [3]. L'analyse du panier de la ménagère est l'une des applications typiques de la recherche de motifs fréquents. Utilisée dans la grande distribution, elle permet d'analyser les tickets de caisse des clients afin de comprendre leurs habitudes de consommation, agencer les rayons du magasin, organiser les promotions, gérer les stocks, etc. Dans les bases de données de vente, un tuple consiste en une transaction regroupant l'ensemble des articles achetés appelés *items*. Ainsi une base de données est un ensemble de *transactions*, qu'on appelle aussi *base transactionnelle* ou *base de transactions.*

2.1.1 Base transactionnelle ou contexte d'extraction.

La recherche des motifs fréquents fait l'hypothèse d'une base de données ''D" décrivant un ensemble de données (objets, transactions, requêtes,. . .) $O = o_1, ..., o_m$ par un ensemble d'attributs $A = a_1, ..., a_n$.

- Un objet $o \in O$ est décrit ou non par un attribut $a \in A$ de sorte que la base de données est assimilable à une relation binaire $R \subseteq O \times A$. où R(o, a) signifie que l'objet o possède l'attribut a.
- Le triplet (O, A, R) est appelé *contexte d'extraction.*

Exemple : La figure III.1 présente deux représentations équivalentes d'une base de données transactionnelle D. Elle représente un contexte d'extraction de 6 observations O identifié par l'ensemble des identificateurs {1, 2, 3, 4, 5, 6} sur 5 attributs A = {A,B,C,D,E}. Dans le tableau de droite chaque enregistrement est représenté par l'ensemble des attributs observés. Dans la partie gauche ces enregistrements sont présentés sous une forme « binaire » (un 1 représente la présence d'un attribut, un 0 son absence).

.

$D =$

O \ A	A	B	C	D	E
1	1	1	1	1	1
2	1	0	1	0	0
3	1	1	1	1	0
4	0	1	1	0	0
5	1	1	1	0	0
6	0	0	0	0	1

TID	Transactions
1	A B C D E
2	A C
3	A B C D
4	B C
5	A B C
6	E

FIGURE III.1: Deux représentations d'une base transactionnelle D

2.1.2 Motif

- On appelle *Motif ensembliste* ou *Motif* (*itemset*) tout sous-ensemble d'attributs de A.
- Un motif constitué de k attributs sera appelé un *k-motif.*
- Un motif M décrit un objet o quand $\forall a \in M, R(o, a)$ (l'objet o possède l'attribut a.)

Exemple : Le motif {A,B,C} est un 3-motif noté ABC. Il décrit les objets : 1, 3, 5

2.1.3 Support

Le *support* d'un motif M, noté *Supp(M)* est la proportion de transactions de D contenant M.

$$Supp\,(M) = \frac{|\{o \in O / M \subseteq o\}|}{|O|}$$

Exemple : Dans l'exemple de la figure III.1, on a supp(BC)= 0.66 (= 66 %) vu que le motif AB apparaît dans 4 observations parmi 6.

2.1.4 Motif fréquent

Soit un seuil $minsup \in [0, 1]$, appelé *support minimum.* Un motif M est dit *fréquent* si : $supp(M) \geqslant minsup$.

Exemple Dans l'exemple de la figure III.1, pour un «minsup = 3/6 »

- *{B, C, D} est un 3-motif de support 2/6 (ce n'est un motif fréquent) ;*
- *{B, C} est un 2-motif de support 4/6. (C'est un motif fréquent.)*

2.1.5 Problème de recherche des motifs fréquents

Le problème de recherche des motifs fréquents associé aux données (O, A, R, f_{min}) consiste à déterminer le sous-ensemble $M_f \subseteq M$ des motifs fréquents ainsi que la fréquence de chacun.

2.2 Extraction de motifs sous contraintes

L'extraction de motifs sous contrainte est un thème de recherche très étudié ces dernières années [67-78]. Tandis que l'extraction de motifs a pour but la découverte d'informations à partir de tous les motifs ou d'un sous ensemble de motifs, l'extraction sous contraintes cherche la collection de tous les motifs satisfaisant un prédicat appelé contrainte (propriétés qu'ils doivent satisfaire). L'une des contraintes sera généralement la contrainte de fréquence minimale qui permet de rechercher des régularités au sein d'une base de données, mais l'on peut associer d'autres mesures d'intérêt comme, e.g., la présence ou l'absence de certains attributs .

L'objectif de ces travaux est d'essayer de tirer profit des contraintes pour réduire la taille de l'espace de recherche, ainsi que de privilégier la capacité de l'utilisateur à interagir avec le système en spécifiant précisément ce qui l'intéresse.

2.2.1 Motivations de l'extraction de motifs sous contraintes

L'extraction des motifs sous contraintes est motivée par la prise en compte des contraintes fixées par l'analyste pour les raisons suivantes :

1. Cibler la recherche des informations à extraire suivant les centres d'intérêts de l'utilisateur. L'intérêt des motifs extraits est donc garanti par le point de vue de l'analyste exprimé à travers la sémantique de la contrainte [84].
2. Outre le fait qu'une contrainte permet de mieux cibler les motifs extraits, leur utilisation est parfois indispensable pour rendre l'extraction faisable. En effet, il est généralement impossible d'extraire tous les motifs (pour 1000 attributs, nous avons environ 10^{300} motifs possibles et décompter les fréquences dans des millions de transactions peut être très couteux) [84]. Le principe des extractions sous contraintes est de «pousser les contraintes au processus d'extraction » afin d'éviter de réaliser une extraction de tous les motifs fréquents suivi d'une deuxième phase de traitement de filtrage appliquant ces contraintes supplémentaires.
3. L'intégration au plus tôt des différents critères de sélection à l'algorithme d'extraction permet de réduire considérablement le temps de calcul et de minimiser les ressources (processeur, mémoire) utilisées pour l'extraction de la réponse.
4. Le nombre de motifs fréquents est souvent prohibitif et peu intelligible pour un utilisateur humain qui se retrouve noyé au milieu d'informations triviales ou redondantes. Le but de l'extraction sous contraintes est de réduire la taille de l'ensemble des motifs en un ensemble de motifs intéressant l'expert, permettant ainsi une meilleure interprétation des résultats.

5. Les processus d'extraction des connaissances à partir des bases de données sont principalement interactifs. L'utilisateur doit intervenir du bout en bout de ce processus pour préparer les données, choisir la bonne technique d'extraction, le paramétrage des extracteurs, la validation des résultats, pour enfin décider d'utiliser de celles-ci ou d'itérer à nouveau. L'extraction sous contraintes privilégie la capacité de l'utilisateur à interagir avec le système. Elle permet à l'analyste d'exploiter sa connaissance du domaine de manière interactive pendant l'extraction en exprimant son intérêt subjectif sous la forme de contraintes que l'extracteur doit pouvoir essayer d'exploiter.

2.2.2 Contrainte

- Soit $P(A)$ désigne l'ensemble des parties de l'ensemble des attributs A.
- Une *contrainte C*, (appelée également *requête* ou *prédicat*) est un prédicat booléen défini sur *P(A)*, i.e., *C : P(A)→{vrai, faux}.*
- Un motif $M \in 2^A$ satisfait la contrainte C sur la base de données D ssi *C(M, D) = vrai.*

Exemple : Soit la contrainte C_{freq} qui impose aux motifs d'être 0.6-fréquents, les motifs {AC}, {BC} satisfont cette contrainte.

2.2.3 Table des valeurs

La vérification de certaines mesures d'intérêts sur les motifs à extraire nécessite de compléter la base transactionnelle D avec des informations supplémentaires pour chaque attribut dans A, (e.g., une table associant des valeurs à chaque attribut). Cette table permet de définir d'autres contraintes du type : $Agre(X)\,\theta\,\alpha$.

Où : *Agre* est un opérateur d'agrégation {MAX, MIN, SOM, MOY,..},$\theta \in \{<, >, =, \leq\ , \geq\}$ et $\alpha \in \Re$.

Par exemple dans la table III.1, lorsque les attributs désignent les articles d'un magasin, un prix peut être associé à chaque article pour rechercher les motifs dont la moyenne des prix des articles le composant soit inférieure à un seuil.
$C_{AvgPrix}(X) \equiv MOY(X.prix) \leq minPrix.$

Attribut / Propriété	*A*	*B*	*C*	*D*	*E*
Quantité	1200	700	200	1500	20
Prix	10	50	100	20	600
Type	Boisson	Laitier	Laitier	Laitier	Viande
...	...	...	...	...	...

TABLE III.1: Table de valeurs associée à la base transactionnelle D de la figure III.1

2.2.4 Problème d'extraction de motifs sous contraintes

L'objectif de l'extraction de motifs sous contraintes consiste à extraire tous les motifs présents dans une base transactionnelle D et satisfaisant une contrainte C.

Exemple : Considérons la base de données transactionnelle D de la figure III.1

- *Si la contrainte de fréquence C_{freq} spécifie qu'un motif doit être 0.6 fréquent, alors la requête : $C_{freq}(2^A)$ donne : {A,B,C,AC ,BC }.*
- *Soit les contraintes $C_{size}(X) \equiv |X| \leq 3$ et $C_{miss}(X) \equiv B, E \bigcap X = \phi$.*
  La requête : $C_{size} \wedge C_{miss}(2^A)$ donne : {A, C, D, AC, AD, CD, ACD }, tandis que la requête : $C_{freq} \wedge C_{size} \wedge C_{miss}(2^A)$ donne : A, C, AC.

2.2.5 Classes et propriétés des contraintes

Les propriétés des contraintes sont des éléments déterminants pour l'extraction. Au lieu de pousser une contrainte particulière, les extracteurs de motifs sont souvent dédies à un ensemble de contraintes (appelé classe de contraintes). En effet, L'idée derrière les stratégies d'élagage d'espaces de recherche est de pouvoir écarter sans perdre de bons candidats de grands espaces de recherche (espace de motifs). Cet élagage pour être réalisée doit être basé sur des propriétés des contraintes [85]. Par exemple, *l'anti-monotonie* permet un élagage efficace de l'espace de recherche, et la diminution du nombre de motifs à considérer elle est exploitée par la majorité des algorithmes. Les contraintes *succinctes* permettent d'énumérer directement les candidats sans nécessairement explorer tout l'espace de recherche en générant et testant toutes les motifs possibles.

Dans cette section, nous présentons les principales classes de contraintes ainsi que leurs propriétés. D'autres classes de contraintes ont été étudiées dans la littérature, que nous ne présenterons pas dans cette section (pour un état de l'art sur les classes de contraintes voir [86]).

2.2.5.1 Contrainte monotone et anti-monotone

Introduites par Mannila et Toivonen en 1997 [87].

Définition : *Une contrainte de motif anti-monotone "C" est une contrainte telle que pour tous les motifs S, S' : (S' $\subseteq$S $\wedge$ S satisfait C) $\Rightarrow$ S' satisfait C.*

L'anti-monotonie signifie que si un motif quelconque S' viole la contrainte C, alors tous sur-motif S de S' la viole aussi.

Exemples *: les trois contraintes suivantes sont anti-monotones :*

1. *C_{3freq} : une contrainte qui impose aux motifs d'être 3-fréquents.*

Dans la base transactionnelle D de la figure III.1 le motif CD est 2-fréquent, il viole la contrainte C_{3freq}, alors les motifs ABCDE et ABCD la-viole aussi.

2. *$C_{sizemax}$: une contrainte sur la taille maximale d'un motif définie par* $C_{sizemax}(X) \equiv |X| \leq 2$.
   Le motif ABC*, viole cette contrainte et il est évident que ses sur-motifs ABCD et ABCDE le font aussi.*
3. *C_{miss} : une contrainte de non-présence d'un motif particulier définie par :* $C_{miss}(X) \equiv B \notin X$.
   Le motif BC, viole cette contrainte et ses sur-motifs ABC, ABCD et ABCDE la viole aussi.

L'anti-monotonicité est une propriété importante, parce qu'elle *permet des élagages dans l'espace de recherche.* Elle est l'une des plus facile à prendre en compte, les algorithmes d'extraction par niveaux tel qu'APRIORI l'utilisent la plupart du temps pour élaguer l'espace de recherche. En effet, quand un ensemble ne satisfait pas la contrainte, ses spécialisations non plus et elles peuvent donc être élaguées [74].

Définition : *Une contrainte monotone est une contrainte C telle que pour tous les motifs S, S' : (S ⊆S' ∧ S satisfait C) ⇒ S' satisfait C.*

La monotonie est une propriété qui stipule que si un sous-motif S de S' satisfait une contrainte C, alors S' le satisfait aussi. Ce type de contrainte *permet des générations de candidats efficaces.*

Exemples **:**

1. *une contrainte sur la somme des prix des articles d'un motif $C_{SomPrix}$ définie par :* $C_{SomPrix}(X) \equiv Sum(X.Prix) \geq 500$ *est une contrainte monotone. Il est évident que si un motif X satisfait cette contrainte, alors tous ces super-ensembles satisfont cette contrainte.*
2. *Les contraintes {A, B, C, D}⊂ X et {A, B, C} ∩ X ≠ ∅ sont aussi monotones.*

On remarque que la négation d'une contrainte monotone est anti-monotone et vice versa.

2.2.5.2 Contrainte Succincte

Informellement, une contrainte succincte [88] est une contrainte telle que pour déterminer si un motif X satisfait ou non cette contrainte, il faut que chaque attribut

composant le motif vérifie la contrainte.

Exemple *: considérons la contrainte min(X.prix) ≥ v, elle est anti-monotone et succincte. Ainsi, si on commence avec un premier ensemble de motifs candidats composé de tous les attributs ayant un prix supérieur à v, on peut générer tous les motifs satisfaisants cette contrainte.*

Une propriété importante des contraintes succinctes est qu'elles permettent d'énumérer exactement tous les motifs la-satisfaisant sans devoir tout générer et tester (pre-counting prunable). L'élagage peut donc être réalisé sans génération de candidats et sans accès au jeu de données.

Remarque : La seule satisfaction de la contrainte n'est pas affectée par le calcul itératif de support.

2.2.5.3 Contraintes convertibles

Introduites dans [56] par Pei et al.

Supposons que tous les attributs dans les motifs sont triés selon l'ordre O :

– Une contrainte C est *convertible anti-monotone* ssi :

1. C n'est pas anti-monotone
2. un motif S satisfait C implique que chaque suffixe de S (respectivement à O) satisfait aussi C.

– Une contrainte C est *convertible monotone* ssi :

1. C n'est pas anti-monotone
2. un motif S satisfait C implique que chaque motif dont S est un suffixe (respectivement à O) satisfait aussi C.

L'idée des contraintes convertibles est que pour une contrainte qui est ni monotone ni anti-monotone, on puisse ordonner les attributs tels qu'il existe une relation stable entre la satisfaction de la contrainte et les préfixes des motifs [85].

Exemple : *Soit S l'ensemble de valeurs (par ordre décroissant) {9, 8, 6, 4, 3, 1}, Avg(S) ≥ v est monotone convertible*

– *Si S est un suffixe de S1, alors avg(S1) ≥ avg(S)*
  – *{8, 4, 3} est un suffixe de {9, 8, 4, 3}*
  – *avg({9, 8, 4, 3})=6 ≥ avg({8, 4, 3})=5*
– *Si S satisfait avg(S)≥ v, alors S1 aussi*
  – *{8, 4, 3} satisfait avg(S) ≥ 4, ainsi que {9, 8, 4, 3}*

2.2.5.4 Exemples sur les classes de contraintes

Le tableau III.2 donne quelques exemples de contraintes appartenant aux différentes classes citées auparavant (provenant de [86]). Ces contraintes s'appuient sur la base de données transactionnelle de la figure III.1

Contrainte \ Classe	Anti-monotone	monotone	succincte	Convertible Anti-monotone	Convertible monotone
Q1 ≡ min (X.Qté) ≥ 500	X		X	X	X
Q2 ≡ max (X.Qté) ≥ 30		X	X	X	X
Q3 ≡ X.type ⊇ {Laitier, Viande}		X	X		X
Q4 ≡ Q1 ∧ Q2			X		
Q5 ≡ Q1 ∨ Q2		X	X		X
Q6 ≡ Q1 ∨ (¬Q2 ∧ Q3)			X		
Q7 ≡ X.Prix = 25	X		X	X	
Q8 ≡ X.type ⊆ {Laitier, Viande}	X		X	X	
Q9 ≡ Viande ∈ X.type		X	X		X
Q10 ≡ max (X.Prix) / avg (X.Prix) ≤ 7				X	X

TABLE III.2: Exemples sur les classes des contraintes en s'appuyant sur la table des valeurs III.1

2.3 Algorithmes d'extraction de motifs sous contraintes

De nombreux algorithmes ont été proposés et tentent d'utiliser efficacement les contraintes pour diminuer les temps d'extraction en élaguant le plus tôt possible l'espace de recherche. Pour se faire, deux stratégies de recherche différentes sont adoptés : 1) *la recherche en largeur d'abord*, appelée aussi *par niveau*, et 2) *la recherche en profondeur d'abord*.

Dans les algorithmes de *recherche par niveau*, la base de données est généralement utilisée telle quelle et on recherche les motifs en augmentant leurs tailles à chaque étape, à la $i^{ème}$ itération, les i-motifs découverts sont utilisés pour générer les (i + 1)-motifs candidats à l'étape suivante.

Dans le domaine des algorithmes par niveau, L'algorithme le plus connu, *Apriori*, se focalise sur la contrainte de *fréquence* pour les ensembles de motifs [91] mais il est facilement adaptable pour n'importe quelle contrainte anti-monotone. Dans le même domaine, l'algorithme *CAP (Constrained APriori)* [88] permet d'extraire des motifs satisfaisant la conjonction de contraintes de fréquence, anti-monotone et succincte. Il se diffère par rapport à Apriori à la phase de génération des nouveaux candidats

où il prend en considérations les contraintes succinctes. L'algorithme *DualMiner* [89] considère simultanément les contraintes monotone et anti-monotone.

Les algorithmes basés sur une recherche en profondeur utilisent généralement des structures d'arbres *FP-tree* (Frequent-Pattern tree) [90], ou des structures similaires pour résumer la base de données et le parcourir en profondeur afin de générer tous les motifs fréquents. Tous les algorithmes proposés dans la littérature pour extraire des motifs sous une contrainte convertible comme *CFG (Constrained Frequent pattern Growth)* et *FIC (Frequent Itemsets with Convertible)* font partie de ce domaine [86]. L'algorithme *FPS (FP-tree and Succinct)* utilise les FP-trees pour extraire des contraintes succinctes.

Dans notre stratégie de sélection d'index nous avons fait recours à l'algorithme *Music-dfs* [86] dont la présentation est l'objectif de la section suivante.

2.3.1 L'algorithme MUSIC -DFS

MUSIC-DFS [86] (Mining with a User-SpecifIed Constraint - Depth-First Search) est un extracteur correct et complet de motifs locaux basé sur une recherche en profondeur qui permet d'extraire efficacement l'ensemble des motifs satisfaisant des contraintes variées. Ces contraintes sont exprimées à travers seulement quelques dizaines de *primitives syntaxique et d'agrégation* , mais qui répondent à des problèmes pourtant très divers. Une liste des contraintes décrites récursivement en se basant sur les primitives $\{\wedge, \vee, \neg, <, \leq, >, \geq, \subset, \subseteq, \supset, \supseteq, +, =, /, count, freq, sum, max, min, \cap, \cup, \setminus\ \}$ et reconnues par le solveur *Music-dfs* est présentée dans [86].

L'efficacité de *Music-dfs* repose sur une stratégie de génération de candidats en profondeur et sur une capacité à élaguer l'espace de recherche en se basant sur la de la condition d'élagage issue des contraintes et de la propriété d'anti-monotonie. Les conditions d'élagage sont fondées sur l'utilisation d'intervalles et non pas sur les sur-ensembles ou sous-ensembles d'un motif. Un intervalle est un ensemble de motifs qui incluent le même motif et qui sont inclus dans sa fermeture [86].

Exemple : *L'intervalle [{1,3},{2,4}] représente les motifs suivante : {{1,3},{1,3,2},{1,3,4},{1,3,2,4}}.*

Les motifs produits par Music-dfs sont sous forme de représentation condensée composée d'intervalles disjoints où chaque intervalle synthétise un ensemble de motifs satisfaisants la contrainte.

Motivation de notre choix d'algorithme

- Music-dfs s'avère pour l'utilisateur un outil efficace entièrement automatique offrant une grande flexibilité au niveau de la contrainte donnée par l'utilisateur comme un simple paramètre.
- Permet à l'utilisateur d'exprimer un large éventail de contraintes (un exemple des contraintes est présenté dans la table III.3).
- Intègre des primitives exploitant des données externes hétérogènes (propriétés des attributs, donnée textuelles,...).
- La richesse combinatoire des primitives qu'il fournit englobe les classes de contraintes les plus usuelles dont les contraintes monotones et anti-monotones.
- L'utilisateur peut librement construire sa contrainte.
- La complétude et la correction de Music-dfs permettent à ce dernier d'assurer que tous les motifs vérifiant un ensemble de contraintes soient trouvés (complétude) et que seulement ceux-là soient trouvés (correction).

$q_1 \equiv count(X) \geq \gamma$	$q_8 \equiv sum(X.val)/length(X) \geq 50$
$q_2 \equiv count(X) \times length(X) \geq 2500$	$q_9 \equiv min(X.val) \geq 30 \Lambda max(X.val) \leq 90$
$q_3 \equiv X \subseteq A$	$q_{10} \equiv max(X.val) - min(X.val) \leq 2 \times length(X)$
$q_4 \equiv X \supseteq A$	$q_{11} \equiv length(X \setminus A) > length(X \cap A)$
$q_5 \equiv length(X) \geq 10$	$q_{12} \equiv q_1 \Lambda q_2$
$q_6 \equiv sum(X.val) \geq 500$	$q_{13} \equiv q_6 \Lambda \neg q_5$
$q_7 \equiv max(X.val) < 50$	$q_{14} \equiv q_5 \Lambda q_7$

TABLE III.3: Exemples de contraintes peuvent être résolues par Music-DFS

3 Approche de sélection d'IJB par extraction de motifs sous contraintes

Dans cette section nous détaillons les différentes étapes de notre approche de sélection des Index de Jointure Binaire basée sur l'extraction de motifs sous contraintes. Notre stratégie s'inscrit dans le cadre des approches basées sur des techniques de Datamining, et dans la poursuite des travaux dans [17], [12] et [47], afin de :

- Rendre le processus de sélection d'index interactif en exploitant l'expertise de l'administrateur.
- Cibler la recherche des informations à extraire suivant ses centres d'intérêts,
- Éviter une deuxième phase de traitement et de filtrage appliquant des contraintes supplémentaires,

- Optimiser le processus d'extraction en optimisant l'utilisation des ressources système,
- Réduire l'ensemble d'index générés en un ensemble d'index apriori pertinents pour l'administrateur.

Notons que dans notre approche nous n'avons pas considéré la contrainte de l'espace de stockage alloué aux index, nous supposons que l'administrateur dispose de suffisamment d'espace disque pour contenir les index créés et qu'à travers la sévérité des contraintes qu'il impose, il peut contrôler la taille de la configuration d'index.

3.1 Architecture générale de l'approche

L'approche que nous proposons, dont le principe général est représenté à la figure III.2, considère un entrepôt de données relationnel modélisé par un schéma en étoile et une charge de requêtes décisionnelles exécutée dessus. Elle exploite cette dernière ainsi que l'expertise de l'administrateur spécifiée sous forme de contraintes imposées au système afin d'en extraire une configuration d'index améliorant le temps d'accès aux données dans un processus interactif de sélection d'index.

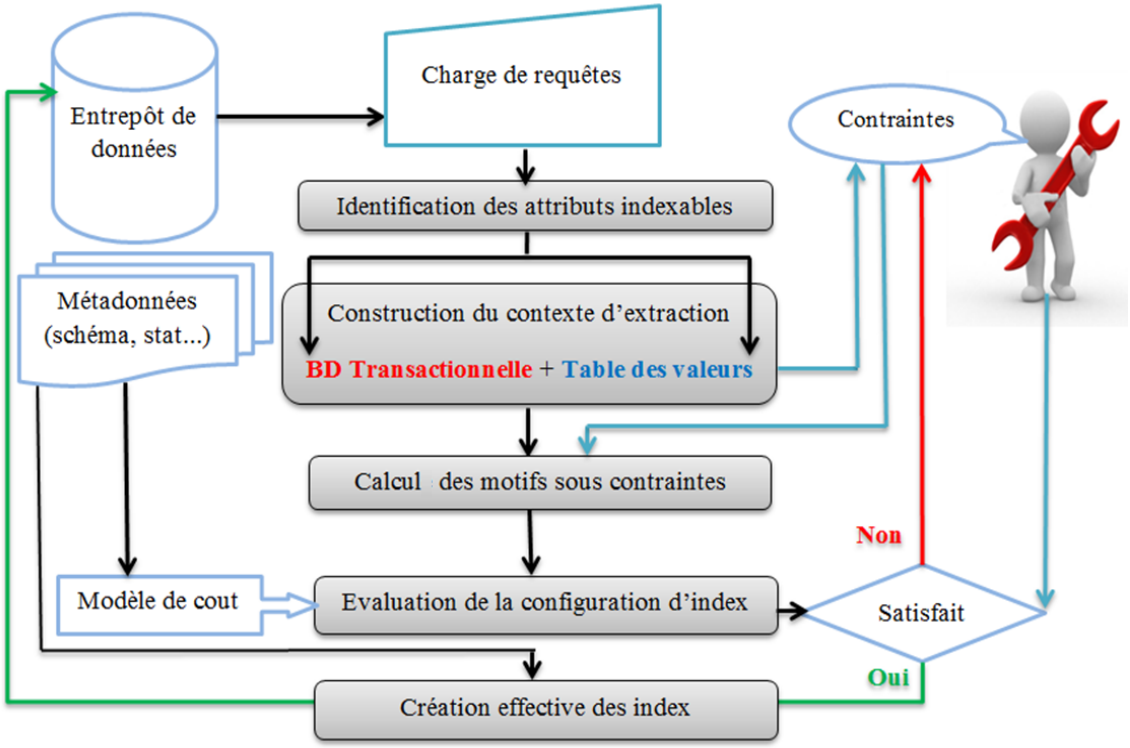

FIGURE III.2: Notre approche de sélection d'IJB basée sur l'extraction de motifs sous contraintes

La première étape, consiste à analyser une charge de requêtes jugée représentative de l'ensemble des requêtes adressées au système afin d'extraire les attributs indexables. Ces attributs sont stockés dans un fichier où chaque ligne contient la liste des attributs indexables de la requête correspondante. Ce fichier représente notre base de données transactionnelle, et il est complété par un autre fichier contenant des informations supplémentaires sur chaque attribut. C'est notre table de valeurs. Il est nécessaire pour permettre à l'administrateur de définir des mesures d'intérêt sur les motifs à extraire. Ces deux fichiers forment notre contexte d'extraction.

Le contexte d'extraction, couplé avec un script exprimant les préférences de l'administrateur sur les motifs à générer vont former les entrées de l'extracteur sous contraintes Music-DFS. Ce dernier, va les-exploiter pour extraire à partir de la base de donnée transactionnelle, les motifs satisfaisants les contraintes définies par l'administrateur en utilisant les informations sur les attributs contenues dans la table des valeurs.

Un modèle de coût mathématique est utilisé pour estimer la bonneté de la configuration d'index choisis par l'extracteur. Si l'administrateur est satisfait du résultat, l'ensemble des index est créé et rajouté à l'entrepôt. Sinon, l'administrateur peut reformuler une nouvelle requête où il refait ces choix pour améliorer le gain en coût d'exécution ou en espace de stockage.

Les grandes étapes de notre approche sont donc :

1. Analyse de la charge de requêtes et identification des attributs indexables ;
2. Construction du contexte d'extraction ;
3. Appliquer l'algorithme Music-DFS sur ce contexte pour calculer les motifs sous contraintes ;
4. Évaluation de la qualité de la solution.
5. Construction de la configuration d'index ;

Nous détaillons chacune de ces étapes dans les sections qui suivent.

3.2 Identification des attributs indexables ;

La première étape de notre approche consiste à faire une analyse syntaxique sur une charge de requêtes extraite à partir du journal des transactions de l'SGBD, et jugée représentative de l'ensemble des requêtes décisionnelles adressées au SGBD durant une période de temps. Le but étant d'identifier les attributs indexables. Dans le contexte des entrepôts de données les requêtes sont de type interrogations ou rafraîchissements, mais ces derniers ne sont réalisés que périodiquement, Donc

nous ne considérons que les requêtes de type *SELECT* dont la forme est la suivante :

SELECT Arep, AGR(Aagr)
FROM, Di, F
WHERE PJ, PS
GROUP BY [Agroup]

Tel que : Arep *: attributs de réponse de la requête.* AGR *: Fonction d'agrégation (Max, Min, Sum, Count, Avg),* Aagr *: attributs l'agrégation.* PJ *: prédicats de jointure,* PS *: prédicats de de sélection.* AGroup *: Attributs de groupement.* Di *: l'ensemble des tables de dimensions concernées par la requête*, F : *la table des faits*

Les *attributs indexables* sont les attributs non clés des tables de dimension présents dans les clauses *WHERE* des requêtes de la charge et sur lesquels un prédicat de sélection *PS* est défini.

3.3 Construction du contexte d'extraction

Les attributs extraits dans l'étape précédente sont utilisés pour construire un contexte d'extraction. Ce dernier, dans le cas de l'extraction sous contraintes est composé de deux fichiers, le premier représente la Base de données Transactionnelle et le deuxième représente la Table de valeurs.

3.3.1 Construction de la Base de données Transactionnelle :

Chaque ligne du fichier représentant la Base de Données Transactionnelle correspond à une requête. Elle contient la liste des attributs indexables contenu dans cette requête.

Pour l'outil Music-DFS, ce fichier porte l'extension « .bin »

3.3.2 Construction de la Table de valeurs

À partir des attributs indexables, nous construisons une matrice « *Attributs - Valeurs*» où chaque ligne représente un attribut et chaque colonne une valeur (informations supplémentaires) sur l'attribut.

Exemple : *Si l'on a 5 attributs {A1, A2, A3, A4, A5} dont les cardinalités sont respectivement {10, 200, 12, 2, 365} et appartenant aux tables de dimension {T1, T1, T2, T1, T3} tel que les tailles en octet des trois tables {T1, T2,*

T3} sont respectivement {800, 200, 9000} et l'on veut construire une table de valeurs contenant ces informations, le fichier aura la forme d'une table telle que la table III.4.

Card	*TabDim*	*TailleTabDim*
10	T1	800
200	T1	800
12	T2	200
2	T1	800
365	T3	9000

TABLE III.4: Exemple de table des valeurs permettant de définir des contraintes sur les attributs constituants un IJB

La première ligne de la table des valeurs liste les noms des valeurs. La deuxième signifie que l'attribut A1 à une cardinalité de 10 et qu'il appartient à la table de dimension T1 qui fait 800 octets de taille. La troisième ligne contient les informations pour l'attribut A2 et ainsi de suite.

Ces informations sont utiles pour définir des contraintes sur la cardinalité des attributs composants les motifs à extraire, les tables de dimension auxquelles ils appartiennent ou leurs tailles.

Pour l'outil Music-DFS, ce fichier porte l'extension « .att »

3.4 Calcul de motifs sous contraintes

Après avoir mis en forme les donnée d'entrées, le tour est à l'administrateur pour spécifier ces préférences sur les motifs à générer sous forme de contraintes. Dans la section suivante, nous présentons les différents types de contraintes que l'administrateur peut spécifier. Ces contraintes sont ceux déjà présent dans les travaux de sélection d'index et reconnues décisives pour ce type de problème à savoir *a) Contrainte de fréquence b) Contrainte sur la cardinalité des attributs c) Contrainte sur la taille des tables de dimensions d) Contrainte sur la longueur des index, e) Contraintes sur les attributs formants les index.*

Cependant, l'administrateur est libre d'utiliser n'importe quelle contrainte qui peut être formulée avec les primitives de l'extracteur Music-DFS.

3.4.1 Les contraintes utilisées :

a) Contrainte de fréquence :

La contrainte de fréquence demande que le support de tout motif satisfaisant la contrainte soit supérieur ou égal ou support minimal. $Support(X) \geq minsup$.

Cette contrainte permet créer des index sur un attribut ou un groupe d'attributs les plus fréquemment utilisés et de maitre en évidence la corrélation des attributs.

Exemples :

- *"music-dfs -i dataset.bin -q "count(X)>=20 ;" -o motifs.txt"*
  Cette requête permet de générer les motifs X dont le nombre de transactions contenant X est supérieur ou égale à 20 (Fréquence minimale absolue).
- *"music-dfs -i dataset.bin –t 0.05 -o motifs.txt"*
  Cette requête permet de générer les motifs X dont le rapport entre le nombre de transactions contenant X et le nombre total de transactions est supérieur ou égale à 5% (Fréquence minimale relative).

b) Contrainte sur la cardinalité des attributs :

La contrainte de cardinalité exige que chaque attribut composant le motif X soit d'une cardinalité (nombre de valeurs distinctes) inférieur à un seuil donné. $Max(X.cardinalité) \leq MaxCard$.

Il est à noter que les valeurs des cardinalités des attributs sont contenues dans la table des valeurs.

Cette contrainte permet limiter la création des index sur des attributs ayant une cardinalité définie par l'administrateur selon l'espace de stockage disponible (généralement des attributs de faible et moyenne cardinalité) afin de minimiser la taille des index.

Exemple :

- *La requête : "music-dfs -i dataset.bin -q "*$count(X) \geq 20\, and\, Max(X.cardinalité) \leq 10$*;" -o motifs.txt"*
  Permet de générer les motifs X dont le nombre de transactions contenant X est supérieur ou égale à 20 et dont la cardinalité de chaque attribut ne dépasse pas 10.

c) Contrainte sur la taille des tables de dimensions :

La contrainte sur la taille des tables de dimensions permet de générer des index sur les attributs appartenant à des tables volumineuses. Donc, une taille minimale pour les tables de dimensions *TailleMin* est définie par l'administrateur et chaque motif contenant un attribut appartenant à une table de dimension dont la taille est inférieure à cette *TailleMin* est supprimé pendant le processus de sélection. $Min(X.TailleTabDimension) \geq TailleMin$.
Cette contrainte permet de créer des index sur les attributs appartenant à des tables de dimension de grandes tailles ce qui permet de réduire considérablement le coût des opérations de jointure car ce dernier dépend fortement de la taille des tables jointes.

Exemples :

- *La requête : "music-dfs -i dataset.bin -q "count(X)≥20 and Min(X.TailleTabDimension)≥8000 ;" -o motifs.txt"*
  Permet de générer les motifs X dont le nombre de transactions contenant X est supérieur ou égale à 20 et dont la taille des tables des dimension des attributs constituants le motif dépasse 8000 Octets.
- *La requête : "music-dfs -i dataset.bin -q "count(X)≥20 and Min(X.TailleTabDimension)≥8000 and Max(X.cardinalité)≤10 ;" -o motifs.txt"*
  Permet de rajouter aux motifs de la requête précédente, une contrainte sur la cardinalité des attributs.

d) Contrainte sur la longueur des index :

On peut imposer une Longueur (nombre d'attributs) minimale/ maximale ou exacte aux index générés.

Exemple : *pour une fréquence absolue minimale supérieure ou égale à 20, l'administrateur peut exiger de ne générer que des :*

- *index mono-attribut :* "music-dfs -i dataset.bin -q "count(X)≥20 and length(X) = 1 ;" -o motifs.txt"
- *index multi-attributs :* "music-dfs -i dataset.bin -q "count(X)≥20 and length(X) > 1 ;" -o motifs.txt"
- *index composés de trois attributs :* "music-dfs -i dataset.bin -q "count(X)≥20 and length (X)= 3 ;" -o motifs.txt"

e) Contraintes sur les attributs formants les index :

La contrainte élément *:*

On peut exiger/refuser la présence/ l'absence d'un attribut.

Exemple : *"music-dfs -i dataset.bin -q "count(X)>=20 and [not {1} subseteq X];" -o motifs.txt".*

Cette requête refuse la présence de tout motif contenant l'attribut 1 même s'il a une fréquence minimale acceptable.

La contrainte motif :

on recherche les motifs contenant un sous-ensemble d'attributs.

Exemple : *dfs -i dataset.bin -q "count(X)>=20 and {2,5} subset X;" -o motifs.txt"*
Cette requête permet de générer les motifs X ayant une fréquence absolue minimale supérieure ou égale à 20 et contenant le sous ensemble de motifs {2,5}.

3.5 Évaluation de la qualité de la solution

Pour mesurer l'efficacité de la configuration d'index crée, nous utilisons un modèle de coût mathématique simplifié adapté de celui de Aouiche [17], dans lequel

- Nous n'avons fait aucune supposition sur l'implémentation physique des index (B-arbre ou autres).
- Nous avons rajouté la taille du RowID dans la formule du coût de stockage des index.
- Nous avons pris en compte seulement le coût d'accès aux données à travers ces index. Le coût de maintenance des index n'est pas pris en considération car dans les systèmes de type OLAP les mises-à-jour se font sous forme de Batchs (jobs) périodiques pour rafraîchir les données et généralement lors de l'inactivité du système.

Le Tableau III.5 résume les notations du modèle adopté.

Paramètre	Signification
F	Table des faits
|X|	Nombre de tuples de la table X ou cardinalité de l'attribut X.
||T||	Nombre de pages nécessaires pour stocker la table T
Ps	Taille en Octet d'une page système
d	Nombre de bitmap utilisé pour évaluer une requête

TABLE III.5: Paramètres des modèles de coût

3.5.1 Coût de stockage d'un index de jointure binaire

Un IJB est défini sur un ou plusieurs attributs non clés des tables de dimension $A_1, A_2, \ldots, A_n$ pour pré-calculer les jointures entre la table de faits F et une ou plusieurs tables de dimension. Donc, un IJB contient autant de lignes que le nombre de tuples de la table des faits. Chaque ligne contient un identificateur de ligne (RowId) ainsi qu'une suite de vecteurs binaires. Chaque vecteur représente une valeur de l'attribut indexé Ai et il contient autant de bitmaps que la cardinalité de l'attribut. Ainsi, la taille en Octet d'un index de jointure binaire peut être donnée par [17] :

$$Taille\ \ (I) = \frac{(|RowID| + \ \sum_{i=1}^{n} |A_i|) \ \times |F|}{8}\ Octets$$

La taille d'un IJB dépend donc de la taille de la table de faits (nombre de tuples) et de la cardinalité des attributs sur lesquelles il est construit.

3.5.2 Coût d'accès aux données

Le coût d'accès aux données est exprimé en nombre d'entrées-sorties nécessaires pour l'exécution des requêtes de jointure en étoile. Pour faire une comparaison entre les coûts d'accès aux données avec et sans indexation, nous donnons dans ce qui suit les formules permettant de calculer ce coût pour chaque scénario.

a) Coût d'accès aux données sans IJB

Les jointures sont supposées être réalisées par un algorithme de hachage. C'est l'algorithme le plus utilisé pour calculer les opérations de jointures dans les SGBD commerciaux, il est simple et donne de bonnes performances dans le cas des tables dont l'une est volumineuse et l'autre est de taille nettement inférieure. Ce qui est le cas dans le contexte des entrepôts de données avec la table des faits et les tables de dimensions [11].

Le nombre d'entrées/sorties nécessaires pour joindre en hachage la table des Faits F et une table de dimension D est donné par [17] :

$$C_{JHachage} = 3(||F|| + ||D||).$$

Pour joindre plusieurs tables, l'ordre des jointures est important car le coût du résultat final dépond des résultats intermédiaires (résultat des jointures déjà calculées) qui peuvent ou non tenir en mémoire centrale. Mais dans notre cas nous avons considéré un modèle de coût simplifié qui suppose que les résultats intermédiaires tiennent en mémoire.

Le coût de jointure de la table des faits avec n tables de dimensions $\{D_1, D_2, .., D_n\}$ peut donc être donné par [17] :

$$CJ_{Hachage} = 3 \times (||F|| \times \sum_{i=1}^{n} ||D_i||)$$

b) Coût d'accès aux données en utilisant des IJB

L'accès aux données en utilisant un IJB se passe en deux étapes

1. Parcourir l'index pour rechercher les bitmaps permettant d'évaluer une requête donnée.
2. Lecture directe des données à partir du disque.

– *Coût de parcours :* Au pire des cas, tous les bitmaps doivent être parcourus pour rechercher le bitmap correspondant. Ce parcours se fait par unité de page système Soit *Sp*. Ce qui donne un coût égal à [17] : $\frac{|F|+|RowID|}{8 \times S_p}$
Maintenant, il faut lire les vecteurs de bitmaps et identifier les bits mis à 1. Pour cela, il faut parcourir chaque bitmap et répéter ce processus autant de fois que le nombre de prédicats de sélection appliqués sut l'attribut indexé soit d. Donc [17] :

$$C_{parcours} = \frac{d \times |A| \times (|F| + |RowID|)}{8 \times S_p}$$

– *Coût de lecture :* le nombre de tuples lus pour une requête utilisant d bitmaps est [17] : $N_r = \frac{d \times |F|}{|A|}$. Étant donnée ce nombre, son coût de lecture est donnée par la formule [17] :

$$C_{lecture} = ||F|| \, (1 - e^{\frac{N_r}{||F||}})$$

– Pour conclure, Le coût d'accès aux données à travers un IJB est égal à :

$$C_{accès} = \frac{d \times |A| \times (|F| + |RowID|)}{8 \times S_p} + ||F|| \, (1 - e^{\frac{N_r}{||F||}})$$

3.6 Construction d'une configuration d'index

Les motifs résultats de la phase précédente respectent les préférences de l'administrateur et peuvent être utilisés pour construire une configuration d'index. Pour cela, le schéma de l'entrepôt de données (clés étrangères de la table de faits, clés primaires des tables dimensions, etc.) est utilisé pour construire un index sur les

attributs de chaque motif.

Exemple

L'instruction de construction d'un index bitmap de jointure a la structure suivante (sous oracle) :

CREATE BITMAP INDEX IJB_Dim1_Dim2
ON Fait (Dim1.Attribut2, Dim2.Attribut2)
FROM Fait, Tab_Dim1, Tab_Dim2
WHERE Fait.Id_Dim1= Dim1.Id and Fait.Id_Dim2= Dim2.Id

Tel que :

- La close *ON* est composée des attributs constituants chaque motif et sur lesquels est construit l'index (Attributj est un attribut non clés de la table de dimensions Dimi).
- La close *FROM* est composée de la table des faits ainsi que la liste des tables des dimensions auxquelles appartiennent les attributs du motif.
- La close *WHERE* est constituée des prédicats de jointure composés des clés étrangères de la table des faits (Fait.Id_Dimi) et des clés primaires des tables de dimension correspondantes (Dimi.Id).

Chapitre IV

Implémentation de la solution et étude expérimentale

1 Introduction

Afin de valider notre approche, nous l'avons implémenté pour pouvoir l'évaluer et lui-appliquer des tests sur des données réelles. Pour cela, nous avons développé un outil nommé CIStool (Constrained Index Selection tool) qui fournit à l'administrateur une interface simple et intuitive lui permettant de spécifier des contraintes et les-appliquer sur une charge de requêtes décisionnelles pour en sortir une configuration d'index permettant d'optimiser la charge tout en respectant les préférences de l'administrateur.

Dans ce chapitre, nous allons présenter dans la première partie l'outil de sélection des index de jointure binaires réalisé et appelé CIStool. Puis, dans la deuxième partie, nous réalisons une série de tests sur un entrepôt réel issu du benchmark APB1 [79] afin d'examiner les comportements de notre approche en fonction de différentes contraintes. Pour cela, nous commençons par introduire notre environnement d'expérimentation, la charge de requête sur laquelle nous avons travaillé et nous finissons par l'évaluation des résultats obtenus.

2 CIStool : Outil de sélection d'index sous contraintes

CIStool est l'implémentation de notre démarche de sélection d'index sous contraintes. C'est un outil d'aide à l'administration et d'optimisation des entrepôts de données à travers les index de jointure binaires IJB qui peut intervenir lors de la conception physique de l'entrepôt et dans les opérations de tuning. Son interface graphique

simple et intuitive permet à l'administrateur d'exploiter son expertise exprimée sous forme de contraintes pour choisir une configuration d'IJB optimisant le temps de réponse des requêtes décisionnelles adressées au SGBD. L'outil permet aussi de donner des coûts théoriques sur la taille de l'espace de stockage nécessaire pour la configuration d'index et sur le coût d'exécution des requêtes sans et avec cette configuration d'index exprimé en nombre d'opérations d'entrées/sorties nécessaires pour la résolution des requêtes. La Figure IV.1 présente l'architecture de notre outil.

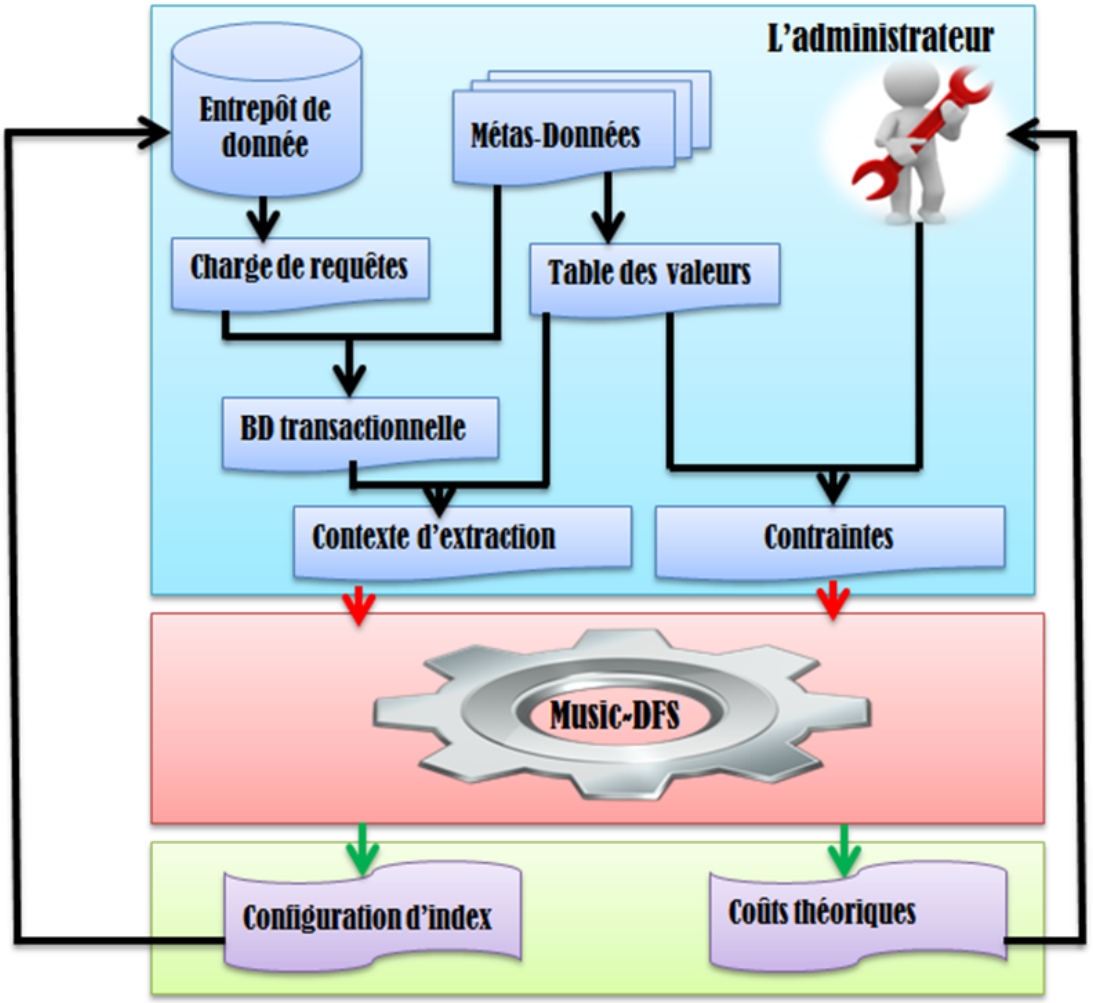

FIGURE IV.1: Architecture de l'outil CIStool

L'outil prend donc en entrée, un contexte d'extraction créé à partir d'une charge de requêtes. Et permet de :

1. Paramétrer les contraintes les plus connues dans la littérature et discutées dans le chapitre III (section 3.4.1)
2. Générer un script directement exploitable par l'algorithme d'extraction de mo-

tifs sous contrainte Music-DFS.

3. Exécuter ce script pour créer une configuration d'index respectant les contraintes de l'administrateur.
4. Donner des informations détaillées ou pas concernant le nombre d'index de la charge, le coût de stockage des index, le coût d'exécution des requêtes avec et sans IJB, le pourcentage des requêtes optimisées, ... etc
5. Visualiser les détails de l'entrepôt de donnée comme : le schéma et les propriétés des différentes tables et attributs.

Dans les sections suivantes nous présentons à travers des imprimes-écrans les fonctionnalités de CIStool.

2.1 Langage et format des fichiers

L'outil est développé sous l'Environnent de Développent Intégré Eclipse. A part sa portabilité, java est choisi pour sa gestion automatique de la mémoire. Cette caractéristique est cruciale car les motifs sont implémentés à travers des listes et des ensembles. Le langage Java offre aussi une multitude de fonctions permettant de manipuler facilement ce type de structures de données.

Les deux fichiers en entrée représentent le contexte d'extraction, ils doivent respecter les formats exigés dans l'outil Music-DFS (Annexe A). Ce sont donc des fichiers texte qui portent l'extension (.bin) pour la base de données transactionnelle et (.att) pour la table des valeurs.

La configuration d'index créé ainsi que ses détails sont sauvegardés dans un fichier texte (.txt).

2.2 Présentation des fonctionnalités

La Figure IV.2 représente l'interface d'accueil de CIStool, elle est composée de quatre onglets : *Constrained Index Selection, Schéma de l'entrepôt de donnée, Tables* et *Attributs*.

2.2.1 L'onglet : Constrained Index Selection

C'est le premier onglet et le plus important. Il permet à l'administrateur de générer une configuration d'index optimisant une charge de requêtes en trois étapes, à savoir :

- Le chargement du contexte d'extraction,
- Le paramétrage des contraintes,

• La création et l'exécution d'une requête Music-DFS.

La partie inférieure est réservée pour l'affichage des résultats.

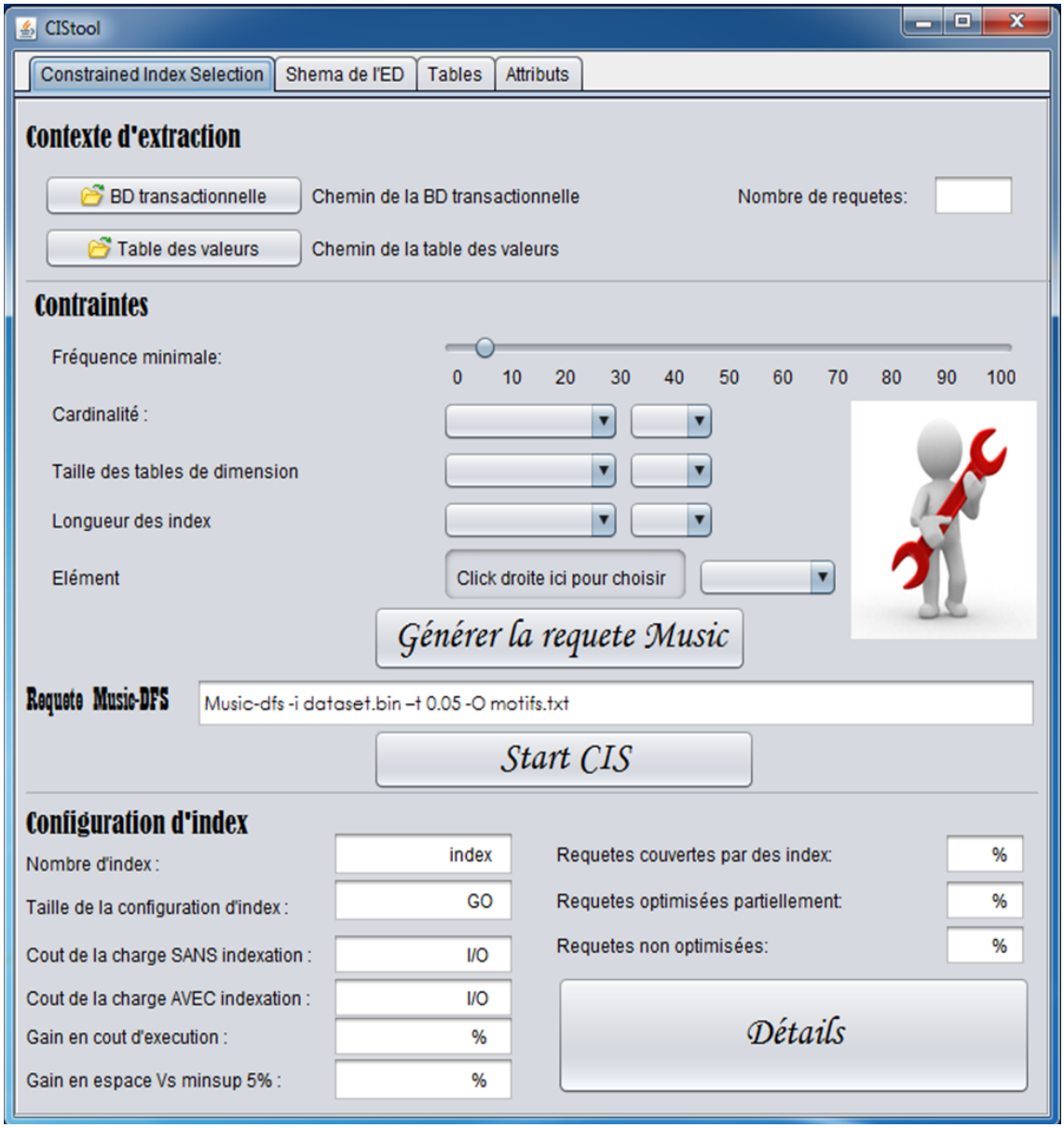

FIGURE IV.2: Interface d'accueil de l'outil CIStool

2.2.1.1 Chargement du contexte d'extraction

L'outil offre à l'administrateur la possibilité d'introduire son contexte d'extraction à travers deux boutons qui permettent de parcourir le disque dur afin de charger les deux fichiers contenant «La base de donnée transactionnelle » et « la table des valeurs ». Ces deux fichiers servent d'entrées à l'algorithme d'extraction de motifs sous contraintes Music-DFS.

2.2.1.2 Expression des contraintes

Dans la deuxième étape, l'administrateur peut librement formuler les contraintes imposées au système à travers des boites de sélection et un slider. Chaque contrainte est suivie d'une primitive permettant de la délimiter. Les contraintes implémentées sont :

- *La fréquence minimale* : implémenté à travers un slider, il permet de spécifier une valeur allant de 0% à 100% au support minimale relative « minsup ». Il est initié à 0.05% (c'est la valeur connue dans la littérature à donner de meilleurs résultats).
- *Cardinalité* : implémenté à travers une boite de sélection, elle permet de sélectionner parmi les cardinalité des attributs, un seuil (minimale/maximale) pour les cardinalité des attributs à retenir selon l'espace de stockage disponible pour lui. Un attribut ne respectant pas cette contrainte ainsi que sa spécialisation/généralisation sera élagué pendant le processus de sélection et non pas dans une deuxième phase.
- *Taille des tables de dimension* : permet de sélectionner parmi les tailles des tables de dimension, la taille (minimale/maximale) des tables à retenir.
- *Longueur des index* : permet d'imposer une longueur aux index, allant de 1 (index mono attribut) jusqu'au nombre d'attributs indexables.
- *Élément* : permet d'imposer la présence ou non d'un attribut ou d'un ensemble d'attributs dans chaque index. Il est implémenté à travers un pop-up menu permettant un choix multiple d'attributs.

Exemple *: La Figure IV.3 donne l'exemple d'une configuration où l'on a chargé les fichiers « contexte2012.bin » et « values.att » et on à exiger la création d'index de taille minimale ou égale à 4. Chaque index créé ne doit contenir que des attributs appartenant à une table de dimension contenant au moins 900 enregistrements. Chaque attribut ne doit pas avoir une cardinalité supérieur à 75. Les attributs 4,11 et 12 doivent être un sous ensemble des attributs de chaque index. Les attributs de chaque index doivent figurer ensemble dans au moins 5% des requêtes de la charge.*

2.2.1.3 Création et exécution d'une requête Music-DFS

Une fois l'administrateur à exprimer ces contraintes, le bouton « *Générer la requête Music*» permet de générer un script directement exploitable par l'algorithme de recherche de motifs sous contraintes Music-DFS.

Remarque : Le script généré est affiché dans une zone de texte, l'administrateur

peut le consulter mais aussi le modifier en rajoutant des contraintes que nous n'avons pas implémentées.

Finalement, le bouton « *Start CIS* » permet de lancer le processus de sélection d'index et d'afficher les résultats.

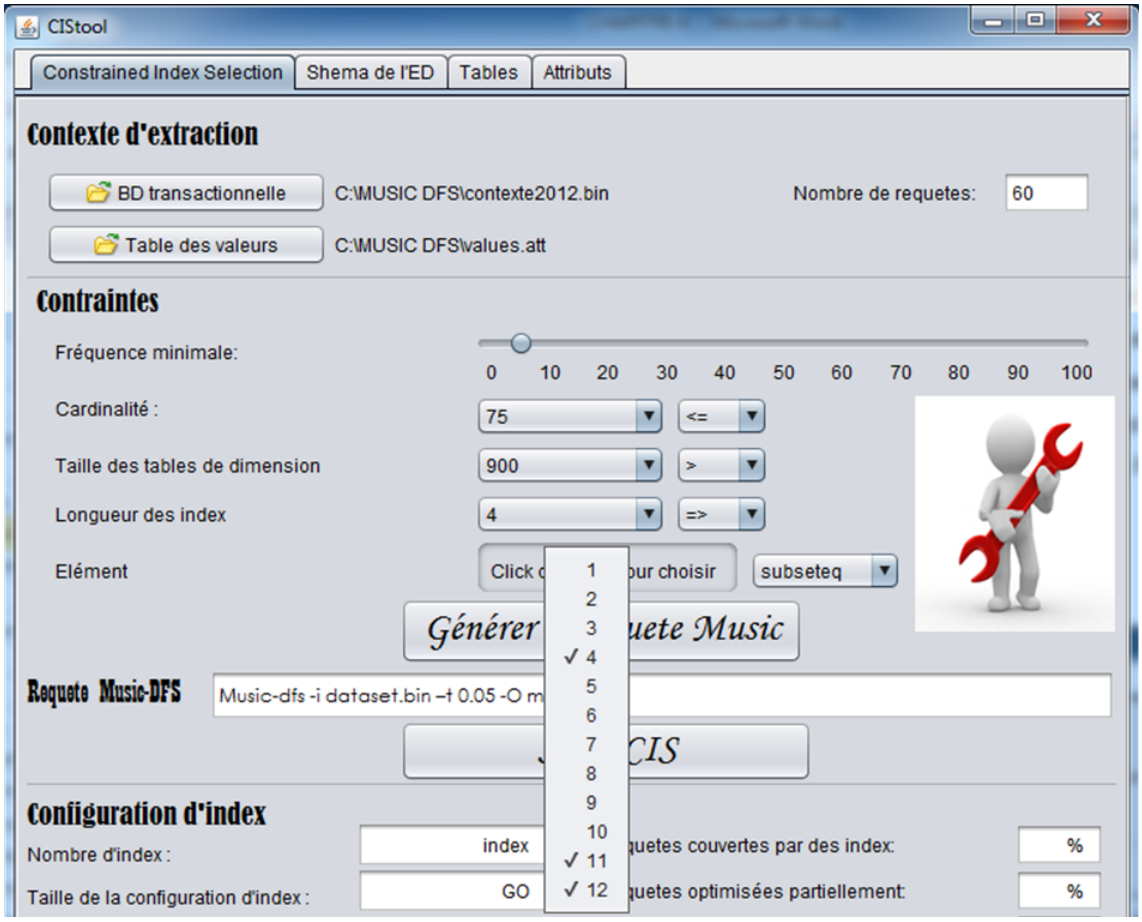

FIGURE IV.3: Exemple d'une configuration de l'outil CIStool

2.2.1.4 Affichage des résultats

Un résumé contenant le nombre d'index générés ainsi qu'une estimation de la taille de la configuration d'index et du coût de l'exécution de la charge des requêtes avec et sans la configuration d'index est affiché en bas de la fenêtre d'accueil de CIStool. Ces estimations sont basées sur le modèle de coût présenté dans le chapitre III (section 3.1.4).

D'autres informations supplémentaires sont affichées, elles concernant

– Le pourcentage de gain en coût d'exécution par rapport au coût d'une charge non optimisée.

- Le pourcentage de gain en espace de stockage par rapport à la configuration d'index obtenue avec la seule contrainte de fréquence minimale avec un minsup = 5%.
- Le pourcentage requêtes non optimisées (résolues par jointure).
- Le pourcentage de requêtes résolues par des index (un ou plusieurs).
- Le nombre de requêtes résolues partiellement par des index (quelques attributs sont indexés et le reste de la requête est résolu par jointure).

Exemple : *La Figure IV.4 représente les paramètres et les résultats d'exécution d'une requête de l'administrateur exigeant des index de longueur supérieur 2 et dont les attributs ont une cardinalité inférieur à 99 et un minsup = 0.05%. Le script Music-DFS correspondant généré est : ./music-dfs -i contexte2012.bin -a values.att -t 0.05 -q "max(x.CARD)< 99 and length (x)>2;" -o motif2012.txt -s*

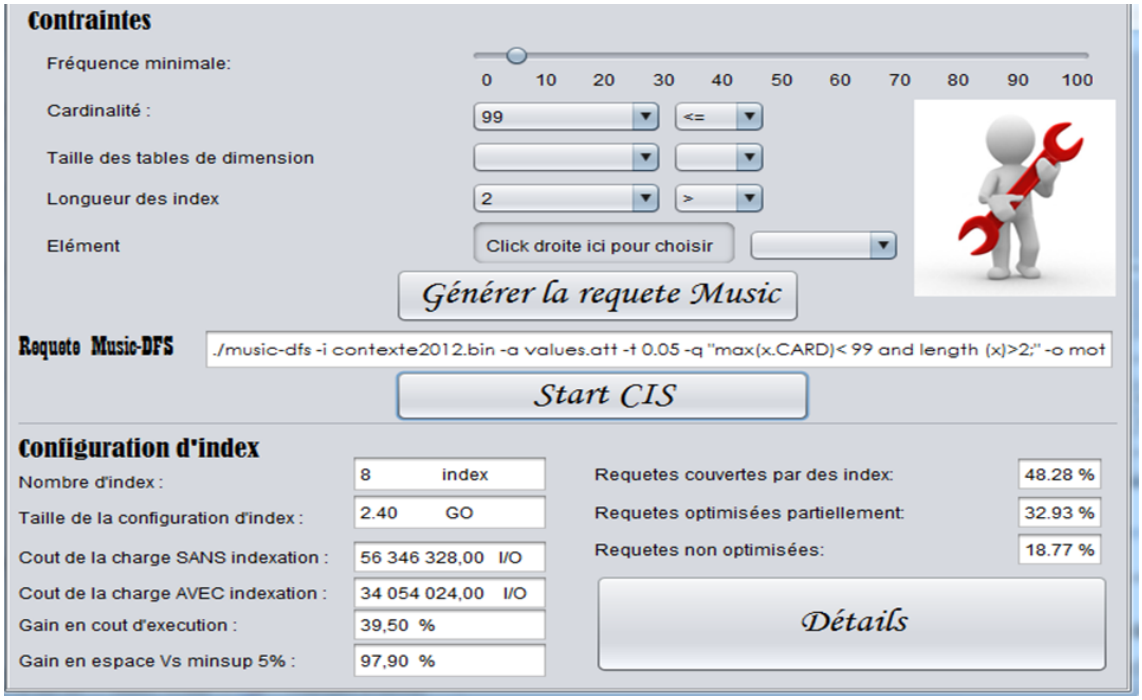

FIGURE IV.4: Génération de requête Music-DFS et affichage des résultats dans CIStool

Pour plus de détails, le bouton « *Détails* » permet d'afficher un fichier texte contenant :

1. *Pour chaque index de la configuration :*
   - Les attributs sur lesquels il est construit
   - Sa taille.

- Son coût de chargement
- Son coût d'accès.

2. *Pour chaque requête :*
   - Son coût d'exécution sans optimisation.
   - Son coût d'exécution en utilisant la configuration d'index
   - Le scénario avec lequel elle a été résolue parmi 3 scénarios :
     a) *jointure seulement :* Si aucun index n'a été utilisé pour résoudre la requête. On affiche les tables jointes.
     b) *Index seulement :* si la requête a pu être résolu en utilisant un ou plusieurs index. On affiche aussi ces index.
     c) *Jointure et index* : si la requête a été résolu en utilisant des index pour quelques attributs et des jointures pour des autres. On affiche les index utilisés et les tables jointes.
3. On affiche aussi, *le coût d'exécution de la totalité des requêtes avec et sans optimisation.*

La Figure IV.5 donne un exemple d'affichage détaillé des résultats.

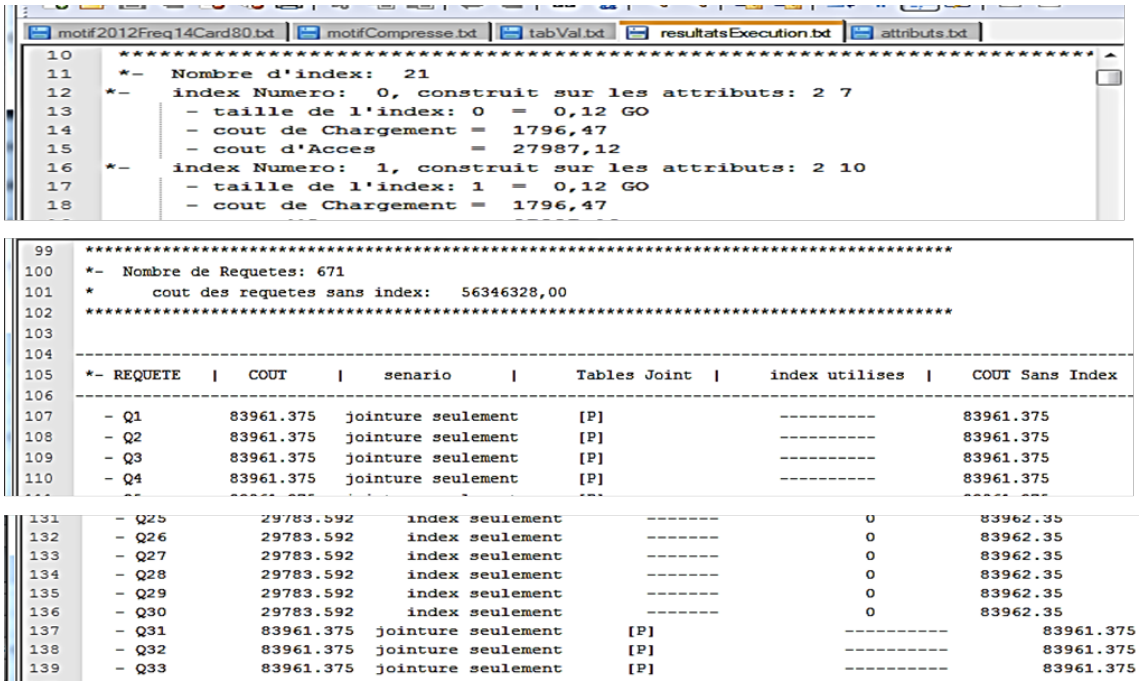

FIGURE IV.5: Résultats détaillés dans CIStool

2.2.2 Les onglets : Schéma de l'ED, Tables et Attributs

Ces trois onglets permettent à l'administrateur de charger et de visualiser le schéma logique de l'entrepôt de donnée ainsi que différents propriétés sur les tables et les attributs. L'administrateur peut revenir sur ces informations lors de la définition des contraintes. Par exemple, l'administrateur peut consulter ces informations pour connaitre les attributs qui vont être éliminés s'il définit une taille minimale pour les tabes de dimensions ou une cardinalité minimale pour les attributs.

La Figure IV.6 donne des imprimes-écrans de ces trois onglets pour l'entrepôt de donnée APB1.

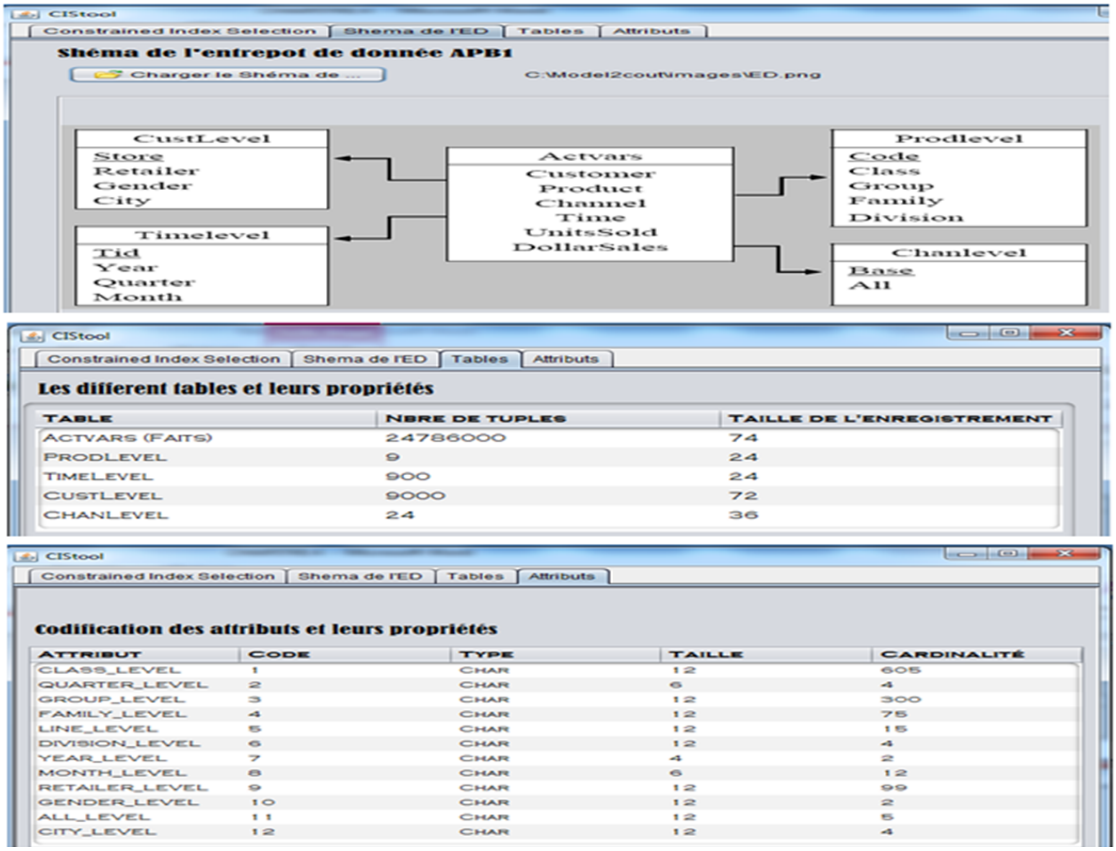

FIGURE IV.6: Les Onglets : Schéma de l'ED, Tables et Attributs

3 Expérimentations

Afin de valider notre approche, nous avons mené des expériences qui nous ont permet de l'évaluer et de la comparer avec d'autres techniques de sélection d'index de jointures binaires. Dans cette section nous présentons notre environnement d'expérimentation et nous discutons les résultats obtenus.

3.1 Environnement d'expérimentation

Les expériences que nous avons mené sous une machine Intel I3, une mémoire de 4Go et un disque dur de 500Go exploitent un entrepôt de données généré par un banc d'essai Benchmark (Data warehouse Benchmark APB-1 release II [76] et une charge de requêtes avec leurs fréquences.

Nous avons considéré que

- La taille d'une Page Système est de 65536
- L'identificateur de ligne (RowId) est codé sur 16 octets

3.1.1 Banc d'essai Benchmark

Les bancs d'essais (benchmark) permettant de mesurer les performances d'un système pour le comparer à d'autres. Dans le contexte des entrepôts de données, le plus connu est APB-1. Publié en 1998 par une association de quatre éditeurs de solutions OLAP, APB-1 simule une situation OLAP réelle de vente. Ce benchmark est utilisé par la plus part des travaux de sélection d'index ce qui nous permet de comparer notre approche par rapport à eux. Le schéma logique de l'entrepôt de donnée issu de ce benchmark est un schéma en étoile composé d'une table de faits *Actvars* et de quatre tables de dimension *ProdLevel*, *TimeLevel*, *CustLevel* et *ChanLevel*.

Le schéma en étoile de cet entrepôt est donné par la Figure IV.7.

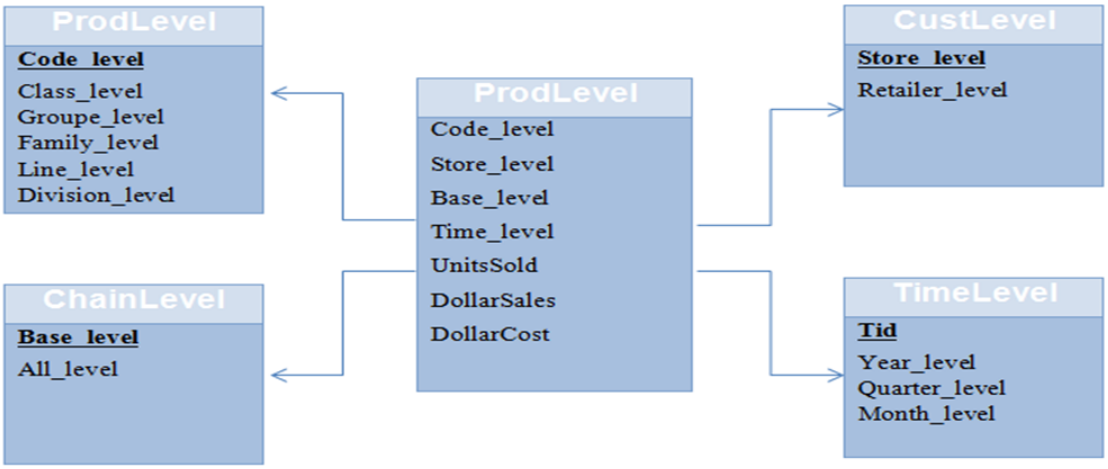

FIGURE IV.7: Schéma de l'entrepôt issu du Benchmark APB-1 realise II

Les caractéristiques des tables qui forment l'entrepôt sont données dans la Table IV.1

Table	Nombre d'enregistrements	Taille de l'enregistrement
Actvars	24 786 000	74
ProdLevel	9	24
TimeLevel	900	24
ChanLevel	9000	72
CustLevel	24	36

TABLE IV.1: Caractéristiques des tables de l'entrepôt utilisé

3.1.2 Charge de requêtes

Nous avons considéré une charge de 60 requêtes de jointure en étoile définies sur l'entrepôt ci-dessus. Ces requêtes sont choisis de tel sorte à ce qu'elles :

- respectent la syntaxe définie dans le chapitre 3 (section 3.2).
- doivent être composées d'un seul bloc et non pas des requêtes imbriquées.
- doivent couvrir l'ensemble des attributs de l'entrepôt.
- appartiennent à des catégories différentes :
  - requêtes du type count(*) avec et sans agrégations,
  - requêtes utilisant les fonctions d'agrégations comme Sum, Min, Max,
  - requêtes ayant des attributs de dimension dans la clause SELECT
  - ,...etc..

La Figure IV.8 représente un extrait de trois requêtes parmi les 60 constituants notre charge de requêtes initiale.

```
1
select Time_level,count(*)
from ACTVARS A,PRODLEVEL P
where
A.PRODUCT_LEVEL=P.CODE_LEVEL and
P.Class_LEVEL='POOHV1RICH5W'
group by Time_level
2
select line_level,sum(Dollarcost)
from ACTVARS A,PRODLEVEL P
where
A.PRODUCT_LEVEL=P.CODE_LEVEL and
P.Class_LEVEL='CI493YZ9KZUJ'
group by line_level
3
select count(*)
from ACTVARS A,PRODLEVEL P
where
A.PRODUCT_LEVEL=P.CODE_LEVEL and
P.Class_LEVEL='FDXAQ1N5U026'
```

FIGURE IV.8: Extrait de la charge des requêtes

La charge de requêtes contient 12 attributs de sélection, ces attributs représentent la liste des attributs candidats pour l'indexation.

Les cardinalité de ces attributs sont données dans la Table IV.2

Chaque requête est caractérisée par sa fréquence d'accès, la fréquence de chaque requête est donnée dans la Figure IV.9. Nous pensons que cette fréquence est un facteur important car si une requête est fréquente, ses attributs ont intérêt à être indexé

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

TABLE IV.2: Attributs de sélection et leurs cardinalités

plus que les attributs d'une requête non fréquente. Donc, lors de la construction du contexte d'extraction nous devons prendre en considération cette fréquence qui est différente de la fréquence d'apparition des attributs dans les 60 requêtes considérées. Pour cela, nous avons répété chaque requête autant de fois que sa fréquence, ce qui donne au final un contexte avec 671 entrées. Ainsi, on pénalise les requêtes moins fréquentes et on donne plus d'intérêt aux requêtes fréquentes.

Requête	Fréquence	Requête	Fréquence	Requête	Fréquence	Requête	Fréquence
Q1	3	Q16	7	Q31	12	Q46	5
Q2	7	Q17	10	Q32	4	Q47	21
Q3	10	Q18	10	Q33	12	Q48	12
Q4	10	Q19	20	Q34	4	Q49	22
Q5	12	Q20	4	Q35	21	Q50	10
Q6	3	Q21	12	Q36	3	Q51	15
Q7	3	Q22	4	Q37	15	Q52	30
Q8	16	Q23	21	Q38	10	Q53	5
Q9	4	Q24	21	Q39	3	Q54	20
Q10	12	Q25	21	Q40	12	Q55	18
Q11	4	Q26	3	Q41	12	Q56	3
Q12	21	Q27	7	Q42	10	Q57	4
Q13	21	Q28	10	Q43	18	Q58	10
Q14	21	Q29	10	Q44	11	Q59	2
Q15	3	Q30	20	Q45	12	Q60	5

FIGURE IV.9: Fréquences d'accès des requêtes de la charge

3.2 Tests et évaluation des résultats

Afin de montrer l'effet de l'intégration des contraintes dans le processus de sélection d'index, nous avons effectué plusieurs expériences qui se basent sur l'évaluation

théorique du coût d'exécution des requêtes en présence des index sélectionnés en utilisant chaque type de contrainte et en combinant quelques-unes. Ce modèle de coût théorique a été présenté dans le chapitre III (section 3.6.1).

L'étude expérimentale a donc comme objectif de :

- Montrer l'intérêt d'optimiser une charge de requêtes par index de jointure binaires choisis en intégrant des contraintes sur eux au processus de sélection.
- Comparer les performances de chaque type de contrainte.
- Montrer l'effet de l'expertise de l'administrateur sur le bon choix des index à créer.
- Comparer les performances de notre approche avec des travaux étroitement liés .

3.2.1 Déroulement des expérimentations

Nos expérimentations se sont déroulées en trois étapes :

1. *Dans la première expérience*, et dans le but de montrer l'effet de la contrainte de fréquence minimale et aussi pour choisir la meilleure configuration qui va subir d'autres contraintes, nous avons varié la valeur du support minimal minsup pour la contrainte de fréquence minimale et calculé le coût d'exécution des requêtes de la charge ainsi que sa taille.
2. *Dans la deuxième série d'expériences*, nous avons fixé la valeur de *minsup* choisie dans la première étape et nous avons rajouté les autres contraintes une par une et essayé de trouver une configuration garantissant un bon compromis entre performance et espace de stockage. Ces expériences ont pour objectif de montrer l'effet des contraintes de la cardinalité des attributs, la taille des tables de dimension et la taille (longueur) des index dans la sélection des index.
3. *Dans la troisième série d'expériences*, nous avons combiné plusieurs contraintes dans le but de voir le comportement de notre approche dans ce cas. Nous avons aussi effectué une comparaison de nos résultats avec ceux obtenus par l'approche de recherche de motifs maximaux.

3.2.2 Effet de la contrainte de fréquence minimale :

La Figure IV.10 montre la variation du coût d'exécution de la charge des requêtes en fonction de la valeur de *minsup*. Nous avons un gain maximal de 46.70% pour un *minsup* de 5%. Cela est dû aux index créés qui évitent des jointures coûteuses entre la table des faits et les tables de dimension. Cependant ce gain diminue jusqu'à ce que *minsup* atteigne 45% où il n'y a plus de gain. Cela est dû au fait que le nombre

d'index diminue lorsque le *minsup* augmente (Figure IV.11), et par conséquent les requêtes utilisant des attributs non indexés sont obligé de faire des jointures entre la table des faits et les tables de dimension de ces attributs ce qui augmente le coût.

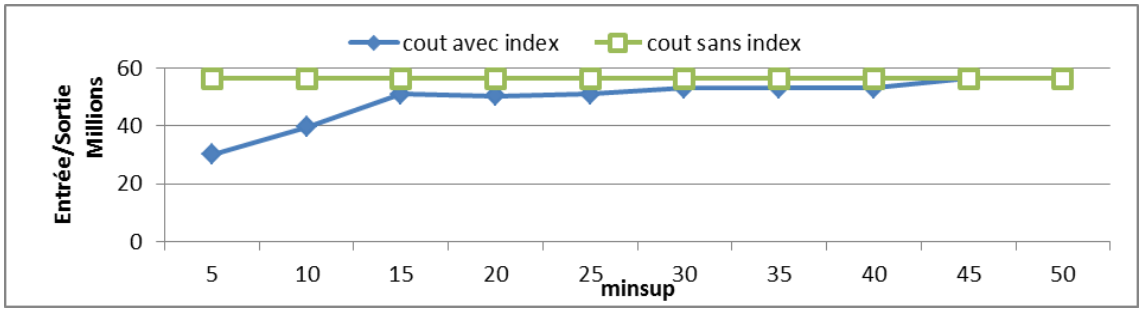

FIGURE IV.10: Effet de la contrainte de fréquence minimale sur le coût d'exécution de la charge de requêtes

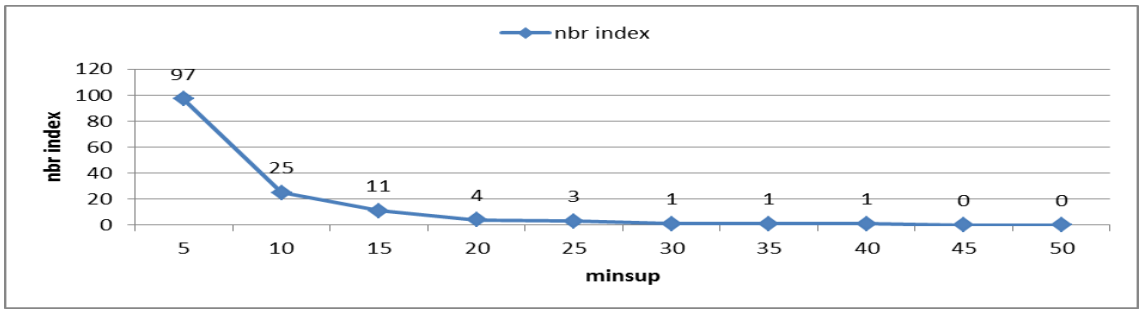

FIGURE IV.11: Nombre d'index Vs minsup

Mais en contrepartie, la taille de la configuration d'index pour la meilleure configuration est de 123.09 GO (Figure IV.12) et elle diminue quand le support minimal augmente. Cela est évident à cause de la diminution du nombre d'index générés.

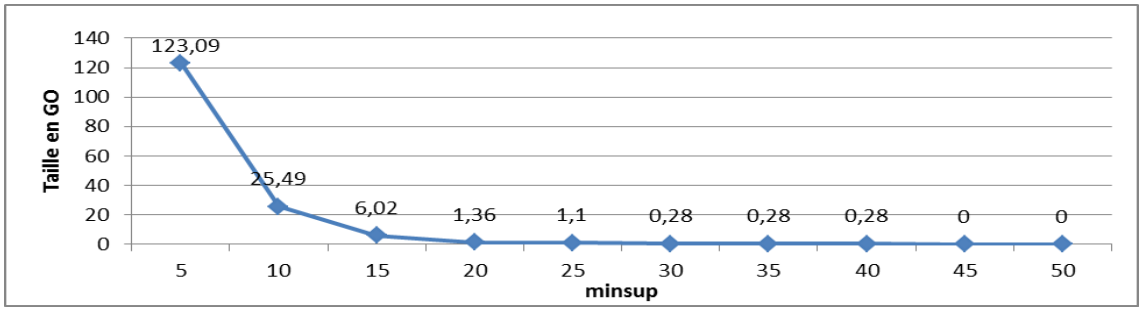

FIGURE IV.12: Taille de la configuration d'index Vs minsup

En effet, lorsque *minsup* est petit nous avons un grand nombre de motifs qui sont considérés fréquents et donc le même nombre d'index est créé ce qui augmente la

taille de la configuration d'index.

Nous pouvons voir que pour un *minsup* de 5%
- Nous avons la meilleure performance en matière de coût d'exécution.
- Un large nombre de motifs est généré ce qui nous permet d'appliquer d'autres contraintes sur eux.

Pour cela, nous allons fixer la valeur de *minsup* à 5% pour nos prochaines expérimentations. Aussi, c'est cette valeur qui a été retenu par les approches de datamining avec lesquelles nous allons faire une comparaison.

3.2.3 Effet de la contrainte de cardinalité des attributs

La taille d'un index de jointure binaire dépend principalement de la cardinalité des attributs sur lesquels il est construit. Pour cela et afin de diminuer la taille de la configuration d'index, l'administrateur peut imposer à l'algorithme de sélection sous contrainte de ne pas considérer les motifs contenant des attributs de forte cardinalité ainsi que leurs sur-ensembles.

En analysant les cardinalité des attributs indexables (Table IV.2) l'administrateur peut s'apercevoir qu'il sera intéressant d'écarter les attributs *ClassLevel*, *GroupLevel*, *RetailerLevel* ayant comme cardinalité, respectivement, 605, 300 et 99.

La Figure IV.13 représente le paramétrage de l'outil CIStool correspondant à ce type de contraintes et la requête Music-Dfs équivalente est : « *./music-dfs -i contexte2012.bin -a values.att -t 0.05 -q "Max(X.CARD)< 99;" -o motif2012.txt –s* » Où : *contexte2012.bin* est la base de donnée transactionnelle, *Values.att* est la table des valeurs et *motifs2012.txt* est le fichier de sortie (Output).

Les résultats montrent qu'avec un minsup de 5% et en écartant les attributs de cardinalité supérieur ou égale à 99, nous avons pu avoir un gain en coût d'exécution de 49.55 % qui est supérieur à celui utilisant la contrainte de fréquence seule (46.70%). Cela peut être expliqué par le fait que l'élimination des attributs de forte cardinalité diminue les tailles des index et du fait le coût de parcours de ces index (voir Chapitre III section 3.1.4.2) et donc le coût des requêtes.

Nous avons pu avoir aussi un gain considérable (93,01%) en espace de stockage qui a passé de 123.09 Go à 8,6 Go. Cela est dû à l'élimination des attributs de forte cardinalité mais aussi à la réduction du nombre d'index qui a passé de 97 à 30 index.

Nous pouvons conclure que la cardinalité des attributs est un paramètre très

important pour la sélection d'une configuration d'index et doit être pris en considération que ça soit pour diminuer l'espace de stockage ou pour garantir une meilleure performance.

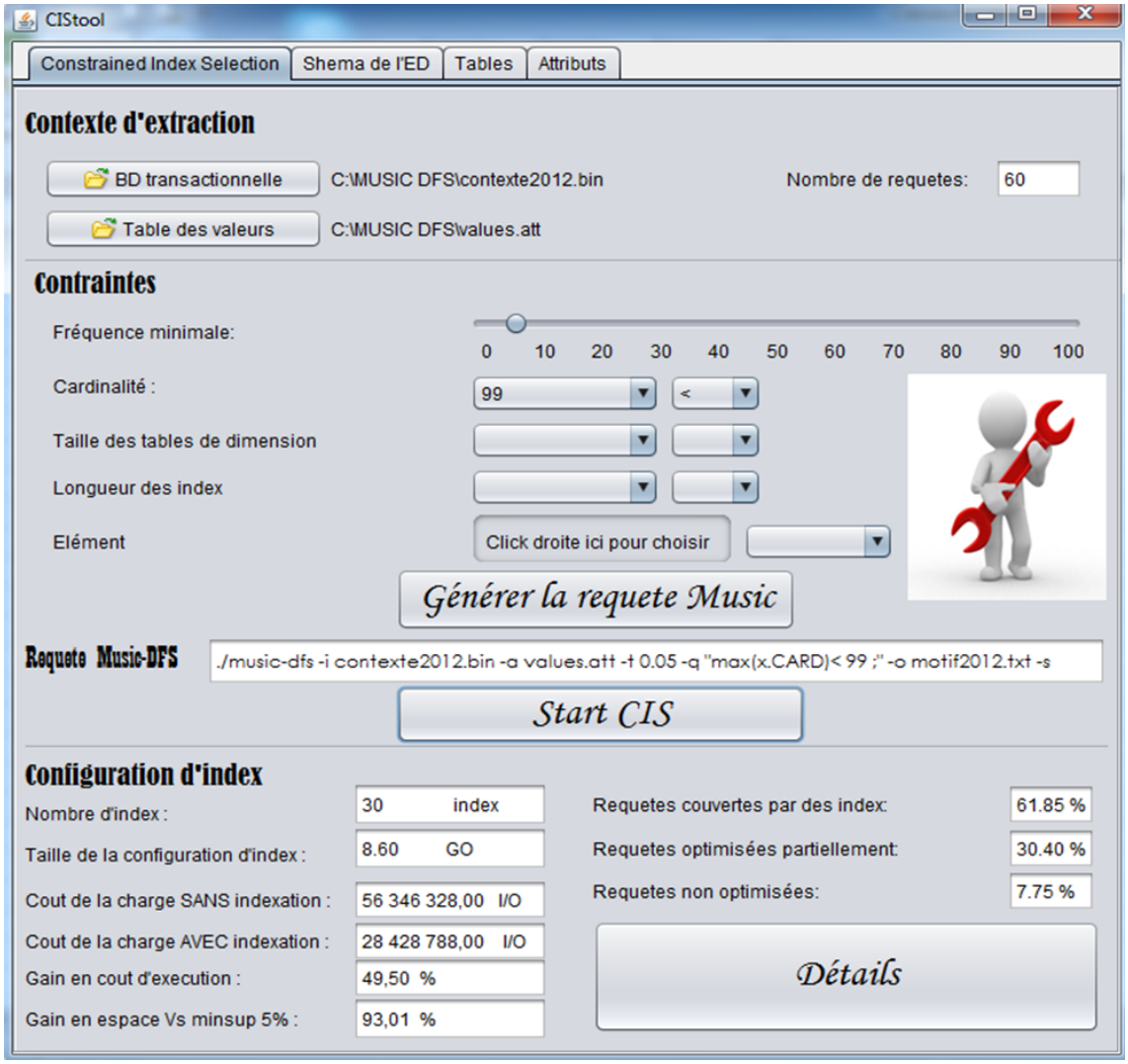

FIGURE IV.13: Effet de la contrainte de cardinalité

3.2.4 Effet de la contrainte de taille des tables de dimensions

Le coût des jointures entre la table des faits et des tables de dimensions volumineuses est important par rapport au coût des jointures avec des tables de dimensions de faible taille. Donc, toujours dans le but de réduire l'espace de stockage en éliminant des attributs, nous avons mené une expérience permettent d'évaluer le coût de la charge de requêtes après élimination des attributs indexables appartenant aux tables de dimension les moins volumineuses.

En constatant le nombre de tuples de chaque table de dimension nous avons remarqué que les tables de dimension *ProdLevel* et *ChanLevel* dont le nombre de tuples est respectivement 9 et 24 ont un nombre de tuples nettement inférieur aux autres tables, donc nous avons paramétré l'outil CIStool afin de générer des index construits sur des attributs n'appartenant pas à ces deux tables.

La Figure IV.14 montre les résultats de l'expérience.

FIGURE IV.14: Effet de la contrainte de la taille des tables de dimension

Les résultats montrent une réduction de 17.99 % dans le coût d'exécution des requêtes, cette réduction n'est pas assez importante comparée à celles des deux expériences précédentes. Nous expliquons ces résultats par le fait que le nombre d'index générés est assez petit (9 indexes) mais surtout au fait que 50% des attributs indexables appartiennent à ces deux tables écartées. En effet, le nombre important d'attributs écartés a influé sur le taux des requêtes optimisées, tel que 53.50% des requêtes ont été résolues par jointure et 31% résolu partiellement par ces index (une partie résolue par index et l'autre par jointure), ce qui a influé négativement sur la qualité de la solution. Mais en contrepartie, une réduction importante (97,11%) de l'espace de stockage est obtenue. Cela s'explique aussi par le nombre réduit d'index.

3.2.5 Effet de la contrainte de longueur des index

Dans cette série d'expérimentations nous avons essayé de voir le comportement de notre approche vis-à-vis la longueur des index (nombre d'attributs sur lesquels il

est construit).

Pour cela, nous avons varié la valeur de la contrainte «*longueur des index* » dans l'outil CIStool pour avoir dans un premier temps des index mono-attribut et ensuite des index de longueur de plus en plus grande.

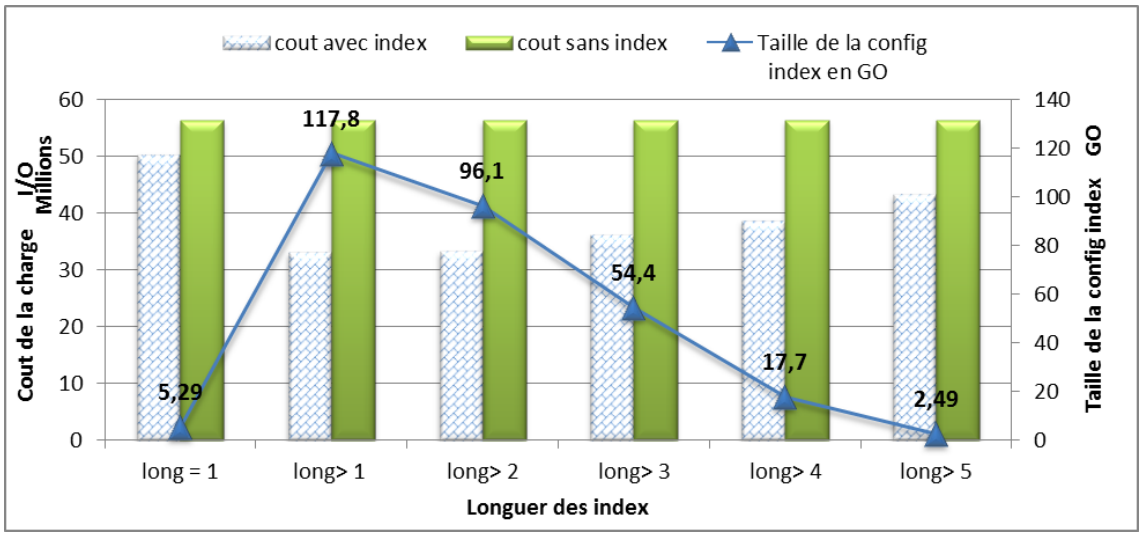

FIGURE IV.15: Coût et taille de la charge des requêtes Vs Longueur des index

Les résultats présentés dans le Figure IV.15 montrent que les index mono-attribut donnent une configuration d'index de taille 5.29 GO soit une réduction de 95,70 dans le coût de stockage ce qui est appréciable, mais ils donnent le plus mauvais résultat en matière de coût d'exécution et la charge des requêtes n'a été amélioré que de 10.76%.

Pour comprendre cette augmentation du coût, nous avons consulté le fichier contenant les résultats détaillés de l'outil CIStool pour voir quelles requêtes sont responsables de cette augmentation. Nous avons remarqué que le coût des requêtes qui portent sur un grand nombre d'attributs est très élevé par rapport au coût de jointure des tables inclus dans ces requêtes donc l'utilisation des index a un effet négatif. Cela peut être expliqué par :

1. L'augmentation du coût du parcours total des index pour rechercher le bitmap correspondant (voir Chapitre III section 3.1.4.2) à cause de la présence d'une colonne supplémentaire pour chaque index constituant l'identifiant du tuple ou RowID (Row Identifier) nécessitant 16 octets supplémentaires (taille d'un RowID dans l'SGBD Oracle et valeur choisis pour nos expérimentations). En multipliant la taille de cette colonne par le nombre de tuples de la table des faits qui est assez important, cela augmente considérablement le nombre de pages systèmes chargées et du fait le coût d'exécution des requêtes.

2. Le coût de lecture des tuples est cumulé autant de fois que le nombre d'index utilisés pour répondre à une requête.

En éliminant les index mono-attribut et en augmentant à chaque fois la longueur des index à retenir nous avons remarqué une réduction dans l'espace de stockage mais aussi une diminution dans le gain sur le coût d'exécution (par rapport aux résultats obtenu avec la contrainte de fréquence minimale seule). Cela est dû à la réduction du nombre d'index qui influe positivement sur la taille de l'espace de stockage mais négativement sur le coût d'exécution.

Nous pouvons conclure que les index multi-attributs représentent un enjeu important dans le problème de sélection d'index de jointure binaire, ils sont plus bénéfiques que les index mono-attribut et permettent une amélioration non négligeable (par exemple, le cas des index dont la longueur est supérieure à 4 permet un gain en espace de stockage de 85.62% tout en présentant un gain en coût d'exécution de 31.36%).

3.2.6 Effet de la combinaison de contraintes et étude comparative.

Dans cette série d'expérimentations, nous avons testé les performances de notre approche en combinant les contraintes précédentes et nous avons comparé les résultats obtenus avec ceux de l'approche de sélection d'IJB en utilisant l'approche de recherche de motifs fréquents maximaux de Ziani et al [47]. Les Figure IV.16 et IV.17 montrent respectivement le gain en coût d'exécution de l'ensemble des requêtes en utilisant la configuration d'index de jointure binaire issue de nos expérimentations et la taille de de cette configuration.

Nous pouvons faire les remarques suivantes :

1. En rajoutant la contrainte de longueur d'index à la contrainte de cardinalité minimale nous avons constaté une diminution dans le gain en coût d'exécution qui a passé de 49.55% à 39.50%. Cela peut s'expliqué par l'élimination des index dont la taille est inférieure à 4 ce qui influe sur la qualité de la solution. Mais, en contrepartie le gain en espace de stockage est énorme où nous avons pu avoir une configuration de 0.31 GO.
   Nous pouvons conclure que si l'administrateur de l'entrepôt de donnée dispose d'une taille limité pour le stockage des index, la combinaison de ces deux contraintes peut être intéressante pour avoir un bon compromis *Gain en coût d'exécution/Espace de stockage*.

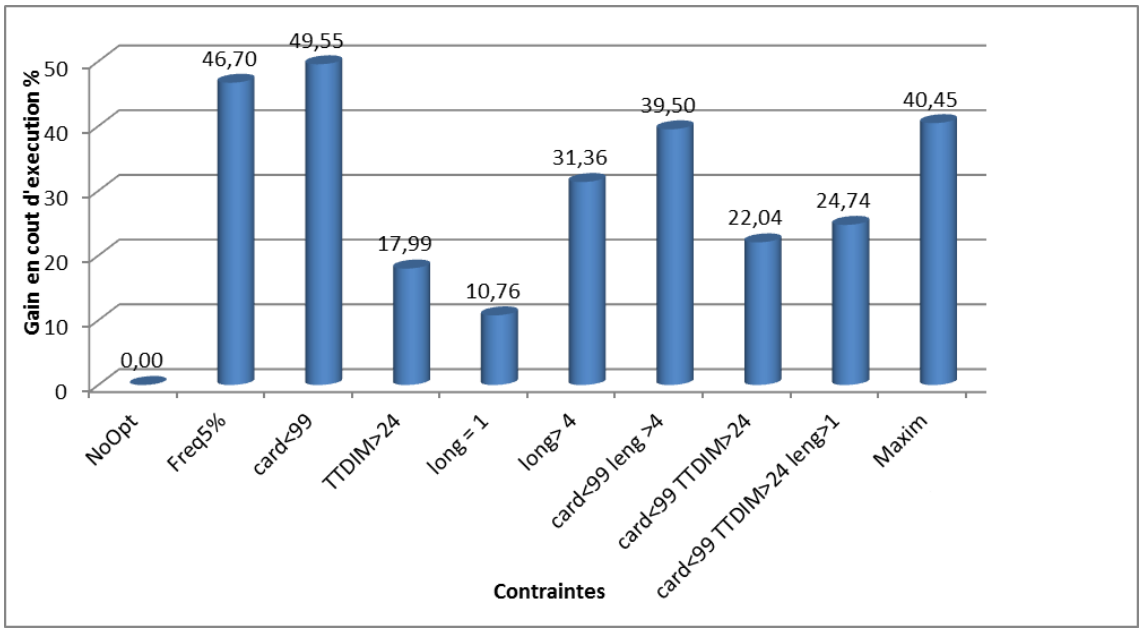

FIGURE IV.16: Taux d'amélioration dans le coût d'exécution des requêtes Vs Contraintes

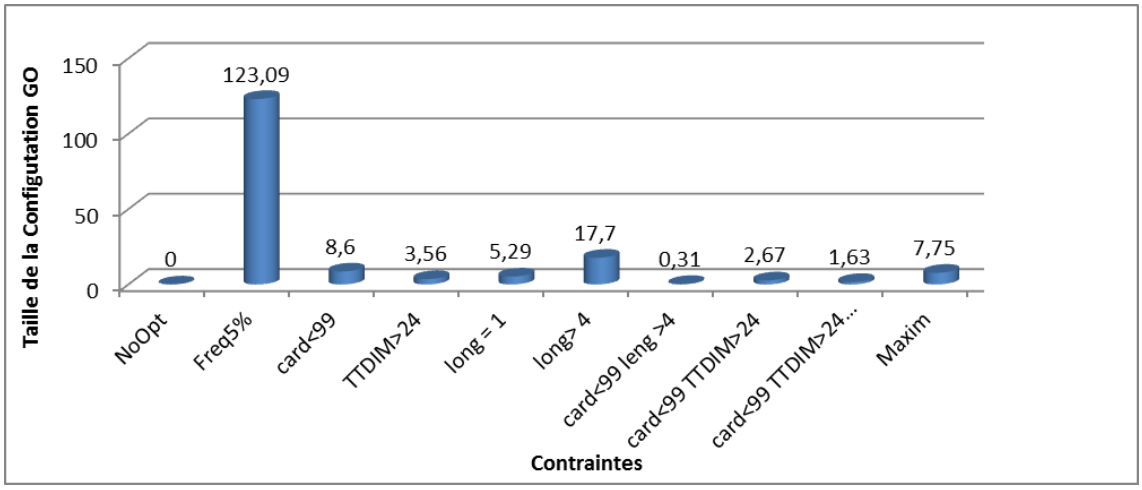

FIGURE IV.17: Taille de la configuration index Vs Contraintes

2. En rajoutant la contrainte de taille des tables de dimensions à la contrainte de cardinalité minimale, le résultat n'est pas assez appréciable que les autres résultats. Cela est dû à l'élimination des attributs appartenant aux tables de dimension les moins volumineuses sans tenir compte d'autres paramètres comme leur nombre (dans notre cas, ces attributs représentent 50% des attributs indexables), ou leurs fréquences d'apparition dans les requêtes de la charge. Cette même remarque est observée dans l'expérimentation où nous avons combiné les quatre contraintes (fréquence, cardinalité, taille des tables de dimension et longueur des index) où la contrainte de taille des tables de dimension a eu un effet négatif sur la qualité de la solution.

   Nous pouvons conclure que pour garantir une meilleure performance lors de la sélection d'une configuration d'index, l'utilisation de la contrainte qui porte sur l'élimination des attributs appartenant aux tables de dimensions les moins volumineuses doit être faite avec précaution et en considérant d'autres paramètres cités précédemment (nombre, fréquence,...).

3. L'approche utilisant les motifs fréquents maximaux donne aussi de bons résultats et un bon compromis *Gain en coût d'exécution / Espace de stockage* (40.45% de gain pour 7.75 GO). Cependant, notre approche et en combinant les trois contraintes : *fréquence des attributs*, leurs *cardinalités* et *la longueur des index* a pu donner des résultats assez proches en matière de cout d'exécution (39.50%) mais seulement pour une configuration d'index de taille de (0.31 GO) et donc un gain important en espace de stockage de (7.44 GO). Notre approche a pu aussi donner de meilleurs résultats (49.55% de gain) mais avec plus d'espace de stockage (8.6 GO) et ce dans le scénario où nous avons procédé à l'élimination des attributs de forte cardinalité.

Conclusion et perspectives

Le travail développé dans ce mémoire s'inscrit dans le cadre de l'optimisation de la conception physique des entrepôts de données relationnels modélisés selon un schéma en étoile. Vu la taille gigantesque de ces entrepôts et la complexité des requêtes dites *requêtes de jointure en étoile* qui les interroge, de nombreuses techniques d'optimisation ont été proposées dans la littérature dont les principales sont les vues matérialisées, la création d'index et la fragmentation. Parmi les techniques d'indexation, les index de jointure binaires ont montré leur efficacité dans l'optimisation de ces requêtes. Par contre, la sélection des index à créer est un problème NP-complet.

Dans ce contexte, nous avons proposé une approche de sélection d'index de jointure binaires basée sur « *L'extraction de motifs sous contraintes*». L'apport de notre travail est la possibilité de cibler le processus de sélection des index suivant les centres d'intérêts de l'administrateur de l'entrepôt de donnée. Notre contribution consiste donc à exploiter l'expertise de l'administrateur de manière interactive en exprimant son intérêt sous forme de contraintes que l'extracteur de motifs doit pouvoir exploiter pendant le processus de sélection. Le but est de ne générer qu'un ensemble d'index apriori pertinents ; diminuer le temps d'extraction en élaguant le plus tôt possible l'espace de recherche (des fois cet élagage est nécessaire pour rendre le processus de sélection possible) et éviter la phase de sélection de la configuration finale d'index dans laquelle un élagage supplémentaire est appliqué car ce dernier est appliqué pendant la première phase d'extraction.

Nous avons implémenté notre approche sous forme d'un outil d'aide à l'administration que nous avons nommé *CIStool (Constrained Index Selection tool)* permettant à l'administrateur d'exprimer facilement les contraintes étudiées dans la littérature pour les intégrer au processus de sélection d'index, à savoir : la fréquence des attributs, leurs cardinalités, la taille des tables de dimensions auxquelles ils appartiennent, la longueur des index à créer et la présence ou non de certains attributs. Un modèle de coût mathématique est utilisé afin d'estimer la qualité de la solution

en matière de coût d'exécution (nombre d'entrée/sortie) et en coût de stockage ce qui permet à l'administrateur de revoir ces choix sur les contraintes appliquées.

Enfin, nous avons expérimenté notre approche à travers cet outil sur un entrepôt de données test. Dans ces expérimentations nous avons testé séparément l'effet de chacune des contraintes citées précédemment sur la qualité des index choisis et nous avons combiné quelques contraintes. Les résultats obtenus ont montré l'intérêt de l'utilisation des contraintes précédentes dans la création d'une configuration d'index qui améliore le coût d'exécution des requêtes tout en restant dans une taille de stockage acceptable. Les performances de notre approche ont atteint un taux de réduction de 49.55% du nombre d'opérations d'Entrées/Sorties par rapport à une exécution sans index et une réduction de 98% dans l'espace de stockage par rapport à une solution utilisant une technique de datamining sans contraintes.

De plus, nous avons comparé les résultats de notre approche avec ceux de l'approche basée sur la recherche des motifs fréquents maximaux qui a montré ses performances dans le problème de sélection d'index, et nous avons démontré que notre approche avec les bons paramètres peut apporter de meilleurs résultats.

Ces tests nous ont permis aussi de faire le bilan suivant :

- La cardinalité des attributs est un critère important, elle influe que ce soit sur la taille de la configuration d'index que sur le coût d'exécution des requêtes.
- Les index multi-attributs représentent un enjeu important dans le problème de sélection des IJB, ils sont plus bénéfiques que les index mono-attribut et permettent une amélioration non négligeable.
- L'élimination des attributs appartenant à de petites tables de dimension est bénéfique, cependant, elle doit être utilisée sous contrainte que ces tables ne contiennent pas un grand nombre d'attributs pour une bonne performance.
- L'expertise de l'administrateur et ces choix sur les contraintes ont un effet important sur les résultats obtenus. Il serait donc judicieux de l'intégrer au processus de sélection.
- L'intégration de contraintes au processus de sélection a permet donc en une seule étape de donner des résultats comparables (et même mieux) à ceux faisons appel à des algorithmes glouton et des heuristiques dans une deuxième étape pour réduire le nombre d'index candidats.
- Le paradigme de recherche de motifs sous contraintes laisse la porte largement ouverte pour le teste et l'intégration de nouvelles contraintes pouvant améliorer les solutions obtenues.

Perspectives

Différentes perspectives sont envisagées :

- En premier lieu, notons que dans notre approche nous n'avons pas considéré la contrainte d'espace de stockage réservé aux index. L'administrateur peut contrôler cette contrainte à travers ses contraintes, mais comme perspective on peut envisager d'intégrer cette contrainte. En effet, les contraintes définies dans ce mémoire sont dites *locales*, car elles peuvent se vérifier isolément sur chacun des motifs. Or la vérification de la contrainte d'espace nécessite de comparer plusieurs motifs entre eux. Ce type de contraintes est appelé *contrainte globale* [80] et sa vérification pose un problème car elle met en relation plusieurs motifs. Des travaux récents s'intéressent à ce type de contraintes [80][81] et nous envisageons s'inspirer d'eux pour permettre à notre approche de prendre en considération l'extraction de motifs sous contraintes globales.

- Une deuxième alternative serait de diminuer le nombre d'index pendant le processus de sélection en les comparants entre eux selon une mesure d'intérêt de l'administrateur (un ratio coût d'exécution/taille de stockage par exemple) et n'en conserver que les meilleurs. Cela rentre aussi dans le cadre de l'*extraction de motifs sous contraintes globales*, mais représente un cas particulier appelé « *top-k motifs* » où le but est de trouver les k motifs les plus significatifs au regard d'un critère choisi par l'utilisateur.

- La troisième perspective est liée à la possibilité de généraliser notre approche pour prendre en considération la sélection combinée des index de jointure binaires et d'autres techniques d'optimisation comme les vues matérialisées et la fragmentation. On peut par exemple imposer comme contrainte, la création des index sur des attributs qui n'ont pas été utilisé par d'autres techniques. La contrainte « *élément* » que nous avons implémenté dans l'outil CIStool permet ce type de traitement, reste à choisir les attributs à inclure ou à exclure.

- Pour l'outil CIStool que nous avons développé, les améliorations suivantes sont possibles :
  - L'ajout d'un module d'analyse syntaxique de la charge des requêtes permettant la création automatique du contexte d'extraction.
  - L'ajout d'un module permettant la lecture des métadonnées d'un entrepôt de donnée et en extraire automatiquement le schéma de l'entrepôt et les carac-

téristiques des tables et des attributs. Ce traitement est fait manuellement dans la version actuelle de l'outil.
  - Un autre module permettant la création effective des index est aussi envisageable.

- Dans la partie expérimentale, une validation de notre approche sous un SGBD tel qu'Oracle est à envisager.

Références et bibliographie

[1] W.H. Inmon, «Building the Data Warehouse». *John Wiley & Sons, troisième édition*, 2002.

[2] Z. Ouaret « Modélisation conceptuelle et logique d'un entrepôt de données XML », *mémoire de magister, école nationale supérieure d'informatique,* 2008

[3] O. Teste, «Modélisation et manipulation d'entrepôts de données complexes et historisées ». *Thèse de Doctorat en Informatique, Université Paul Sabatier*. 2000.

[4] A. Bouzeghoub, « Modélisation des Entrepôts de données XML : Application au domaine de la sécurité social », *mémoire de magister, Institut National de formation en Informatique (INI)*, 2008

[5] S. Anahory et D. Murray «Data Warehousing In the Real World» ; *Pearson* ; 1997

[6] R. Kimball. « Entrepôts de données, Guide pratique du concepteur de datawarehouse ». *John Wiley and Sons, Inc.*, 1996.

[7] E. Ziyati, « Optimisation de requêtes OLAP en Entrepôts de Données Approche basée sur la fragmentation génétique», », *thèse de doctorat, Université Mohammed V –AGDAL, Maroc* ; 2010

[8] S. Khouri, « Modélisation conceptuelle à base ontologique d'un entrepôt de données », *mémoire de magister, école nationale supérieure d'informatique*, 2009

[9] M. Encinas, « Entrepôts de données pour l'aide à la décision médicale : conception et expérimentation ». *Thèse de doctorat, Université Joseph Fourier.* 2005.

[10] L. Bellatreche, « 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. :

[11] K. Boukhalfa, « De la conception physique aux outils d'administration et de tuning des entrepôts de données », *Thèse de doctorat, l'école nationale supérieure de mécanique et d'aérotechnique*, 2009

[12] L. Bellatreche, R. Missaoui, H. Necir et H. Drias, "A Data Mining Approach for Selecting Bitmap Join Indices", *Journal of Computing Science and Engineering, vol. 2(1)*, January, 2008, pp. 206-223

[13] R. Bouchakri, L. Bellatreche et K. Boukhalfa, « Administration et Tuning des Entrepôts de Données : Optimisation par Index de Jointure Binaires et Fragmentation Horizontale ». *Doctoriales STIC'09*, Msila, Décembre 2009.

[14] R. Bouchakri, « Une approche dirigée par la classification des attributs pour fragmenter et indexer des entrepôts de données », *mémoire de magister, école nationale supérieure d'informatique*, 2009

[15] A. Sanjay, V. R. Narasayya, et B. Yang. «Integrating vertical and horizontal partitioning into automated physical database design. », *Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 359–370*, June 2004.

[16] L. Bellatreche, « Techniques d'optimisation des requêtes dans les data warehouses », *Sixth International Symposium on Programming and Systems,* 2003, pp. 81-98

[17] 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.

[18] J. Darmont, « Optimisation et évaluation de performance pour l'aide à la conception et à l'administration des entrepôts de

données complexes » , *Habilitation à Diriger des Recherches, Université Lumière Lyon 2*, 2006

[19] M. Barr, « Approche dirigée par les fourmis pour la fragmentation horizontale des entrepôts de données relationnels», *mémoire de magister, école nationale supérieure d'informatique*, 2009

[20] M. Bouakkaz, Y. Ouinten et B. Ziani. «Vertical Fragmentation of Data warehouses using the FP-Max Algorithm », *International Conference on Innovations in Information Technology* (IIT) pp 273–276, 2012.

[21] H. Gupta. « Selection and maintenance of views in a data warehouse. » *Ph.d. thesis, Stanford University*, September 1999.

[22] M. Teschke ; A. Ulbrich-vom Ende ; « Using Materialized Views to Speed Up Data Warehousing » , *Technical Report, IMMD 6 Universität Erlangen-Nürnberg,* 1997.

[23] Gray, J. ; Bosworth, A. ; Layman, A. ; Pirahesh, H. : « Data Cube : A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals », *in : Proc. of the 12th IEEE Int. Conf. on Data Engineering,* 1996

[24] V. Harinarayan, A. Rajaraman, et J. Ullman. « Implementing data cubes efficiently ». *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 205-216, June 1996.

[25] T. Johnson. « Performance measurements of compressed bitmap indices ». *Proceedings of the International Conference on Very Large Databases,* 1999.

[26] C. Chee-Yong. « Indexing techniques in decision support systems. » *Phd. thesis, University of Wisconsin - Madison,* 1999.

[27] Red Brick Systems. « Star schema processing for complex queries ». *White Paper,* July 1997.

[28] R. Bouchakri , L. Bellatreche, « On Simplifying Integrated Physical Database Design », *15th East European Conference on Advances in Databases and Information Systems (ADBIS'2011), LNCS, Springer,* September, 2011, pp. 333-346

[29] L. Bellatreche, « Dimension Table Selection Strategies to Referential Partition a Fact Table of Relational Data Warehouses », *Recent Trends in Information Reuse and Integration Book, Springer*, 2012, pp. 19-42

[30] H. Mahboubi , « Optimisation de la performance des entrepôts de données XML par fragmentation et répartition », *Thèse pour obtenir le grade de Docteur en Informatique*, *Université Lumière Lyon 2,* 08/12/2008

[31] L. Bellatreche, K. Karalapalem, et A. Simonet. « Algorithms and support for Horizontal class partitioning in object-oriented database ». *In the distributed and parallel Databases joutnal,* 8(2) : 155-179, April 2000.

[32] A. Noaman Y., Barker Ken. « A Horizontal Fragmentation Algorithm for the Fact Relation in a Distributed Data Warehouse ». *In : Proceedings of the 1999 ACM CIKM International Conference on Information and Knowledge Management,* November 1999, pp. 162-169

[33] L. Bellatreche , M. Schneider, Mukesh K. Mohania , Bharat K. Bhargava . « PartJoin : An Efficient Storage and Query Execution for Data Warehouses ». *In : Proceedings of Data Warehousing and Knowledge Discovery, 4th International Conference, DaWaK, Springer-Verlag,* 2002, pp. 296-306

[34] L. Bellatreche, K. Boukhalfa. « An Evolutionary Approach to Schema Partitioning Selection in a Data Warehouse ». *In : Proceedings of Data Warehousing and Knowledge Discovery, 7th International Conference, DaWaK 2005, Springer-Verlag,* 2005, pp. 115-125.

[35] P.E. O'Neil et G. Graefe : « Multi-table joins through bitmapped join indices ». *SIGMOD Record,* 24(3) :8–11, 1995.

[36] H. Gupta, V. Harinarayan, A. Rajaraman, et J. D. Ullman. «Index Selection for OLAP ». *In ICDE*, pages 208–219, 1997.

[37] W. J. Labio, D. Quass, and B. Adelberg. «Physical Database Design for DataWarehouses ». *In ICDE*, pages 277–288, 1997.

[38] D. Comer , « The difficulty of optimum index selection », A*CM Transactions on Database Systems (TODS)*, 3(4) :440-445. 1978.

[39] M.R. Frank, E. Omiecinski, et S.B. Navathe, « Adaptive and automated index

selection in RDBMS », *3rd International Conference on Extending Database Technology (EDBT 92), Vienna, Austria ; LNCS, Vol. 580, Springer, Heidelberg,* 1992, pp. 277-292.

[40] G. Valentin, M. Zuliani, D. Zilio, G. Lohman et A. Skelley : " DB2 advisor : An optimizer smart enough to recommend its own indexes". *16th International Conference on Data Engineering (ICDE 00), San Diego, USA, pages* 101–110, 2000.

[41] M. Golfarelli, E. Rizzi, et S. Saltarelli. «Index selection for data warehousing». *Proceedings 4th International Workshop on Design and Management of Data Warehouses (DMDW'2002), Toronto, Canada, pp* 33–42, 2002.

[42] S. Chaudhuri, M. Datar, et V. Narasayya, « Index Selection for Databases : A Hardness Study and a Principled Heuristic Solution ». *IEEE Transactions on Knowledge and Data Engineering, VOL. 16, NO.* 11. 2004.

[43] S. Taktak, J. Feki : "A maintenance Approach of a BJI Index Configuration" *In : ICSEA 2011, The Sixth International Conference on Software Engineering Advances, October 2011, Pages :* 221-226

[44] R. Bouchakri, L. Bellatreche, « Sélection statique et incrémentale des index de jointure binaire : concepts, algorithmes, étude de performance », *Journal of Decision Systems , vol. 21(1), January,* 2012, pp. 51-70

[45] S. Chaudhuri, V. Narasayya, « An efficient cost-driven index selection tool for Microsoft sql server » *In : Proceedings of the International Conference on Very Large Databases, August 1997, pp.* 146–155 (1997)

[46] K. Aouiche, O. Boussaid, et F. Bentayeb, « Automatic Selection of Bitmap Join Indexes in Data Warehouses ». *7th International Conference on Data Warehousing and Knowledge Discovery (DAWAK05), p.* 64-73. 2005.

[47] B. Ziani, Y. Ouinten, «Vers l'auto-sélection des index dans les entrepôts de données : une approche basée sur la recherche des motifs fréquents maximaux», *10e Colloque Africain sur la Recherche en Informatique et en Mathématiques Appliquées, Yamoussoukro, Côte d'Ivoire, pages* 301–308, 2010

[48] K. Boukhalfa, L. Bellatreche et B. Ziani «Index de Jointure Binaires : Stratégies de Sélection & Étude de Performances», *6ème Journées Francophones sur les Entrepots de Données et Analyse en Ligne (EDA10). - Jerba-Tunisie :* 2010. - pp. 175–190.

[49] L. Bellatreche and K. Boukhalfa, « Yet Another Algorithms for Selecting Bitmap Join Indexes », *Proceeding of 12th International Conference on Data Warehousing and Knowledge Discovery (DAWAK'08), Springer-Verlag,* September 2010.

[50] K. Aouiche, J. Darmont, O. Boussaid, "Sélection automatique d'index dans les entrepôts de données", *1er atelier Fouille de Données Complexes dans un processus d'extraction des connaissances, EGC 04, Clermont-Ferrand, Janvier* 2004, 91-102.

[51] T.I. Gundem ., « Near optimal multiple choice index selection for relational databases », *Computers & Mathematics with Applications,* 37(2) :111_120. 1999.

[52] Y.A. Feldman , J. Reouven , « A knowledge_based approach for index selection in relational databases », *Expert System with Applications,* 25(1) :15_37. 2003.

[53] T. Stöhr, H. Märtens, and E. Rahm. « Multidimensional database allocation for parallel data warehouses ». *Proceedings of the International Conference on Very Large Databases, pages* 273–284, 2000.

[54] S.Agrawal„ S.Chaudhuri, et Narasayya. « Materialized view and index selection tool for Microsoft SQL server 2000 ». *In ACM SIGMOD international conference on management of data (SIGMOD 2001) (p. 608). Santa Barbara, USA.* 2001

[55] N. Pasquier, Y. Bastide, R. Taouil et L.Lakhal, « Pruning closed itemset lattices for association rules ». *Actes des 14emes journées Bases de Données Avancées BDA'98*, 1998, pp. 177-196.

[56] J. Pei , J. Han et R. Mao, « CLOSET : An efficient algorithm for mining frequent closed itemsets. *In : ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp.* 21-30, 2000.

[57] M. J. Zaki et C. Hsiao, « Charm : an Efficient Algorithm for Closed Association

Rule Mining ». *In : 2nd SIAM International Conference on Data Mining, Arlington, Avril* 2002.

[58] G. Stumme , R. Taouil , Y. Bastide, N. Pasquier, L. Lakhal, « Computing Iceberg concept lattices with TITANIC », *Data & Knowledge Engineering* 2(42), p. 189-222.

[59] S. Azefack , « Sélection dynamique d'index dans les entrepôts de données », *Mémoire du master ECD, Université Lumière Lyon 2*, 2007

[60] C.-H Fung, K. Karlapalem, Q. Li, : « Cost-driven vertical class partitioning for methods in object oriented databases." *VLDB Journal 12*(3), 187–210 (2003)

[61] B. Ziani, Y. Ouinten, « Mining Maximal Frequent Itemsets : a java implementation of FPMAX algorithm », *6th International Conference on Innovations in Information Technology* (IIT09), 2009.

[62] G. Liu, "Supporting Efficient and Scalable Frequent Pattern Min-ing," *PhD dissertation, Dept. of Computer Science., .Hong Kong University., May* 2005.

[63] G. Grahne, J. Zhu, "Fast Algorithms for Frequent Itemset Mining Using FP-Trees", IEEE Transactions on Knowledge and Data Engineering, Vol 17, No 10, pages 1347-1362, October 2005.

[64] D. Burdick, M.Calimlim ,J.Flannick, J.Gehrke,Y.Yiu, "MAFIA : A Maximal Frequent Itemset Algorithm", *IEEE Transactions on Knowledge and Data Engineering, VOL* 17, No 11,Pages 1490 - 1504, November 2005.

[65] J. Roberto Jr. Bayardo Bart Goethals and Mohammed Javeed Zaki, FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004

[66] J. Han, J. Pei, Y. Yin, « Mining Frequent Patterns without Candidate Generation. » *Proc. ACMSIGMOD Int. Conf. on Management of Data (SIGMOD'00), Dallas, TX* , 2000.

[67] R. Srikant , Q. Vu & R. Agrawal . « Mining association rules with item constraints ». *In Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (KDD'97)*, p. 67–73 (1997)

[68] R. NG , L. V. Lakashmanan , J. HAN & A. Pang. « Exploratory mining and pruning optimizations of constrained associations rules ». *In Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'98),* (1998)

[69] J. Pei et J. Han. « Can we push more constraints into frequent pattern mining ? » *In Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining (KDD'00)*, p. 350–354, (2000)

[70] M. J. Zaki. « Sequence mining in categorical domains : incorporating constraints ». *In Proceedings of the 9th ACMInternational Conference on Information and Knowledge Management (CIKM'00)*, p. 422–429, (2000)

[71] C. Bucila, J. E. Gehrke, D. Kifer et W. White. « Dualminer : A dual-pruning algorithm for itemsets with constraints ». *Data Mining and Knowledge Discovery,* 7(4), 241–272. (2003)

[72] F. Bonchi, F. Giannotti, et D. Pedreschi.« A Relational Query Primitive for Constraint-Based Pattern Mining. » *In : Constraint-Based Mining and Inductive Databases, Vol.* 3848Springer , p. 14-37. (2004)

[73] F. Bonchi et C. Lucchese. « Pushing tougher constraints in frequent pattern mining ». *In Proceedings of the Ninth Pacific-Asia Conference on Knowledge Discovery and Data Mining* (PAKDD'05), Hanoi, Vietnam, 2005

[74] B. Jeudy et F. Rioult, « Extraction de concepts sous contraintes dans des données d'expression de gènes » *In François Denis, editor, Actes de CAP 05, Conférence francophone sur l'apprentissage automatique* - 2005, Nice, France. pages 265-280, 2005.

[75] L. De Raedt and A. Zimmermann, « Constraint-based pattern set mining, » *in Proc. SDM. SIAM,* pp. 237–248. 2007

[76] J. Klema, , S. Blachon, , A.Soulet, , B. Cremilleux, , O.Gandrilon, « Constraint-Based Knowledge Discovery from SAGE Data ». *In Silico Biology,* 8, 0014, 2008.

[77] S. Dzeroski, B. Goethals & P. Panov (Eds.), Inductive Databases and Constraint-Based Data Mining : 59-77, Springer (2011) ;

[78] N. Anh Tran, Hai V. Duong, Tin C. Truong, Bac H. Le : « Efficient Algorithms for Mining Frequent Itemsets with Constraint ». *in Third International*

Conference on Knowledge and Systems Engineering, KSE 2011, Hanoi, Vietnam, pp 19-25, 2011

[79] O. COUNCIL, " Apb-1 OLAP benchmark, release II". Http : //www.olapcouncil.org/research/resrchly.htm », Consulté le 25-10-2012

[80] A. Soulet, B. Crémilleux « Extraction des Top-k Motifs par Approximer-et-Pousser » *Actes des cinquièmes journées Extraction et Gestion des connaissances (EGC'2007)*, vol. 9, 2007 .Pages : 271-282

[81] M. Khiari, P. Boizumault, B. Crémilleux, « Extraction de motifs n-aires utilisant la PPC ». *6-èmes Journées Francophones de Programmation par Contraintes (JFPC'10)* 2010

[82] J. Kratica, I. Ljubic, et D. Tosic, « A Genetic Algorithm for the Index Selection Problem ». *In Applications of Evolutionary Computing, EvoWorkshop.* 2003.

[83] V. Sirirut, Le Gruenwald : "Indexing techniques for data warehouses Queries". *University of Oklahoma.*1999

[84] J-F. Boulicaut, A. Bykowski, B. Jeudy, C. Rigotti : "Représentations condensées et extractions sous contraintes de motifs fréquents". *Soumission à Bilan du Groupe de Travail GaFoumm, INSA Lyon.* 2002

[85] I. Mitasiunaite : « Mining string data under similarity and soft-frequency constraints : Application to promoter sequence analysis ». *Ph.D. thesis, INSA Lyon.* 2009

[86] A. Soulet, « Un cadre générique de découverte de motifs sous contraintes fondées sur des primitives ». *PhD thesis, Université de Caen Basse-Normandie, France,* 2006

[87] H. Mannila, et H. Toivonen, « Levelwise search and borders of theories in knowledge discovery ». *Data Min. Knowl. Discov.,* 1(3) : Pages :241-258. 1997

[88] R. Ng, T. Lakshmanan, L. V. S., Han, J. et A. Pang, « Exploratory mining and pruning optimizations of constrained association rules ». *Proceedings ACM SIGMOD International Conference on Management of Data*, Seattle, Washington, USA., pages 13-24. ACM Press. 1998

[89] C. Bucila, , J. Gehrke, , D. Kifer, et W. White, « DualMiner : A dual-pruning algorithm for itemsets with constraints. » *In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, July 23-26, 2002

[90] J. Han, , J. Pei, et Y. Yin, « Mining frequent patterns without candidate generation. » *In : SIGMOD Conference, pages* 1-12. ACM. 2000

[91] R. Agrawal, et R. Srikant, « Fast algorithms for mining association rules in large databases ». *In Bocca, J. B., Jarke, M. et Zaniolo, C., éditeurs : VLDB*, pages 487-499. 1994

Annexes

« Qui n'aime point escalader
les monts, traine à jamais dans
les crevasses.. »

(Abou el Kacem Chebbi)

Annexe A

L'outil Music-DFS

Un outil développé par Arnaud Soulet implémentant l'algorithme Music-DFS est disponible au site : http ://www.info.univ-tours.fr/~soulet/music-dfs/music-dfs.html. Cet Annexe donne une description de l'outil Music-DFS, elle est principalement tirée du tutoriel rédigé par l'auteur dédié à cet outil.

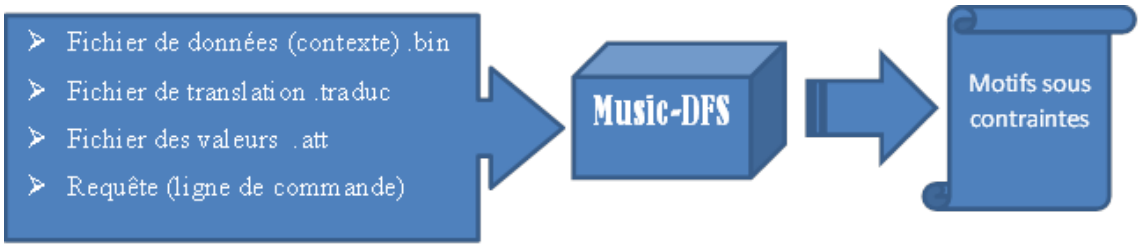

FIGURE A.1: Entrées et sorties de l'extracteur Music-DFS

1 Les entrées

1.1 Fichier de données (contexte) :

C'est un fichier binaire (.bin) contenant la base de données transactionnelle. Un exemple de fichier de donnée nommé data.bin est donné dans la figure A.2 où :

- La ligne commençant par un # est un commentaire.
- Chaque nombre correspond à un attribut.
- Chaque ligne (sans #) représente un objet (transaction, requête,...).

Par exemple, dans le contexte du problème de sélection d'index, le premier objet peut correspondre à une requête portant sur les attributs 1, 2, 5 et 6 .

```
# data.bin
1 2 5 6
1 5
1 2 3 4
1 2 3 4 5
4 5
3 6
```

FIGURE A.2: Exemple de fichier de donnée de d'outils Music-DFS

1.2 Fichier de translation :

Ce fichier optionnel dont l'extension est (.traduc) peut être utilisé pour convertir chaque nombre (correspondant à un attribut) en une chaine de caractères (par exemple : le nom de l'attribut).

La figure A.3 représente un exemple de fichier nommé par exemple *attributs.traduc*

```
1  A
2  B
3  C
4  D
5  E
6  F
```

FIGURE A.3: Exemple de fichier de traduction pour l'extracteur Music-DFS

1.3 Fichier des valeurs :

Ce fichier a comme extension (.att) et représente la table de valeurs contenant des informations supplémentaires pour chaque attribut. Par exemple, le fichier *values.att* présenté dans la figure A.4 donne pour chaque attribut, deux valeurs DIV et PRICE.

```
DIV     PRICE
4       15
2       10
4       5
5       20
2       100
2       40
```

FIGURE A.4: Exemple de fichier de valeurs pour l'extracteur Music-DFS

– La première ligne présente les noms des valeurs (ici, DIV et PRICE) séparés par une tabulation.

– La deuxième ligne signifie que le premier objet a une valeur DIV = 4 et un PRIX = 15.
– La troisième ligne donne les mesures des mêmes valeurs pour le deuxième objet et ainsi de suite.

1.4 La Requête :

Les requêtes de l'utilisateur sont exprimées à travers une ligne de commande, telle que la forme est la suivante :

music-dfs -i <FICHIER> -q <CONTRAINTE> [OPTION] ...

Example : *music-dfs -i dataset.bin -q "count(X)*length(X)>=100;" -o motifs.txt*

– *-i* : spécifie le fichier binaire de donnée en entrée (ici, dataset.bin)
– *-o* : spécifie le fichier de sortie. C'est une option pour sauvegarder le résultat dans un fichier, sinon le résultat est affiché sur la sortie standard. D'autres options sont données dans la Figure A.5
– *-q* : spécifie la contrainte (ici, la fréquence d'un motif X * sa longueur >=100).
  – *X* est la variable désignant un motif.
  – Il est à noter que la contrainte se termine par un point-virgule « ; » et que l'utilisation des guillemets est parfois indispensable.
  – Une contrainte est une simple combinaison des primitives données dans la figure A.6 et qui peuvent être combinées récursivement.

Option	Signification	Description
-h	Aide (help)	Aide de l'extracteur
-q	Contrainte <CONSTRAINT>	Contrainte à satisfaire
-p	Élagage <PRUNING>	condition d'élagage négatif
-m	Mesure <MEASURE>	Mesure d'un motif
-v	version	La version de l'extracteur
-V	verbeux ...	mode verbeux
-i	Input <FICHIER>	Fichier de donnée
-o	Output < FICHIER >	Fichier contenant les motifs extraits
-a	Valeurs < FICHIER >	Fichier contenant les valeurs des attributs
-S	Similarité < FICHIER >	Matrice de similarité
-d	delta <NOMBRE>	nombre maximal des exceptions
-g	gamma < NOMBRE >	seuil absolu minimal de fréquence
-t	seuil <[0,1]>	seuil minimal de fréquence
-T	traduction < FICHIER > ...	Fichier permettant la traduction des motifs extraits
-l	latex	Sortie au format latex
-s	statistique	donne plusieurs statistiques

FIGURE A.5: Liste des Options de l'extracteur Music-DFS

primitive	opérandes	Nom	Description
COUNT	Un motif	Fréquence	***count(X)*** retourne la fréquence du motif X
LENGTH	Un ensemble	Longueur	***length(X)*** retourne la longueur du motif X
BCOUNT	Un motif, un ensemble d'objets	Fréquence partielle	***bcount(X,O)*** retourne la fréquence du motif X dans le sous ensemble de donnée O
SUBSET	Deux ensembles	Sous-ensemble	***S1 subset S2*** retourne Vrai si $S1 \subset S2$
SUBSETEQ	Deux ensembles	Sous-ensemble ou égale	***S1 subseteq S2*** retourne Vrai si $S1 \subseteq S2$
SUPSET	Deux ensembles	Sur-ensemble	***S1 supset S2*** retourne Vrai si $S1 \supset S2$
SUPSETEQ	Deux ensembles	Sur-ensemble ou égale	***S1 subseteq S2*** retourne Vrai si $S1 \supseteq S2$
EQ	Deux ensembles	Egale	***S1* EQ *S2*** retourne Vrai si $S1 = S2$
≥	Deux réels	Supérieur ou égale à	$r1 \geq r2$ retourne Vrai si $r1 \geq r2$
>	Deux réels	Supérieur à	$r1 > r2$ retourne Vrai si $r1 > r2$
≤	Deux réels	Inferieur ou égale à	$r1 \leq r2$ retourne Vrai si $r1 \leq r2$
<	Deux réels	Inferieur à	$r1 < r2$ retourne Vrai si $r1 < r2$
=	Deux réels	égale à	$r1 = r2$ retourne Vrai si $r1 = r2$
EXT	Un motif	extension	**EXT** *(X)* retourne l'extension du motif X
SUM	Un ensemble, un identifiant	somme de valeurs	**SUM(S.VAL)** retourne la somme des valeurs VAL de chaque élément dans S
MAX	Un ensemble, un identifiant	max de valeurs	**MAX(S.VAL)** retourne le maximum des valeurs VAL de chaque élément dans S
MIN	Un ensemble, un identifiant	min de valeurs	**MIN(S.VAL)** retourne le minimum des valeurs VAL de chaque élément dans S
+	Deux réels	plus	$r1 + r2$ retourne $r1 + r2$
-	Deux réels	moins	$r1 - r2$ retourne $r1 - r2$
*	Deux réels	multiplié	$r1 * r2$ retourne $r1 * r2$
/	Deux réels	division	$r1 / r2$ retourne $r1 / r2$
UNION	Deux ensembles	union	S1 **UNION** S2 retourne $S1 \cup S2$
INTER	Deux ensembles	intersection	S1 **INTER** S2 retourne $S1 \cap S2$
\	Deux ensembles	privation	$S1 \setminus S2$ retourne $S1 \setminus S2$
INSIM		En SIMilarité	
MINSIM		Min SIMilarité	
MAXSIM		Max SIMilarité	
SUMSIM		Somme SIMilarité	
SVSIM		Stated Valeur SIMilarité	
MVSIM		Missing Valeur SIMilarité	
REGEXP		EXPression REGulière	

FIGURE A.6: Les primitives implémentées dans l'extracteur Music-DFS

2 Les sorties :

Le résultat de l'extraction des motifs sous contraintes spécifiées par l'utilisateur est affiché sur la sortie standard et peut être enregistré dans un fichier ou sous format latex. Le résultat de la requêtes suivante : *$ music-dfs -i data.bin -q "count(x)>=2;"* est présenté dans la figure A.7 :

```
$ music-dfs -i data.bin -q "count(x)>=2;"
# music-dfs 0.0.5
1 & 4
1 3 , 2 4 & 2
1 5 & 3
1 5 2 & 2
1 4 , 2 & 2
1 2 & 3
6 & 2
4 & 3
4 3 , 2 & 2
4 5 & 2
4 2 & 2
3 & 3
3 2 & 2
5 & 4
5 2 & 2
2 & 3
```

FIGURE A.7: les sorties de l'extracteur Music-DFS

- La première ligne est un commentaire qui donne le nom et la version de l'extracteur
- Chaque ligne donne soit un motif soit un intervalle de motifs (avant le &) et sa fréquence (après le &).
  - 1 & 4 signifie que le motif 1 a une fréquence de 4
  - 1 3 , 2 4 & 2 signifie que tous les motifs inclus dans l'intervalle [{1,3},{2,4}] ont une fréquence de 2.
  - La liste des motifs est la suivante : {{1,3},{1,3,2},{1,3,4},{1,3,2,4}}.
- Un motif satisfaisant la contrainte n'est présent que dans un seul intervalle.

Annexe B

La charge de requêtes

1. SELECT time_level,count(*)
   FROM actvars a,prodlevel p
   WHERE a.product_level=p.code_level and p.class_level='po0hv1rich5w' group by time_level
2. SELECT line_level,sum(dollarcost)
   FROM actvars a,prodlevel p
   WHERE a.product_level=p.code_level and p.class_level='ci493yz9kzuj' group by line_level
3. SELECT count(*)
   FROM actvars a,prodlevel p
   WHERE a.product_level=p.code_level and p.class_level='fdxaq1n5u026'
4. SELECT time_level,avg(unitssold)
   FROM actvars a,timelevel t
   WHERE a.time_level=t.tid and t.quarter_level='q1' group by time_level
5. SELECT division_level,count(*)
   FROM actvars a,prodlevel p
   WHERE a.product_level=p.code_level and p.group_level='e4njtw0zr9fn' group by division_level
6. SELECT max(unitssold)
   FROM actvars a,timelevel t
   WHERE a.time_level=t.tid and t.quarter_level='q2'
7. SELECT time_level,count(*)
   FROM actvars a,prodlevel p
   WHERE a.product_level=p.code_level and p.family_level='bemfvk0n8125' group by time_level
8. SELECT year_level,sum(dollarcost)
   FROM actvars a,prodlevel p, timelevel t
   WHERE a.product_level=p.code_level and a.time_level=t.tid and p.family_level='ujhz4tzmjt6v group by year_level
9. SELECT count(*)
   FROM actvars a,prodlevel p
   WHERE a.product_level=p.code_level and p.family_level='shdf8qt29kff'
10. SELECT customer_level,avg(unitssold)
    FROM actvars a,prodlevel p
    WHERE a.product_level=p.code_level and p.family_level='agg214dg271q' group by customer_level
11. select month_level,count(*)
    from actvars a,timelevel t
    where a.time_level=t.tid and t.quarter_level='q3' group by month_level
12. SELECT sum(dollarcost)
    FROM actvars a,prodlevel p
    WHERE a.product_level=p.code_level and p.line_level = 'mj1f1u1eg009'
13. SELECT product_level,count(*)
    FROM actvars a,timelevel t
    WHERE a.time_level=t.tid and t.quarter_level='q4' group by product_level
14. SELECT retailer_level,avg(unitssold)
    FROM actvars a,prodlevel p ,custlevel c
    WHERE a.product_level=p.code_level and a.customer_level=c.store_level and p.division_level = 'bcr2t4k2k9d3' group by retailer_level
15. SELECT count(*)
    FROM actvars a,prodlevel p
    where a.product_level=p.code_level and p.division_level = 'xrlxy6h61slc'

16. SELECT customer_level,sum(dollarcost) FROM actvars a,prodlevel p WHERE a.product_level=p.code_level and p.division_level = 'rc5406urp1ie' group by customer_level
17. select all_level,count(*) from actvars a,prodlevel p ,chanlevel h where a.product_level=p.code_level and a.channel_level=h.base_level and p.division_level = 'g4ha5yitg3h7' group by channel_level
18. select count(*) from actvars a,timelevel t where a.time_level=t.tid and t.year_level = '1995'
19. SELECT product_level,sum(dollarcost) FROM actvars a,timelevel t WHERE a.time_level=t.tid and t.year_level = '1996' group by product_level
20. SELECT division_level,avg(unitssold) FROM actvars a,timelevel t ,prodlevel p WHERE a.time_level=t.tid and a.product_level=p.code_level and t.month_level = '1' group by division_level
21. SELECT count(*) FROM actvars a,timelevel t WHERE a.time_level=t.tid and t.month_level = '2'
22. SELECT product_level,count(*) FROM actvars a,timelevel t WHERE a.time_level=t.tid and t.month_level = '3' group by product_level
23. SELECT division_level,sum(dollarcost) FROM actvars a,timelevel t ,prodlevel p WHERE a.time_level=t.tid and a.product_level=p.code_level and t.month_level = '4' group by division_level
24. SELECT avg(unitssold) FROM actvars a,timelevel t WHERE a.time_level=t.tid and t.month_level = '5'
25. SELECT product_level,count(*) FROM actvars a,timelevel t WHERE a.time_level=t.tid and t.month_level = '6' group by product_level
26. select division_level,sum(dollarcost) from actvars a,timelevel t ,prodlevel p where a.time_level=t.tid and t.month_level = '7' and a.product_level=p.code_level group by division_level
27. SELECT sum(dollarcost) FROM actvars a,timelevel t WHERE a.time_level=t.tid and t.month_level = '8'
28. SELECT customer_level,count(*) FROM actvars a,timelevel t WHERE a.time_level=t.tid and t.month_level = '9' group by customer_level
29. select year_level,sum(dollarcost) from actvars a,timelevel t where a.time_level=t.tid and t.month_level = '10' group by year_level
30. SELECT count(*) FROM actvars a,timelevel t WHERE a.time_level=t.tid and t.month_level = '11'
31. SELECT product_level,time_level,avg(unitssold) FROM actvars a,timelevel t WHERE a.time_level=t.tid and t.month_level = '12' group by product_level,time_level
32. select year_level,month_level, max(unitssold) from actvars a,custlevel c ,timelevel t where a.customer_level=c.store_level and c.retailer_level = 'z6ofs4yaad4j' and a.time_level=t.tid group by year_level,month_level
33. SELECT count(*) FROM actvars a,custlevel c WHERE a.customer_level=c.store_level and c.retailer_level = 'rqjnen0upkmq'
34. SELECT product_level,time_level, min(unitssold) FROM actvars a,custlevel c WHERE a.customer_level=c.store_level and c.retailer_level = 'nxeyfsiqe3jm' group by product_level,time_level
35. SELECT year_level,sum(dollarcost) FROM actvars a,custlevel c ,timelevel t WHERE a.customer_level=c.store_level and a.time_level=t.tid and c.gender_level='m' group by year_level
36. SELECT count(*) FROM actvars a,chanlevel ch WHERE a.channel_level=ch.base_level and ch.all_level ='abcdefghijkl'

37. SELECT channel_level,time_level, sum(dollarcost)
    FROM actvars a,chanlevel ch
    WHERE a.channel_level=ch.base_level and ch.all_level ='bcdefghijklm' group by channel_level, time_level
38. SELECT class_level,year_level, time_level, avg(unitssold)
    FROM actvars a,chanlevel ch ,prodlevel p ,timelevel t
    WHERE a.channel_level=ch.base_level and a.product_level=p.code_level and a.time_level=t.tid and ch.all_level ='cdefghijklmn' group by class_level,year_level, time_level
39. SELECT count(*)
    FROM actvars a,chanlevel ch
    WHERE a.channel_level=ch.base_level and ch.all_level ='defghijklmno'
40. SELECT time_level,count(*)
    FROM actvars a,chanlevel ch
    WHERE a.channel_level=ch.base_level and ch.all_level ='efghijklmnop' group by time_level
41. SELECT year_level,month_level, sum(dollarcost)
    FROM actvars a,custlevel c,prodlevel p ,timelevel t
    WHERE a.customer_level=c.store_level and a.time_level=t.tid and a.product_level=p.code_level and p.class_level='ci493yz9kzuj' and c.retailer_level='rqjnen0upkmq' and c.city_level='bordeaux' group by year_level,month_level
42. SELECT sum(dollarcost)
    FROM actvars a, prodlevel p,timelevel t
    WHERE a.product_level=p.code_level and a.time_level=t.tid and t.quarter_levl='q2' and t.year_level='1996' and p.class_level='fdxaq1n5u026'
43. SELECT customer_level, time_level, avg(unitssold)
    FROM actvars a, chanlevel h,prodlevel p, timelevel t,custlevel c
    WHERE a.product_level=p.code_level and a.time_level=t.tid and a.customer_level=c.store_level a.channel_level=h.base_level and p.class_level='fdxaq1n5u026' and h.all_level ='abcdefghijkl' and t.querter_level='q1' and c.gender_level='f' group by customer_level, time_level
44. SELECT year_level, division_level, max(unitssold)
    FROM actvars a, custlevel c,timelevel t ,prodlevel p
    WHERE a.customer_level=c.store_level and a.product_level=p.code_level and a.time_level=t.tid and t.month_level='1' and c.retailer_level ='rqjnen0upkmq' and c.gender_level='f' and c.city_level='poitiers' group by year_level, division_level
45. SELECT sum(dollarcost), avg(unitssold)
    FROM actvars a, custlevel c,timelevel t
    WHERE a.customer_level=c.store_level and a.time_level=t.tid and t.month_level='12' and c.retailer ='rqjnen0upkmq' and c.city_level='dijon'
46. SELECT customer_level, time_level,min(unitssold)
    FROM actvars a, chanlevel h,timelevel t, custlevel c
    WHERE a.channel_level=h.base_level and a.cutomer_level=c.store_level and a.time_level=t.tid and t.year_level='1996' and t.quarter_level='q3' and h.all_level='defghijklmno' and c.gender_level='m' group by customer_level, time_level
47. SELECT month_level, all_level, time_level, sum(dollarcost)
    FROM actvars a, custlevel c,prodlevel p,timelevel t ,chanlevel h
    WHERE a.customer_level=c.store_level and a.product_level=p.code_level and a.channel_level=h.base_level and a.time_level=t.tid and t.year_level='1996' and c.retailer_level ='nxeyfsiqe3jm' and c.city_level='paris' and p.line_level='mj1f1u1eg009' group by month_level,all_level, time_level
48. SELECT avg(unitssold)
    FROM actvars a, chanlevel h,prodlevel p,timelevel t
    WHERE a.product_level=p.code_level and a.channel_level=h.base_level and a.time_level=t.tid and t.month_level='1' and p.division_level='xrlxy6h61slc' and h.all_level='bcdefghijklm'
49. SELECT customer_level, product_level, channel_level, sum(dollarcost)
    FROM actvars a, chanlevel h,custlevel c,prodlevel p
    WHERE a.customer_level=c.store_level and a.product_level=p.code_level and a.channel_level=h.base_level and

p.family_level='agg214dg271q' and c.retailer_level='nxeyfsiqe3jm' and c.gender_level='m' and h.all_level='defghijklmno' group by customer_level, product_level, channel_level

50. SELECT division_level, year_level, max(unitssold)
FROM actvars a, chanlevel h,custlevel c,timelevel t ,prodlevel p
WHERE a.customer_level=c.store_level and a.product_level=p.code_level and a.channel_level=h.base_level and a.time_level=t.tid and t.month_level='4' and c.retailer_level='nxeyfsiqe3jm' and c.city_level='poitiers' and c.gender8level='f' and h.all_level='bcdefghijklm' group by division_level, year_level

51. SELECT min(unitssold)
FROM actvars a, chanlevel h,custlevel c,prodlevel p,timelevel t
WHERE a.customer_level=c.store_level and a.product_level=p.code_level and a.channel_level=h.base_level and a.time_level=t.tid and t.month_level='7' and p.group_level='e4njtw0zr9fn' and c.retailer_level='z6ofs4yaad4j' and h.all_level='efghijklmnop'

52. SELECT customer_level, channel_level, sum(dollarcost)
FROM actvars a, chanlevel h,custlevel c,prodlevel p,timelevel t
WHERE a.customer_level=c.store_level and a.product_level=p.code_level and a.channel_level=h.base_level and a.time_level=t.tid and t.month_level='11' and p.class_level ='fdxaq1n5u026' and c.retailer_level ='rqjnen0upkmq' and c.city_level='poitiers' and h.all_level='defghijklmno' group by customer_level, channel_level

53. SELECT line_level, month_level, avg(unitssold)
FROM actvars a, chanlevel h,custlevel c,prodlevel p,timelevel t
WHERE a.customer_level=c.store_level and a.product_level=p.code_level and a.channel_level=h.base_level and a.time_level=t.tid and t.year_level='1995' and t.quarter_level='q1' and p.group_level='e4njtw0zr9fn' and c.retailer_level='rqjnen0upkmq' and h.all_level='bcdefghijklm' group by line_level, month_level

54. SELECT min(unitssold)
FROM actvars a, chanlevel h,custlevel c,prodlevel p,timelevel t
WHERE a.customer_level=c.store_level and a.product_level=p.code_level and a.channel_level=h.base_level and a.time_level=t.tid and t.year_level='1995' and t.quarter_level='q1' and t.month_level in('2','3','4') and p.class_level='ci493yz9kzuj' and c.retailer_level in ('z6ofs4yaad4j','rqjnen0upkmq') and c.gender_level='f' and c.city_level='poitiers' and h.all_level ='abcdefghijkl'

55. SELECT customer_level, product_level, time_level, channel_level, max(unitssold)
FROM actvars a, chanlevel h,custlevel c,prodlevel p,timelevel t
WHERE a.customer_level=c.store_level and a.product_level=p.code_level and a.channel_level=h.base_level and a.time_level=t.tid and t.year_level='1996' and t.month_level in('5','6','7') and p.class_level='fdxaq1n5u026' and c.gender_level='m' and c.retailer_level='rqjnen0upkmq' and h.all_level ='defghijklmno' group by customer_level, product_level, time_level, channel_level

56. SELECT sum(dollarcost)
FROM actvars a,timelevel t
WHERE a.time_level=t.tid and c.gender_level='f'

57. SELECT month_level,sum(dollarcost)
FROM actvars a,custlevel c ,timelevel t
WHERE a.customer_level=c.store_level and a.time_level=t.tid and c.city_level='poitiers' group by month_level

58. SELECT time_level,avg(dollarcost)
FROM actvars a,custlevel c
WHERE a.customer_level=c.store_level and c.city_level='bordeaux' group by time_level

59. SELECT avg(dollarcost)
FROM actvars a,custlevel c
WHERE a.customer_level=c.store_level and c.city_level='paris'

60. SELECT year_level,max(dollarcost)
FROM actvars a,custlevel c
WHERE a.customer_level=c.store_level and c.city_level='dijon' group by year_level