REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université Abou Bekr Belkaid
Tlemcen Algérie

جامعة أبي بكر بلقايد
تلمسان الجزائر

Faculté des Sciences – Tidjani HADDAM
Département de l'Informatique

Mémoire

Pour l'obtention du diplôme de

MASTER

Spécialité :
Informatique

Option :
Modèles Intélligents et Décision (MID)

THÈME

Techniques Data Mining pour la sélection d'une configuration d'index de jointure binaire

Présenté par :

Mr : AMARA Mohammed
Mr : BENOUAZ Omar Farouk

Devany le jury

Président :
Mr. LAHSAINI Mohammed
Encadreur :
Mr. HADJILA Fethallah
Co-Encadreur :
Mr. LYAZID Toumi
Examinateurs :
Mr. BELABED Amine
Mr. BENAMAR Abdelkrim
Mr. BENAZZOUZ Mourtada
Mme. ILES Nawel
Mr. SMAHI Ismail

Année universitaire : 2010/2011

REMERCIEMENTS

Louanges à Dieu qui nous a donné la force, le courage, et l'espoir nécessaire pour accomplir ce travail et surmonter l'ensemble des difficultés.

Nous tenons à remercier vivement :

- **Nos encadreurs, Mr. Hadjila Fathallah, Mr. Lyazid Toumi pour leurs conseils et leur suivi durant la réalisation de ce projet.**
- **Mr. A. Benammar, Mr. M. Lahasaini, Mr. M. Benazzouz, Mr. A. Belabed, Mr. I. Smahi et Mme. N. Iles qui ont bien voulu faire partie de jury.**

Nous remercions toutes les personnes qui ont aidé de prés ou loin au cours de ce travail.

Sommaire

Liste de figures

Liste des tableaux

Introduction générale

Introduction Générale

Un entrepôt de données, ou Data Warehouse, est une vision centralisée et universelle de toutes les informations de l'entreprise. C'est une structure (comme une base de données) qui à pour but, contrairement aux bases de données, de regrouper les données de l'entreprise pour des fins analytiques et pour aider à la décision stratégique. La décision stratégique étant une action entreprise par les décideurs de l'entreprise et qui vise à améliorer, quantitativement ou qualitativement, la performance de l'entreprise. En gros, c'est un gigantesque tas d'informations épurées, organisées, historiées et provenant de plusieurs sources de données, servant aux analyses et à l'aide à la décision.

Les entrepôts de données sont très souvent modélisés par un schéma en étoile. Ce schéma est caractérisé par une table de faits de très grande taille liée à un ensemble de tables de dimension de plus petite taille. La table des faits contient les clés étrangères des tables de dimension ainsi qu'un ensemble de mesures collectées durant l'activité de l'organisation. Les tables de dimension contiennent des données qualitatives qui représentent des axes sur lesquels les mesures ont été collectées. Les requêtes définies sur un schéma en étoile (connues par requêtes de jointure en étoile) sont caractérisées par des opérations de sélection sur les tables de dimension, suivies de jointures avec la table des faits. Aucune jointure n'existe entre les tables de dimension. Toute jointure doit passer par la table des faits, ce qui rend le coût d'exécution de ces requêtes très important. Sans technique d'optimisation, leurs exécutions peuvent prendre des heures, voire des jours.

Pour optimiser ces requêtes, l'administrateur est amené à effectuer une tache importante: la conception physique. Durant cette phase l'administrateur choisit un ensemble de techniques d'optimisation à sélectionner. Ces techniques appartiennent à deux catégories : redondantes comme les index et les vues matérialisées et non redondantes comme la fragmentation horizontale et le traitement parallèle. Dans le cadre de ce mémoire, nous nous intéressons à une technique d'optimisation redondante (index de jointure binaire).

Le but de notre travail est de trouver une configuration d'index binaire de jointure optimale qui permet d'optimiser le coût de l'ensemble des requêtes. Dans ce but nous voulons proposer une nouvelle approche basée sur la logique floue. Une étude comparative a été faite avec d'autres approches déjà proposées telles que DynaClose, DynaCharm. Les tests réalisés sur un banc d'essais (APB-1) démontrent la qualité de l'approche que nous avons proposée.

Le mémoire est organisé comme suit :

Le premier chapitre est consacré aux définitions propres aux domaines des entrepôts de données.

Le deuxième chapitre présente les différentes approches d'extraction des motifs fréquents. Dans ce chapitre nous avons détaille les 2 approches classiques d'extraction des motifs (Apriori, Close) ainsi qu'une approche a base de la tolérance aux fautes. Puis une comparaison a été faite avec les différentes approches selon le temps d'exécution et le nombre des items fréquents générés.

Le troisième chapitre quant à lui présente d'une façon détaillée la technique d'optimisation index binaire de jointure, ainsi, que le problème de sélection des index binaire de jointure. Nous présentons aussi les travaux qui portent sur ces derniers. Nous présentons aussi dans ce chapitre le modèle de coût utilisé pour évaluer les différents approches utilises.

Le quatrième chapitre de ce mémoire présente la description des principes de l'approche proposée, ainsi que explication des étapes de développement du nouvel algorithme de sélection d'une configuration d'index binaire de jointure.

A la fin du quatrième chapitre, nous avons présenté nos résultats de validation sur un banc d'essai APB-1 et un ensemble de 60 requêtes. Les résultats obtenus ont été comparés à d'autres approches comme DynaClose et DynaCharm.

Enfin le mémoire se termine par une conclusion générale rappelant tous les Apports de l'approche propose avec une proposition de quelques perspectives Jugées utiles.

Chapitre I

Les entrepôts de données

Introduction

Les entrepôts de données manipulent généralement des données d'une taille de Plusieurs Téraoctets et cette taille augmente énormément dans le cas des bases de Données scientifique, ce type de bases de données est modélisé par de grands schémas (schémas en étoile, schémas en flocon de neige, etc. [13]) avec plusieurs tables qui contiennent beaucoup d'attributs. L'exécution d'une requête sur ces schémas peut prendre un temps de réponse énorme où les opérations de jointure prennent énormément de temps. Plusieurs structures d'optimisation sont proposées dans la littérature et dans les SGBDs commerciaux pour accélérer le traitement des requêtes [14]. Ces structures sont subdivisées en deux principales catégories [12] : les structures redondantes et les structures non redondantes. Dans la première catégorie, on recense les index (mono ou multi-attributs), les vues matérialisées ainsi que la fragmentation verticale. La deuxième catégorie, quand à elle, regroupe la fragmentation horizontale primaire et dérivée.

Nous présentons dans ce chapitre l'architecture des entrepôts et les différents modèles de leurs modélisations ainsi que la classification de différentes techniques d'optimisation.

I.1) Architecture et modélisation des entrepôts de données

Nous présentons dans cette section l'architecture générale d'un entrepôt de données et ce qui le différencie d'une base de données classique. Nous présentons ensuite les différents modèles pour représenter les entrepôts de données multidimensionnels et relationnels.

I.1.1) Architecture d'un entrepôt de données

Selon Bill Inmon, l'un des premiers fondateurs des entrepôts des données, "un entrepôt de données est une collection de données thématiques, intégrées, non volatiles et historiées pour la prise de décisions."[15].

L'intégration des sources de données hétérogènes ayant des structures différentes est le principal objectif de la naissance des entrepôts de données. Cependant, l'entrepôt de données joue un rôle stratégique dans la vie d'une entreprise. Il stocke des données pertinentes aux besoins de prise de décision en provenance des systèmes opérationnels de l'entreprise et d'autres sources externes. A la différence d'une base de données

classique supportant des requêtes transactionnelles de type **OLTP** (On-Line Transaction Processing), un entrepôt de données est conçu pour supporter des requêtes de type **OLAP** (On-Line Analytical Processing). L'interrogation est l'opération la plus utilisée dans le contexte d'entrepôt de données où la mise à jour consiste seulement à alimenter l'entrepôt. Le tableau I.1 montre la différence entre un système opérationnel doté d'une base de données classique et un entrepôt de données [12].

	Systèmes opérationnels	Entrepôt de données
But	**Exécution d'un processus métier**	**Evaluation d'un processus métier**
Interaction l'utilisateur	Insertion, Modification, Interrogat Suppression	Interrogation
Données	Courantes	Courantes et Historiques
Usage	Support de l'opération de l'entreprise	Support de l'analyse de l'entreprise
Requêtes	Simples, Prédéterminées	Complexes, Ad-hoc
Type d'utilisateur	Employé	Décideur
Principe conception	Troisième forme normale	Conception multidimensionnelle

Tab. I.1 – Base de données vs Entrepôt de données

Le processus de construction d'un entrepôt de données est composé de trois principales phases [12] :

1) Extraction des données à partir de différentes sources.
2) Organisation et intégration des données dans l'entrepôt.
3) Accès aux données intégrées et analyse de ces dernières dans une forme efficace et flexible (voir figure I.1)

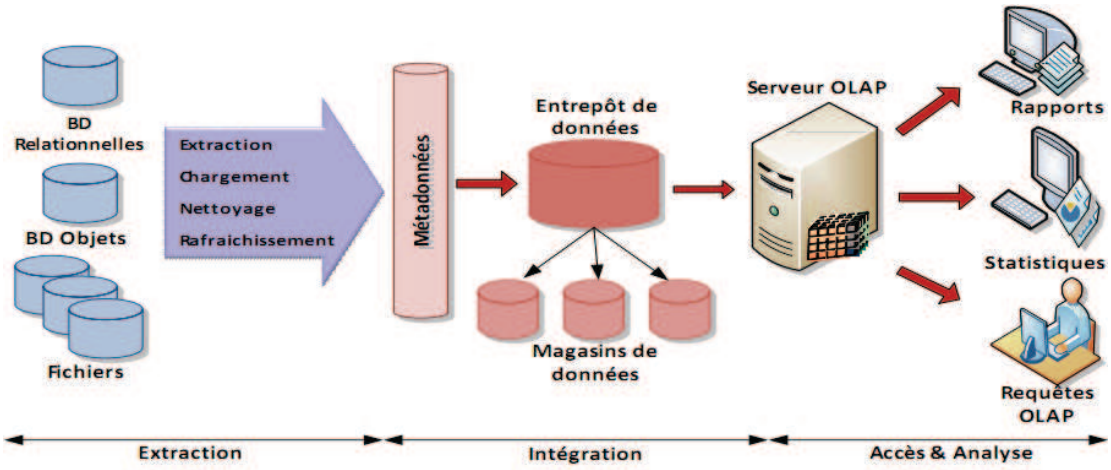


Fig. I.1 – Architecture d'un entrepôt de données

Dans la première et la deuxième phase, les données issues de différentes sources de données sont extraites, nettoyées et intégrées dans l'entrepôt de données. Les métadonnées contiennent des informations utiles sur la création, l'utilisation et la gestion de l'entrepôt. Durant la troisième phase, un serveur OLAP se charge de présenter les informations demandées par les utilisateurs sous plusieurs formes : tableaux, rapports, statistiques, etc.

I.1.2) Modélisation d'un entrepôt de données

Les données manipulées dans le contexte d'entrepôts de données sont représentées sous forme **multidimensionnelle** qui est mieux adaptée pour le support des processus d'analyse [16]. Ce modèle permet d'observer les données selon plusieurs perspectives ou axes d'analyse et facilite l'accès aux données.

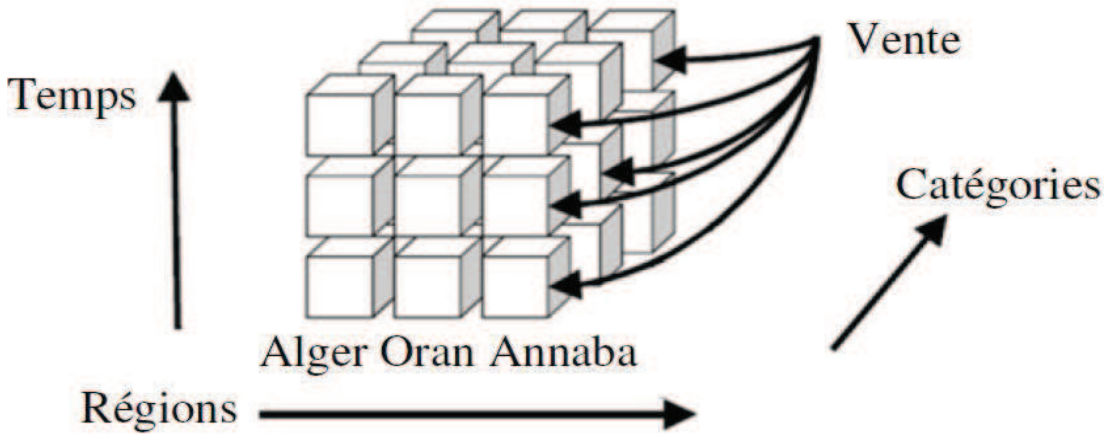


Fig. I.2 – schéma multidimensionnel de données

Deux concepts fondamentaux caractérisent le modèle multidimensionnel : **faits et dimensions**. Un fait représente le sujet analysé, il est formé de mesures correspondant aux informations de l'activité étudiée. Ces mesures sont calculables à partir d'un grand nombre d'enregistrements en appliquant les opérations d'addition, de calcul du minimum ou de la moyenne, etc. [17]. Par exemple pour la gestion des commandes client, les mesures peuvent être la quantité de produits commandés, le montant de la commande, etc.

La dimension représente l'axe d'analyse de l'activité. Elle regroupe des paramètres et des informations qui peuvent influencer sur les mesures d'activités du fait. Si on considère l'analyse de la quantité de produits vendus, il est évident que cette quantité varie selon la nature des données présentes dans la dimension produite. C'est ainsi que les dimensions doivent fournir des règles de calculs pour chaque mesure.

L'implémentation du modèle multidimensionnel sur un SGBD réel peut se faire selon deux modèles : **MOLAP** (Multidimensional On-Line Analytical Processing) et **ROLAP** (Relational On-Line Analytical Processing). Nous décrivons ces deux implémentations dans la section suivante.

I.1.2.1) MOLAP

Les systèmes de type MOLAP constituent une approche qui permet de représenter les données de l'entrepôt sous la forme d'un tableau multidimensionnel à n dimensions, où chaque dimension de ce tableau est associée à une dimension de l'hyper cube de données. La figure I.3 représente un tableau multidimensionnel pré-calculant le volume de ventes effectuées par type de produit (dimension Produit), trimestre (dimension Temps) et ville (dimension Client). Les systèmes MOLAP nécessitent un pré-calcul de toutes les agrégations possibles afin de les matérialiser dans les cellules du tableau multidimensionnel. L'avantage principal de cette implémentation est le gain considérable en temps d'exécution des requêtes, vu que l'accès aux données est direct. Par contre, les opérations de mise à jour sont difficiles à effectuer compte tenu que les valeurs des cellules du tableau multidimensionnel doivent être recalculées après chaque opération de mise à jour.

Produit	Temps	Trimestre 1			Trimestre 2			Trimestre 3			Trimestre 4			Total		
	Ville	P	N	Total	P	N	Total	P	N	Total	P	N	Total	P	N	Total
TV LCD		12	34	46	22	36	58	24	37	61	33	55	88	91	162	253
Lecteur DVD		29	66	95	44	50	94	56	55	111	44	39	83	173	210	383
Caméoscope		55	34	89	69	27	96	31	26	57	68	70	138	223	157	380
Total		96	134	230	135	113	248	111	118	229	145	114	309	487	529	1016

Fig. 1.3 – Représentation sous forme de tableau multidimensionnel

I.1.2.2) ROLAP

Les systèmes de type ROLAP utilisent une représentation relationnelle du cube de données. Chaque fait correspond à une table appelée table de faits et chaque dimension correspond à une table appelée table de dimension. La table de faits possède comme attributs les mesures d'activités et les clés étrangères vers les tables de dimension. Ce modèle a pour avantage l'utilisation des systèmes de gestion de bases de données existants, ce qui réduit le coût de mise en œuvre. Plusieurs modèles ont été proposés pour la modélisation des systèmes ROLAP, les plus utilises sont : le schéma en étoile, le schéma en flocon de neige.

I.1.2.2.1) Schéma en étoile :

Dans ce modèle, chaque groupe de dimensions est placé dans une table de dimension ; les faits sont placés dans une table des faits. Le résultat de cette classification est un schéma en étoile où la table des faits se trouve au milieu de l'étoile et les tables de dimension dans les côtés [12].

Dans ce travail, nous considérons un entrepôt de données modélisé par un schéma en étoile.

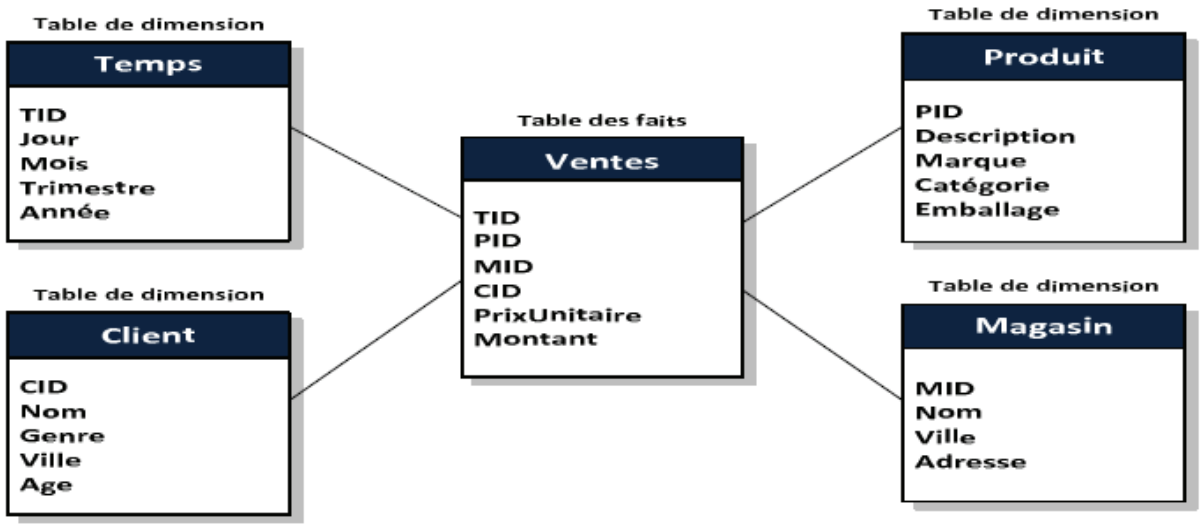


Fig. I.4 –Schéma en étoile

I.1.2.2.2) Schéma en flocon de neige

La dé-normalisation des tables de dimension dans un schéma en étoile ne reflète pas les hiérarchies associées à chaque dimension. Pour mettre en évidence cette hiérarchie, le modèle en flocon de neige a été proposé. Chaque table de dimension est éclatée en un

ensemble de hiérarchies. La figure 1.5 représente un schéma en flocon de neige. Sur ce schéma, deux hiérarchies sont représentées : (Jour, Mois, Trimestre, Année) et (Ville, Département, Pays).

I.1.3) les structures d'optimisation des entrepôts de données

La conception physique d'un entrepôt de données consiste à établir une configuration physique sur le support de stockage. Aujourd'hui face à la complexité des requêtes à traiter et le volume important des données, la conception physique a reçu une importance phénoménale [24].

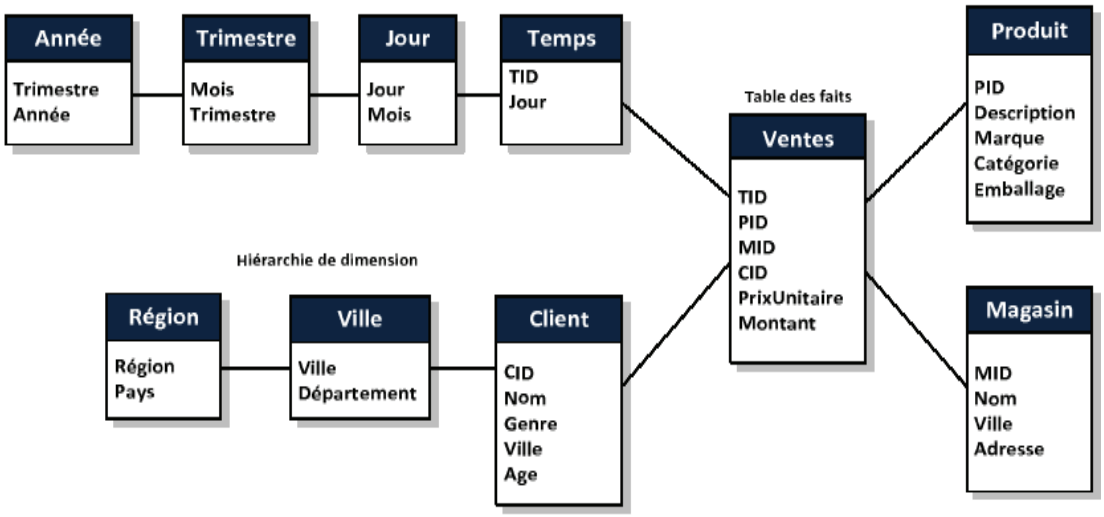


Fig. I.5–Schéma en flocon de neige

Cela comprend la spécification détaillée des éléments de données, les types de données et la sélection des techniques d'optimisation. Cette dernière est au cœur de la conception physique. Plusieurs techniques d'optimisation ont été proposées dans la littérature et supportées par les systèmes de gestion de bases de données commerciaux. Nous pouvons les classer en deux catégories principales [12] : techniques redondantes et techniques non redondantes. La figure I.6 montre une classification des principales techniques d'optimisation.

I.2.3.1) Les techniques redondantes

Les techniques redondantes optimisent les requêtes, mais exigent un coût de stockage et de maintenance. Cette catégorie regroupe les vues matérialisées [20], les index [18], la fragmentation verticale [22], etc.…

I.2.3.2) Les techniques non redondantes

Les techniques non redondantes ne nécessitent ni coût de stockage ni coût de maintenance. Cette catégorie regroupe la fragmentation horizontale [21], le traitement parallèle [23], etc.

Le nombre limité de travaux de sélection des IBJ existants, nous a motivés pour étudier cette technique et essayer de présenter ses différents concepts.

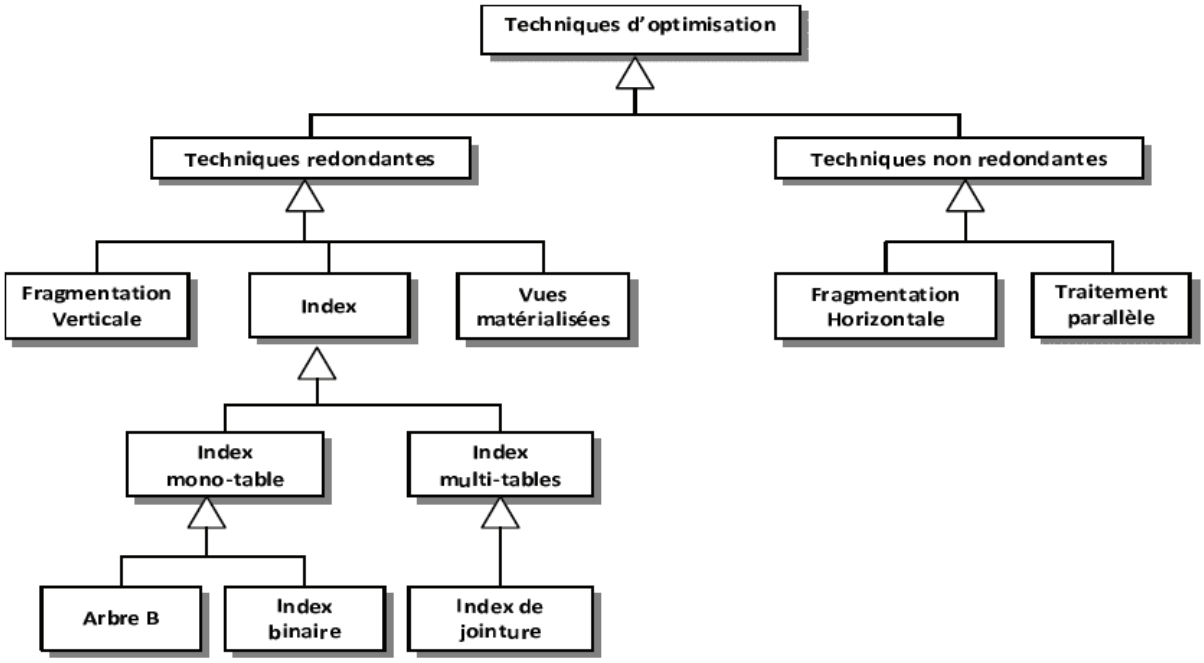


Fig. I.6–Techniques d'optimisation

Conclusion

Nous avons présente dans ce chapitre une présentation des entrepôts de données, leurs différence avec les bases de données traditionnelles, aussi leurs construction et leurs architecture générales.

Nous avons aussi vue la modélisation des entrepôts de données avec le modèle multidimensionnel qui peut être implémentée sur deux modèles : **MOLAP** et **ROLAP.** Deux schémas principaux utilisés pour la modélisation des systèmes ROLAP : le schéma en étoile et le schéma en flocon de neige etc… nous avons mis l'accent sur la classification des différentes techniques d'optimisation.

Chapitre II

Les motifs fréquents

Introduction

L'extraction de connaissances dans les bases de données (ECBD) est « le processus non trivial d'extraction d'informations valides, implicites, potentiellement utiles et ultimement compréhensibles à partir de grandes bases de données » [35]. De nombreuses applications dans divers domaines ont bénéficié des techniques de l'ECBD et de nombreux travaux ont été menés sur ce sujet. Un des grands problèmes traités par ECBD, et l'extraction des motifs fréquents.

La découverte des motifs fréquents consiste à trouver des groupements d'items apparaissant ensemble avec une fréquence significative. La découverte de ces motifs fréquents constitue l'étape principale de résolution de certain nombre de problèmes de l'ECBD [11] comme le problème de motifs fermés [10] et de motifs tolérants [1].

Dans le cadre de conception physique des entrepôts des données, il existe deux travaux qui se basent sur l'utilisation d'une méthode d'extraction des motifs fréquents dans leurs méthodes d'optimisation utilisé (plus précisément les IJBs). Nous inspirons de ces deux travaux, notre travail d'optimisation des entrepôts des données qui est base sur l'utilisation de l'une de ses algorithmes (Motif à base tolérance de faute). A ce point, Nous abordons dans ce chapitre les différents concepts de cette méthode et une présentation des deux algorithmes (Close et Charm) utilises dans les travaux précédents [28][29].

II.1) Extraction des motifs fréquents

Le but de cette partie n'est pas de détailler les différents concepts d'extraction des motifs fréquents, pour plus des détailles sur ces concepts, voir [30].

II.1.1) Motifs fréquents

Un motif fréquent est un ensemble d'éléments (items) présents dans un nombre suffisamment grand de lignes d'une base de données. Soient $I = i_1, \ldots, i_m$ un ensemble de m items et $B = t_1, \ldots, t_n$ une base de données de n transactions. Chaque transaction est composée d'un sous-ensemble d'items $I' \subseteq I$. Un sous-ensemble I' de taille k est appelé un k-itemset. Une transaction t_i contient un motif I' si et seulement si $I' \subseteq t_i$. Le support d'un motif I' est la proportion de transactions de B qui contiennent I'. Le support est donné par la formule suivante :

$$support(I') = \frac{|\{t \in B, I' \subseteq t\,|}{|\{t \in B|}$$

Pour décider si un motif est fréquent ou non, l'utilisateur fixe un seuil minimum du support *minsup*. Si *Support(I') >=minsup* alors *I'* est un motif fréquent, sinon il est non fréquent.

II.1.2) Algorithmes A-priori

Agrawal et al. ont proposé dans [8], le premier algorithme qui permet d'extraire des itemset fréquents dans les bases de données transactionnelles, l'algorithme Apriori. Etant donnée une base de données de transactions, le but principal de l'algorithme Apriori est d'extraire touts itemset fréquents et générer des règles d'association qui lit ces itemset fréquents entre eux.

Apriori se base essentiellement sur la propriété d'anti-monotansaonicité [1]existant entre les itemsets. En effet, cette propriété est utilisée à chaque itération de l'algorithme Apriori afin de diminuer le nombre d'itemsets candidats à considérer.

Exemple 2.1 : *l'exemple ci-dessous montre le processus d'extraction des motifs fréquents sur une base de transaction D qui contiens 5 transactions composons par sous ensemble des items {a,b,c,d}, pour in support minimum 0.4 correspond a 2 transaction.*

D	
tid	transaction
1	a b
2	a c
3	c d
4	b c d
5	a b c d

[1] Le terme « anti-monotone » est utilisés pour nommer cette propriété fondamentale : si $X \subseteq Y$, alors $f(X) \supseteq f(Y)$.

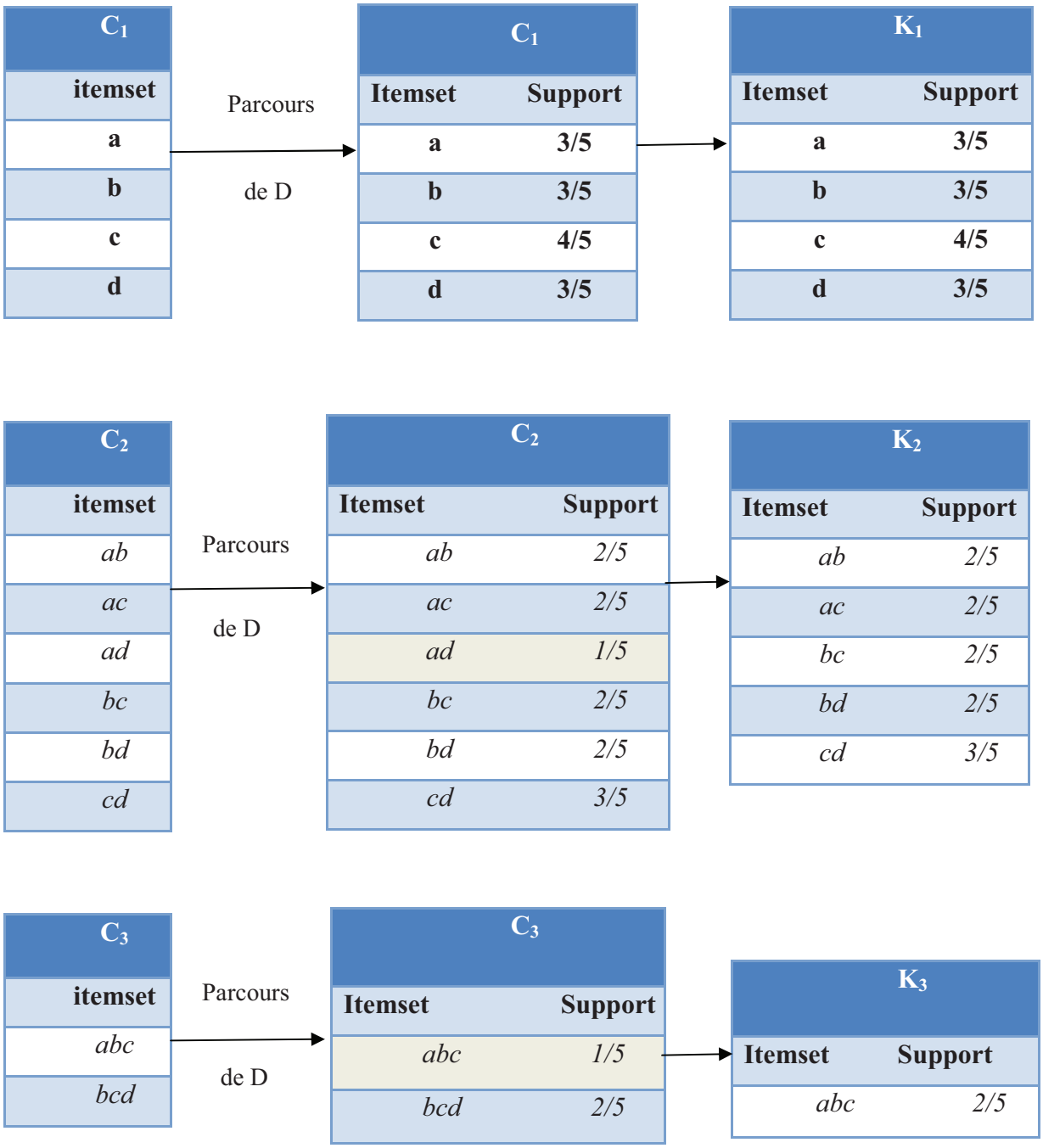


Figure II.1 – processus d'extraction des motifs fréquents avec A-priori

II.2) Extraction des motifs fermes

II.2.1) Les motifs fréquents fermes

Un motif fermé est un ensemble maximal de motifs communs à un ensemble d'objets. Un motif $i' \subseteq I$ tel que support(i) $\geq$ minsup est appelé motif fréquent fermé.

II.2.2) L'algorithme CLOSE pour la recherche des motifs fréquents

La génération des motifs fréquents est une tâche compliquée, car le nombre de motifs fréquents peut exploser lorsque le nombre de motifs initial est important. Pour réduire cette complexité, l'algorithme CLOSE génère les motifs fréquents fermés, ils se basent sur la propriété de Galois [10]. L'ensemble des motifs fréquents peut être généré directement à partir de l'ensemble des motifs fréquents fermés.

L'algorithme se base sur les étapes suivantes :

L'algorithme CLOSE parcourt l'ensemble des générateurs des motifs fermés fréquents par niveaux. À l'étape k = 1, l'ensemble des 1-générateurs est initialisé aux 1-itemsets. À chaque itération, l'algorithme considère un ensemble de k-itemsets générateurs. Il construit un ensemble de motifs fermés candidats qui sont les fermetures de ces k-générateurs et détermine ensuite parmi ces candidats les motifs fermés fréquents selon le seuil minimal du support minsup. Finalement, il crée les (k + 1)-générateurs qui seront utilisés lors de l'itération suivante afin de construire l'ensemble des motifs fermés candidats qui sont les fermetures des (k + 1)-générateurs. Un balayage du contexte d'extraction est nécessaire durant chaque itération, afin de déterminer les fermetures des k-générateurs et calculer leurs supports.

Si l'ensemble de k-générateurs fréquents est vide, l'algorithme s'arrête. Sinon, ce nouvel ensemble de (k +1)-générateurs est utilisé à l'itération suivante stocke un ensemble FF_k contenant les k-générateurs fréquents, leur fermeture qui est des itemsets fermés fréquents et leurs supports.

Exemple 2.2 *La figure II.2 Présente le résultat d'application l'algorithme Close sur la base transactionnelle D définit dans l'exemple 3, avec un minsup =3.*

II.3) Itemsets fréquent à base tolérance de faute

En 2011, *Cheng Yang et al* [3], ils ont montré par exemple que les algorithmes classique utilisés pour extraire les itemsets fréquents, ils sont incapable d'extraire certains itemsets qui sont considérés comme intéressants pour analyse des données, ce problème se trouve au niveau de définition stricte du support d'un itemset. Pour balayer ce problème, les auteurs en [3] ont conduit à introduire la notion de relaxation. Ce dernier permet de découvrir les motifs considères flous pour les méthodes classiques.

Dans cette section, nous présentons les principales notions de relaxation, les niveaux d'applications de la relaxation, et l'algorithme qui nous avons choisi d'utiliser pour les extraire.

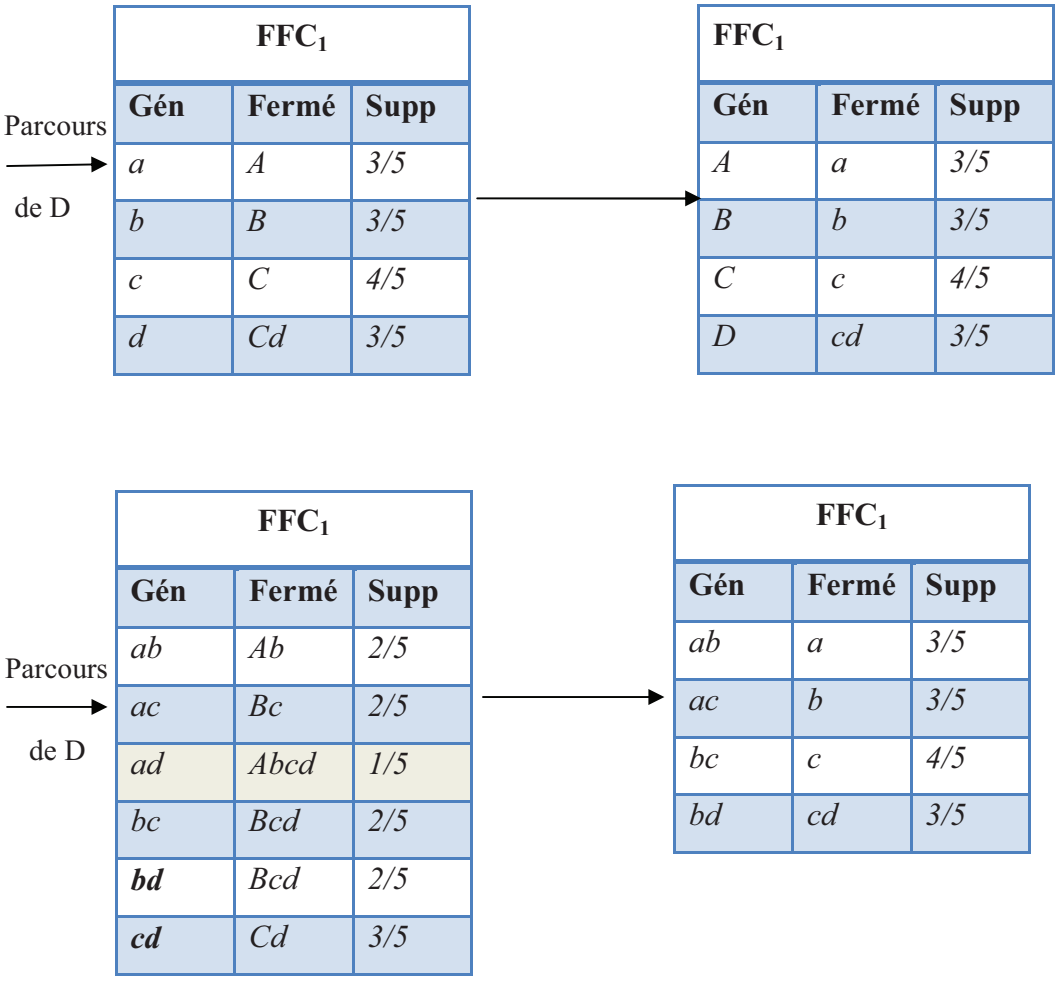


Figure II.2 – Génération des itemsets fréquents fermés à l'aide de Close, minsup= 40%

II.3.1) Notation

Avant que nous expliquions les différents critères de relaxation, nous présentons dans cette section les notations essentielles utilisées pour exprimer touts les définitions.

Une base de données transactionnelle consiste d'un ensemble de transactions *T* sur un ensemble d'items *I*. Pour clarté, chaque item est dénoté comme un caractère en minuscule *a, b, c*,..., et un itemset (ensemble d'item) est dénoté par une séquence de caractères, par exemple, *abc*. Une fonction $I(y), y \in Y$ représente l'itemset contenu par la

transaction y. De la même façon, une fonction $T(x), x \in I$ représente la transaction-set qui contient l'item x. Le support (exact) d'un itemset $X \subseteq I$, $sup(X)$, est le nombre de transactions qui contiennent X. Comme nous avons déjà cité, Un motif sera défini comme une matrice formée par transaction-set et un itemset.

Sans tolérance de faute, un motif est considéré intéressant (fréquent) si toutes ses transactions contiennent tous ses items, et le nombre de transactions est plus grand que le seuil du support minimum *minsup*. Avec tolérance de faute, nous relâchons l'exigence pour toutes les transactions qui contient tous les items, en introduisant des critères de la relaxation [1].

II.3.2) Définitions

II.3.2.1) Relaxation transaction

Un motif $P = Y \times X$ satisfait la relaxation de transaction si toutes ses transactions contiennent la plupart de ses items. Mathématiquement, $\forall y \in Y, |X \setminus I(y)| \leq \varepsilon_t \cdot |X|$, où ε_t est la proportion maximale souhaitée des items manqués par n'importe quel transaction dans P.

Le nom	La Description
T	L'ensemble de transaction
I	L'ensemble des items
minsup	Seuil du support minimum
$\varepsilon_t, \varepsilon_t^c$	Relaxation transaction proportionnelle / constante
$\varepsilon_i, \varepsilon_i^c$	Relaxation item proportionnelle / constante
$\varepsilon_p, \varepsilon_p^c$	Relaxation motif proportionnelle / constante
$SupFt(X, \varepsilon_t, \varepsilon_i, \varepsilon_p)$	Support faute tolérant (FT-support) d'itemset X avec la tolérance donné.

Tab II.1 – Liste de notations

II.3.2.2) Relaxation item

Un motif $P = Y \times X$ satisfait relaxation d'item si tous ses items sont contenus par la plupart de ses transactions. Mathématiquement, $\forall x \in X, \mid Y \setminus T(x) \mid \leq \varepsilon_i \cdot \mid Y \mid$, où ε_i est la proportion maximale souhaitée des transactions manqués par n'importe quel item dans *P*.

II.3.2.3) Relaxation motif

Un modèle $P = Y \times X$ satisfait relaxation motif si le nombre total d'items manquants par l'ensemble des transactions est faible. Mathématiquement, $(\sum_{y \in Y} |X \setminus I(y)|) \leq \varepsilon_p . |X| . |Y|$ ou ε_p est la proportion maximale souhaitée de manque dans *P*.

Remarque : *il faut prendre* $\varepsilon \neq 0$ *, sinon le problème se dégrade au problème sans tolérance de faute. Lorsqu'un critère de relaxation est imposé ou pris en considération cela signifie que la tolérance d'erreur correspondante est inférieure à 1, et vice versa [1]. Par exemple " relaxation transaction est imposée désigne* $\varepsilon_t < 1$*. Lorsque la relaxation est définie comme une valeur constante au lieu de propositionnelle, un exposant "c" est ajouté aux variables, par exemple* ε^c_t, ε^c_i, ε^c_p*. Certaines notations communes sont résumées dans le tableau 3.3 pour plus de clarté [1].*

II.3.2.4) L'approche d'extraction des motifs à tolérance de faute

Ardian et al [1] ont considère un motif intéressant s'il satisfait tous les critères de relaxation. Ils ont défini leur but comme suit :

Définition1. **Motifs fréquents à base tolérance de faute** :

Un motif $P = Y \times X$ est considéré comme un *MFTF* si elle satisfait aux critères de relaxation spécifie, et si la taille relaxation spécifié, et si sa taille, $\mid Y \mid \geq minsup$.

Définition2. **Faute-tolérante support (FT-Support)** :

Le FT-Support d'un itemset *X* est la taille maximale de l'ensemble de tous *MFTF* $Y \times X$. *FT-Support* d'un itemset X relativement a ε_t, ε_i *et* ε_p est notée *SupFt(X,* ε_t, ε_t, ε_t*) ou SupFt(X)* lorsque le contexte est clair.

*Définition3***. Itemset fréquent à base tolérance de faute :**

Un itemset *X* est *IFTF,* si $SupFt(X) \geq minsup$.

Le problème étudiait par [1] est formalise comme suit :

Définition4. **L'extraction d'IFTF**

Étant donné une base de données transactionnelle, les critères de relaxation, et le seuil minimal de support, le problème des mines *IFTF* consiste à énumérer tous *IFTF* avec leur *FT-Support*.

II.3.3) Algorithme d'Adrian et al

Parmi les algorithmes rare qui sont utilises pour extraire les IFTF, on trouve l'algorithme de Adrian et al [1]. Cet algorithme traite le problème des IFTF avec des paramètres de relaxation propositionnel, nous présentons dans la section suivante les problématiques traites par cet algorithme et ses différentes étapes :

II.3.3.1) Problématique

Extraction des IFTF avec leurs supports exactes en appliquant des paramètres de relaxation propositionnelles est considère comme une tache difficile a cause de deux préoccupations :

1- La non anti-monotonie de FT-Support : contrairement au support conventionnel, le FT-support de X : *SupFT (X),* n'est pas anti-monotone.
2- Difficulté de calcule de FT-Support.

Alors deux solutions ont été proposes par [1] :

- *Solution 1 :*

Avec des propriétés ***anti-monotonie***[2], nous pouvons arrêter en toute sécurité l'extension d'un itemset une fois qu'il n'est pas fréquent, donc la complexité des mines est proportionnelle au nombre de motifs fréquents. Ce n'est pas le cas lorsque l'activité minière utilisant la relaxation proportionnelle [3,4].

Pour faire face à ce problème, une fonction *f(X)* est définie. Cette fonction sous-estime jamais le FT-support de *X* ou des ses supersets, et ensuite l'utiliser pour limiter l'espace de recherche chaque fois que nous atteignons itemset *X* où *f(X)* < minsup. L'idée de cette fonction bondissant est illustre par l'exemple suivant :

Exemple 2.3 : *soit la figure 2.3. Ici l'axe des x représente un chemin préfixe d'itemsets, tels que* $A \subset B \subset C$ *, la figure montre 3 fonctions candidats* $f_i(x)$, $f_o(x)$, $f_l(x)$ *dont aucune ne sous-estime SupFT (X). Bien que* $f_i(x)$ *est proche de SupFT(X), elle serait incorrect comme une fonction bondissant, puisque* f_i *(A) <SupFT (B).la conséquence est*

[2] Signifie que si f (X) est vrai pour une fonction f et un ensemble X, alors f (X' ⊆ X) est aussi vrai.

que même si SupFT (A) < f_i (A) <minsup, nous ne pouvons pas tailler la recherche a A, car il y a éventuellement autres itemsets B ⊇ A, avec SupFT (B) ≥ minsup. La fonction souhaitée serait f_o, car lorsque f_o < minsup, nous pouvons directement élaguer l'espace de recherche. Dans cet exemple, nous pouvons directement prune à itemset C. f_l peut également être utilisé comme une fonction de sélection, mais elle est très perdante, et le résultat est un espace de recherche très large.

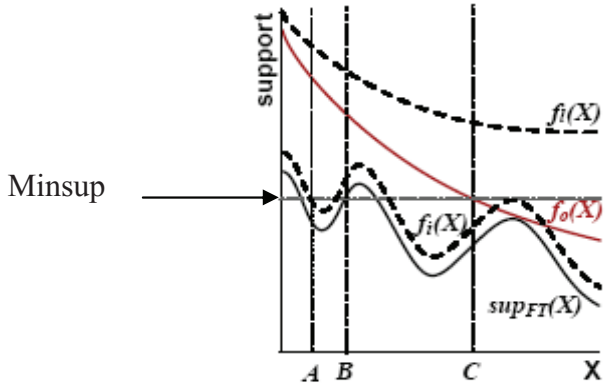


Figure II.3 – Bondissant support non anti-monotone

- *Solution 2 :*

Une tâche intégrante, mais souvent coûteuse dans le processus d'extraction est de déterminer le support d'un itemset donnée. Bien qu'avec les motifs fréquents classiques, nous pouvons simplement compter le nombre de transactions contenant cette itemset, calculant du support de FTTF (même avec une tolérance d'erreur) est connu pour être un problème NP-complet [2]. Pour résoudre ce problème, les auteurs en [1,2] ont transformés le problème en un modèle de programmation linéaire entier (PLE).

II.3.3.2) Structure et Algorithme

La structure proposée par [1], est composé de 3 modules : a) l'énumération de touts les itemsets, b) contrôle de borne, c) calcule du support. Cette structure est présentée dans l'algorithme 1.

Le module d'énumération d'itemsets peut être traité comme un processus de produire des candidats (ligne 8) et choisir quel itemset à explorer. (Ligne 3). La procédure de contrôle de la borne (ligne 4) vérifie si tout superset de *X* (incluant *X*) peut-être fréquent et la procédure de calcule du Support (lignes 5-6) calcule le FT-support exact de X, et return le résultat *X* si son Ft-support est plus grand que ou égale au *minsup*.

Un itemset dont à borne *f (X)* est calculé est considéré comme un candidat bordé, et ce dont le support est plus grand que le support minimum comme candidat fréquent.

Pour accélérer le calcul du support, afin de calculer f (X) et supFT (X), une extraction des classes d'équivalence de transactions en ce qui concerne X [2] est nécessaire. Ce module est dénoté comme extraction des classes d'équivalence.

Algorithme1 : Structure
1 C ←candidate 1-itemset; *2 Tant que C est non vide faire* *3 X←itemset-suivant(C) ; // X est un candidat border* *4 Si f(X) ≥ minsup alors* *5 // X est un candidat fréquent* *6 Si supFt(X) ≥ minsup alors* *7 Output(X,supFt(X)) ; //X est un IFFT* *8 Fin si ;* *9 Update(C, X) ;* *10 Fin tan que ;* *11 Fin.*

II.3.3.2.1) Extraction des équivalences classes

Ce module est introduit en [2] pour accélérer la procédure de calcule de support en utilisant la PL. L'idée ce de réduire le nombre des variables de PL, du nombre de transaction au nombre d'équivalence classe, pour calculer le support. En basent sur les critères des équivalences, il existe 02 type d'équivalence classe :

II.3.3.2.1.1) E_{num}^{X}

Deux transactions sont E_{num}^{X} équivalentes s'ils contiennent le même nombre d'item d'itemset *X*. Mathématiquement : $t_1 \equiv t_2 \leftrightarrow |I(t_1) \cap X| = |I(t_2) \cap X|$. Pour un itemset *X*, le nombre de E_{num}^{X} est $|\ X\ |+1$. La classe d'équivalence pour transactions qui contiennent i items X est dénotée comme E_i^X.

Exemple 2.4 *On considère la base de données transactionnelle suivante :*

TID	Itemset
1	c
2	b, c
3	a, b
4	a, b, c

Soit X = bc. Ici, nous avons trois classes de l'équivalence : $\boldsymbol{E_0^X}$ *= {},* $\boldsymbol{E_1^X}$ *= {1, 3}, et* $\boldsymbol{E_2^X}$ *= {2, 4}.*

II.3.3.2.1.1) $\boldsymbol{E_{iset}^X}$

Deux transactions sont E_{iset}^X équivalentes s'ils contiennent le même ensemble d'item d'itemset *X*. Mathématiquement : $t_1 \equiv t_2 \leftrightarrow I(t_1) \cap X = I(t_2) \cap X$. Pour un itemset *X*, le nombre de E_{iset}^X est $2^{|X|}$. La classe d'équivalence pour transactions qui contiennent itemset Y mais pas X\Y est dénotée comme E_Y^X.

Exemple 2.5 *Considérant la base de données transactionnelle d'exemple 2.4. Soit X = bc. Ici, nous avons trois classes de l'équivalence :* $\boldsymbol{E_\emptyset^X}$ *= {},* $\boldsymbol{E_b^X}$ *= {3},*$\boldsymbol{E_c^X}$ *= {1}, et* $\boldsymbol{E_{bc}^X}$ *= {2, 4}.*

Le meilleur algorithme courant pour extraire $\boldsymbol{E_{num}^X}$ prend *O (|X |² + S (X))* itérations [5], et cela pour extraire $\boldsymbol{E_{iset}^X}$ prend *O* $(|X|^{2^{|X|}} + S(X))$ itérations [6], où *S (X)* est le nombre d'itérations pour calculer le support exact de *X*. donc, extraire $\boldsymbol{E_{num}^X}$ est plus effectif qu'extraire$\boldsymbol{E_{iset}^X}$.

II.3.3.2.2) Contrôle de la borne

L'objectif de ce module est dériver une fonction qui peut borner l'espace de la recherche de IFTF. Donnant un itemset *X*, cette fonction *f (X)* ne doit jamais sous-estimer le support de *X* ou tous ses super sets. Mathématiquement, $f(X) \geq max_{Y \supseteq X} supFT(Y)$. Le fait que le supFT (X) avec toute relaxation constante d'erreur est anti-monotone a été prouvé dans [11], bien que les supFT (X) avec relaxation proportionnel motif et transaction ne sont pas anti-monotones a été tracé dans [3] et [4]. Ces conclusions sont résumées dans Tableau II.2

Relaxation	Anti-Monotone	
	Constante	Proportionnelle
Transaction	AM	Non AM
Item	AM	AM
Motif	AM	Non AM

Tab II.2 – effet de critères de la relaxation sur propriété anti-monotone de support

En basant sur les propriétés d'anti-monotone décrites au dessus, *Adrian et al* [1]. ont dérivé une fonction $f_a(X)$ pour borner l'espace de recherche par défaire la relaxation item avec relaxation motif. Mathématiquement :

$$f_a(X) = supFT(X, \varepsilon_{t=1}, \varepsilon_i=1, \varepsilon_p= \varepsilon_i).$$

II.3.3.2.3) Calcule de support

Cette section présente la façon dont *Adrian et al* [1] ont calculée le FT-support d'un itemset de manière efficace. Avant de présenter l'algorithme, nous présentons leurs analyse de l'impact de chaque contrainte sur la complexité de la procédure de calcule de support. Ce résultat est résumé dans le tableau II.3 [1].

Lorsque seule la relaxation transaction est considérée, le FT-support est simplement le nombre de toutes les transactions dans le motif qui satisfait relaxation transaction. Lorsque seul la relaxation sur le motif est considéré, un algorithme glouton peut être utilise, avec la répétition de la prise des transactions avec moins nombre d'erreurs jusqu'à ce que la contrainte est violée [3, 2].

En calculantE^X_{num}, $\mathrm{E}^{\mathrm{X}}_{\mathrm{iset}}$ ces deux fonctions peuvent être calculées en temps $O(|X|)$ itérations. Lorsque relaxation item est imposée, E^X_{num}, n'est pas suffisante, car nous avons besoin de connaître les erreurs contenues par des items spécifiques (section 3.4.3.2). Ce fait indique la difficulté (NP-complet) de ce problème, qui a officiellement été démontré dans [2] pour le cas de relaxation constant.

Dans cette partie, nous présentons la solution du problème de calcule de support avec la relaxation item présenté par [1] et les différentes heuristiques utilises pour accélérer le calcule du support.

Relaxation	Calcule support	
	Constante	Proportionnelle
Transaction	*O* (|X|)	*O* (|X|)
Item	NPC	NPC
Motif	*O* (|X|)	*O* (|X|)

Tab II.3 – complexité calcule de support

II.3.3.2.3.1) relaxation item constante :

Poernomo et Gopalkrishnan [2] montrent que le problème de calcule de support peut être traduit en un problème de programmation linéaire en nombres entiers (PLE), où les variables représentent les classes d'équivalenceE_{iset}^{X}, et le nombre de contraintes est égal au nombre d'items (|X|). L'élagage se fait comme suit :

Soit *SupFT (X)* correspond au motif $P = Y \times X$ qui satisfait tous les critères de relaxation, et $|Y| = SupFT(X)$. Les paramètres de PLE sont :

1- Soit V= {v_{x1}, v_{x2}, …} un ensemble de variables, tel que $v_{Xi} \in V$ correspond aux nombres de transactions dans Y de la classe d'équivalence E_{xi} (Section 3.4.4.2.1.1).

2- Pour chaque item $a \in X$ on ajoute une contrainte (inégalité) :

$$\sum_{xi} ax_i \quad vx_i \leq \varepsilon_i^c$$

Lorsque $ax_i = 1$ si les transactions associes à la classe d'équivalence E_{xi} ne contiennent pas l'item a et $ax_i = 0$ autrement.

3- La valeur des variables vx_i doit être entre *0* et $|E_{xi}|$ ($0 \leq vx_i \leq |E_{xi}|$) et la fonction objective est de maximiser le nombre de transactions, qui peut être écrit comme suit :

$$max \sum_{xi} vx_i .$$

L'exemple suivant explique bien cette transformation en PLE.

Exemple 2.6 : *Soit itemset ab. On se concentrant sur l'item a, le nombre de transactions qui ne contient pas a est au plus* ε_i^c *(section* 3.4.3.2*). Ce nombre est spécifié par la somme des variables associés à des classes d'équivalence qui ne contiennent pas l'item a. qui est dans ce cas* $E_\emptyset$ ***et*** $E_{\boldsymbol{b}}$ *d'ou on a une inégalité* $v_\emptyset + v_{\boldsymbol{b}} \leq \varepsilon_i^c$*. La même chose pour l'item b on a* $v_\emptyset + v_{\boldsymbol{a}} \leq \varepsilon_i^c$*. Le but est de trouver la taille maximale transactions-items. La fonction objectif a pour but de maximiser la somme de toutes les variables, qui est* $v_\emptyset + v_{\boldsymbol{a}} + v_{\boldsymbol{b}} + v_{\boldsymbol{ab}}$.

Dans le pire des cas, la taille du PLE (nombre de contraintes × nombre de variables) est $|X| \times \min(2^{|X|}, |T|)$ [1] ce qui est assez grand même pour les tailles modérés de X (environ 20). Cependant les transactions qui manquent plus de ε_i^c(ou $\varepsilon_{\boldsymbol{t}} \cdot |\mathbf{X}|$) items peut être ignores, et donc le nombre de variables est réduit à :

$$\sum_{i=0}^{\varepsilon_i^c} \binom{|X|}{i}$$

Par exemple quand |X|=20 et $\varepsilon_i^c = 3$. Le nombre de variables est égal à 1.351, ce qui est petit pour un PLE.

Pour accélérer le traitement du problème PLE, une heuristique a été utilisée qui consiste à réduire le nombre de variables de PLE. Cette heuristique est définie comme suit :

Heuristique : Quand on considère l'item relaxation les transactions qui manquent au plus un item conduit toujours à la taille maximale| *Y* | qui satisfait tous les critères de relaxation. Cette propriété a été prouvée par Adrian et al [1].

En utilisant cette l'heuristique, on peut faire un prétraitement des variables associes aux transactions manquant au plus un item, pour réduire le nombre de variables pour PLE. De plus, si le nombre de transactions contenant un tel item est plus grand que ε_i, la tolérance d'erreur par rapport à l'item correspondant est saturée. Par conséquent, nous pouvons supprimer toutes les variables correspondant à des opérations qui manquent un tel item. Notons que cette propriété n'est pas valable pour les transactions qui manquent deux ou plusieurs items.

Exemple 2.6 *Poursuivant l'exemple 2.5, soit* ε_i^c *=10 et* $|E_b|$*=20. On appliquant l'heuristique, la prise de cette transaction (qui manque un seul item) entraine le choix optimal. Par conséquent, nous pouvons prendre autant que possible, en mettant*

$v_b = 10$. Lorsque $v_b = 10$, la première inégalité (item a) est saturé. Par conséquent, nous ne pouvons pas prendre toutes les transactions qui manquent de plus l'item a :

$$v_{xi} = 0 \; \forall a \notin X_i = 0$$

II.3.3.2.3.2) relaxation item propositionnelle :

Pour calculer le support-FT d'un itemset dans le cas de relaxation d'item proportionnelle, 02 solutions ont été proposé par *Adrian et al* [1].

1) La première solution : consiste à garder la même structure utilisée dans la relaxation item constante (Section 3.4.4.2.3.2). Le problème peut être modélisé par un PLE, en ajoutant un facteur supplémentaire, ε_i, dans les coefficients.

Exemple 2.7 *En continuant l'exemple 2.5, les contraintes de PLE seront transformer au :*

$$v_\emptyset + v_b \leq \varepsilon_i . (v_\emptyset + v_a + v_b + v_{ab}) \quad \boxed{\text{Par rapport à } a}$$

$$v_\emptyset + v_a \leq \varepsilon_i . (v_\emptyset + v_a + v_b + v_{ab}) \quad \boxed{\text{Par rapport à } b}$$

Avec cet effet de changement, l'heuristique a appliqué dans la relaxation constante ne peut pas se faire. En d'autres termes, on ne sait pas combien de transactions pouvons-nous prendre pour chaque itemset manquant un seul item.

2) La deuxième solution : consiste a transformé la contrainte en une valeur constante. Pour se faire cette transformation, Adrian et al [1] ont utilisé un algorithme qui dérive FT-support avec la relaxation proportionnelle, par calcul itérativement FT-support avec la relaxation constant. L'idée de cet algorithme est décrite comme suit :

Soit un itemset X, et soit ε_t, ε_i, ε_p les trois critères de relaxation proportionnelle. Soit S ensemble des transaction-sets réalisable, c.-à-d. ensembles des transaction-set qui vérifient la propriété suivante :

$$\forall s \in S \; supFT\left(X, \varepsilon_t^c = |X|.\ \varepsilon_t, \varepsilon_i^c = s.\ \varepsilon_i, \varepsilon_p^c = s.|X|.\ \varepsilon_p\right) = s$$

Le support-FT exacte de X SupFT(X) =max(s). Une méthode itérative est proposée pour calculer le plus grand s. Cette méthode est présentée dans Algorithme 2.

Une démonstration de l'efficacité de résultat de cette méthode est prouvée par Ardian et al [1] comme suit :

Comme S est la borne supérieure, si S' = S, alors S est la plus grande valeur qui est réalisable, ce qui implique S est le FT-support (ligne 5). Sinon, si S' <S, alors le FT-support doit être inférieur ou égal à S', car S' est monotone à S.

Algorithme2 : la Méthode itérative proposé par Ardian et al, pour calculer support (prop. relaxation) en l'utilisation le support (Const. relaxation)
Entrée: itemset X, ε_t, ε_i, ε_p *Sortie: SupFT(X)* *1 S ← B.S //* borne supérieure 2 Tant que vrai faire 3 s'= $supFT\left(X, \varepsilon_t^c = \lvert X\rvert .\ \varepsilon_t, \varepsilon_i^c = s.\ \varepsilon_i, \varepsilon_p^c = s.\lvert X\rvert .\ \varepsilon_p\right)$; 4 si s=s' alors 5 return s; 6 sinon 7 s←s' ; 8 fin si ; 9 fin ;

Tout d'abord, nous initialisons S comme borne supérieure de la FT-support (ligne 1). Une approche simple et assez bonne est en relâchant item relaxation par motif relaxation [1], donc UB =$supFT\left(X,\ \varepsilon_t,\ \varepsilon_i = 1, \min\left(\ \varepsilon_i, \varepsilon_p\right)\right)$. Ayant obtenu la borne supérieure de S, nous itérative vérifier si S est réalisable (ligne 4). Si c'est le cas, nous avons obtenu le résultat, sinon, nous plus bas de l'estimation.

II.3.3.3) La complexité globale

Soit N_{cb} et N_{cf} le nombre des candidats border et des candidats fréquents respectivement; T_{ie} est le temps total pris pour l'itération des itemsets; T_{num}(ou T_{iset}), T_b, T_{sc} être le temps pris pour l'extraction des classes d'équivalence de chaque itemset, de contrôle de la borne, et au calcule du support respectivement.

Le temps nécessaire estime par *Ardian et al* [1] pour extraire tout IFFT est donné par :

$$T_{ie} + N_{cb} \cdot (T_{num} + T_b) + N_{cf} \cdot (T_{iset} + T_{sc}).$$

II.4) Evaluation des 3 Algorithmes

Afin d'évaluer les 3 Algorithmes d'extraction des motifs fréquents, qui sont présentés dans ce chapitre, nous présentons l'implémentation de ces 03 algorithmes sur la même base de donnée transactionnelle. Cette évaluation concernent 02 axes : 1) le nombre de motifs fréquents générés et 2) le temps de réponse pour chaque algorithme.

La caractéristique de jeu de donnée utilisé pour évaluer est présentée dans le tableau II.4. C'est un jeu de donnée Mushrooms[3] décrivant les caractéristiques de champignons [7]. Pour les 02 premiers algorithmes (Apriori et Close), nous présentons l'évaluation étudiée dans [8], et pour l'algorithme de calcule de IFTF, nous présentons l'évaluation étudiée dans [1].

Mushrooms	**Les Caractéristiques**	
	$\|T\|$	$\|I\|$
	8,124	119

Tab II.4 – Caractéristiques de base de données Mushrooms

Le minsup et les paramètres de relaxation (Pour la proche IFTF) utilisées sont présentés dans le tableau II.5.

Le temps de réponse, et le nombre des items fréquents pour chaque approche sont présentés dans le tableau II.6

Les paramètres			
minsup	ε_t	ε_i	ε_p
20 %	30 %	4 %	2 %

Tab II.5 – les paramètres d'évaluation

[3] http://fimi.ua.ac.be/data/mushroom.dat

	Temps d'exécution (s)	Nombre des items fréquent
Apriori	115,82	53 337
Close	9,63	53 337
Faut tolèrent	86,03	53563

Tab II.6 – le temps de réponse et le nombre des motifs fréquents

En comparant les deux méthodes classiques Close et Apriori, Close a retourné le même nombre de motifs fréquents comme Apriori dans un temps très court par rapport à cette dernière, car elle utilise le mécanisme de fermeture pour réduire le nombre de calculs des supports. Par contre les résultats montrent que malgré la grande consommation du temps par l'approche Faute tolèrent en comparant avec Close, sauf qu'elle a retourné un nombre plus grand des motifs fréquents grâce à la relaxation de la propriété de support introduit dans les 02 autre approches .Ces résultats montrent l'efficacité de l'approche de faute tolèrent dans le domaine de la découverte des motifs fréquents.

Conclusion

Les techniques d'extractions des motifs fréquents sont utilisées dans plusieurs domaines, et leur utilisation a appliqué aussi dans le domaine de conception physique des entrepôts de donnée.

Nous avons présenté dans ce chapitre les 02 algorithmes classiques qui sont utilisés dans la découverte des motifs fréquents, après nous avons détaillé l'approche d'extraction des motifs de faute tolèrent, qui est basé sur la notion de la relaxation, et nous avons présenté une comparaison avec les 02 autres approches Close et Apriori, qui ont montré l'efficacité de cette approche par rapport aux autres algorithmes classiques.

Chapitre III

Les index de jointure binaire

Introduction

Une des caractéristiques des entrepôts de données est leurs volume de données important qui peut atteindre quelques milliers de giga-octets (téraoctet).Comme nous avons cité dans chapitre I, Les entrepôts de données sont dédiés aux applications d'analyse et de prise de décision. Cette analyse est souvent réalisée par l'intermédiaire de requêtes complexes caractérisées par des opérations de sélections, de jointures et d'agrégations. Exécuter de telles requêtes sur un entrepôt volumineux nécessite un temps de réponse très élevé qui n'est pas acceptable par les décideurs qui exigent un temps de réponse raisonnable afin de répondre aux besoins décisionnels. Pour réduire ce dernier et satisfaire les besoins des décideurs, l'administrateur (ou le concepteur) de l'entrepôt est amené à faire une bonne conception physique, où il doit sélectionner un ensemble de techniques d'optimisation (voir Chapitre 1) qu'il considère pertinent pour l'ensemble des besoins des décideurs (exprimés sous forme de requêtes). Parmi les techniques d'optimisation les plus utilisés, on trouve les index de jointure binaires qui servent à pré-calculer la jointure entre la table des faits et une ou plusieurs tables de dimension.

Pour bien exprimer les différents concepts cités au dessus, Nous avons organisé ce chapitre en 3 sections. La section 1 présente la technique d'optimisation : les index en particulier les index de jointure binaire. La section 2 donne la stratégie d'exécution d'une requête de jointure en étoile en présence d'un IJB, et nous étudions la complexité d'une sélection d'une configuration index. La section 3 présente le modèle de coût que nous avons choisi pour évaluer les différentes techniques d'optimisation des IJB présentes dans notre travail.

III.1) La techniques d'optimisation redondantes : les index

Un index est une structure redondante ajoutée à la base de données pour permettre les accès rapides aux données. Il permet à partir d'une clé d'index de trouver l'emplacement physique des n-uplets recherchés. Parmi les techniques d'indexation proposées dans le cadre des bases de données classiques, nous pouvons citer l'index B-tree, l'index de hachage, l'index de projection, l'index de jointure, etc. La majorité de ces index est aussi utilisée dans le cadre des entrepôts relationnels. Certaines techniques d'indexation sont apparues dans le contexte d'entrepôts de données comme les index binaires, les index de jointure binaires, les index de jointure en étoile, etc. Nous

pouvons classer les index proposés en deux catégories, les index mono-table et les index multi-tables. Les index mono-table sont des index définis sur un ou plusieurs attributs de la même table, comme les index B-tree, de hachage, binaires, etc. Les index multi-tables sont des index définis sur plusieurs tables comme les index de jointure standards, en étoile et binaires.

III.1.1) Index B-arbre

L'index B-arbre est l'index par défaut pour la plupart des SGBD commerciaux [25]. Cet index est organisé sous forme d'arbre à plusieurs niveaux. Chaque noeud d'un niveau pointe vers le niveau inférieur. Les nœuds feuilles (niveau le plus bas) contiennent les entrées d'index ainsi qu'un pointeur vers l'emplacement physique de l'enregistrement correspondant (généralement un identifiant physique, ROWID).

III.1.2) Index de hachage

L'index de hachage repose sur l'utilisation d'une fonction de hachage. Cette fonction permet, à partir d'une valeur de clé c, de donner l'adresse f(c) d'un espace de stockage où l'élément doit être placé [12].

III.1.3) Index binaire (bitmap index)

L'index binaire repose sur l'utilisation d'un ensemble de vecteurs binaires (contenant des valeurs 0 ou 1) pour référencer l'ensemble des n-uplets d'une table. Pour chaque valeur de l'attribut indexé, un vecteur de bits dit bitmap est stocké. Ce vecteur contient autant de bits qu'il y a de n-uplets dans la table indexée. L'index binaire a été considéré comme le résultat le plus important obtenu dans le cadre de l'optimisation de la couche physique des entrepôts de données [26].

III.1.4) Index de jointure

L'opération de jointure est toujours présente dans les requêtes OLAP. Elle est très coûteuse, puisqu'elle manipule de grands volumes de données. Plusieurs implémentations de la jointure ont été proposées dans les bases de données traditionnelles : les boucles imbriquées, les fonctions de hachage, la tri-fusion, etc. Ces implémentations sont limitées lorsque la taille des tables concernées par la jointure est importante. L'index de jointure matérialise les liens existant entre deux tables en utilisant une table à deux colonnes chacune représentant l'identifiant d'une table [12].

III.1.5) Index de jointure binaire

L'index de jointure binaire (IJB) est une variante des index de jointure. Il constitue une combinaison entre l'index de jointure et l'index binaire. Il a été proposé pour pré-calculer les jointures entre une ou plusieurs tables de dimension et la table des faits dans les entrepôts de données modélisés par un schéma en étoile [27]. Au contraire des index binaires standards où les attributs indexés appartiennent à la même table, l'IJB est défini sur un ou plusieurs attributs appartenant à plusieurs tables. Plus précisément, il est défini sur la table des faits en utilisant des attributs appartenant à une ou plusieurs tables de dimension.

III.1.5.1) La construction de l'index de jointure binaire IJB

Supposons un attribut *A* ayant *n* valeurs distinctes $v_1, v_2, ..., v_n$ appartenant à une table de dimension *D*. Supposons que la table des faits *F* est composée de *m* instances. La construction de l'index de jointure binaire IJB défini sur l'attribut *A* se fait de la manière suivante [12] :

1. Créer n vecteurs composés chacun de m entrées ;
2. Le $i^{ème}$ bit de vecteur correspondant à une valeur v_k est mis à 1 si le n-uplet de rang i de table des faits est joint avec un n-uplet de la table de dimension D tel que la valeur de A de ce n-uplet est égale à v_k. il est mis à 0 dans le cas contraire. Plus formellement : $IJB^k_j=1$ si $\exists td \in D$ tel que $tf_j.D_{id} = td.D_{id} \wedge td.A = v_k$ ou IJB^j_k, tf_i, td, D_{id} représentent respectivement le j^{me} bit de vecteur correspondant à la valeur v_k, le n-uplet de *F* de rang *j*, un n_uplet de table *D*, la clé étrangère de *D*.

Exemple 3.1 : *Pour comprendre la construction des index de jointure, considérons l'exemple de la figure III.1. Cette figure représente l'index de jointure binaire Client_Ville_IJB_idx construit sur la table ventes en utilisant l'attribut Ville. Cet index peut être construit par la commande SQL suivante :*

```
CREATE BITMAP INDEX Client_Ville_BI_idx
ON Vente(Client.Ville)
FROM Ventes V, Client C
WHERE V.CID= C.CID
```

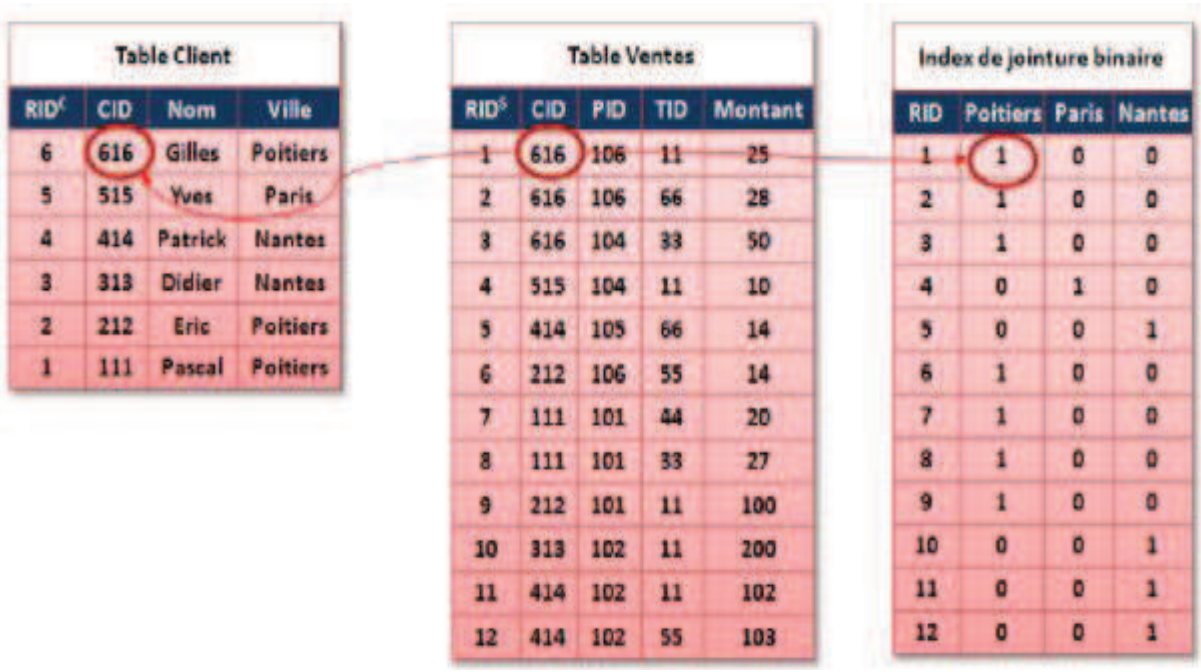


Table Client

RID	CID	Nom	Ville
6	616	Gilles	Poitiers
5	515	Yves	Paris
4	414	Patrick	Nantes
3	313	Didier	Nantes
2	212	Eric	Poitiers
1	111	Pascal	Poitiers

Table Ventes

RID	CID	PID	TID	Montant
1	616	106	11	25
2	616	106	66	28
3	616	104	33	50
4	515	104	11	10
5	414	105	66	14
6	212	106	55	14
7	111	101	44	20
8	111	101	33	27
9	212	101	11	100
10	313	102	11	200
11	414	102	11	102
12	414	102	55	103

Index de jointure binaire

RID	Poitiers	Paris	Nantes
1	1	0	0
2	1	0	0
3	1	0	0
4	0	1	0
5	0	0	1
6	1	0	0
7	1	0	0
8	1	0	0
9	1	0	0
10	0	0	1
11	0	0	1
12	0	0	1

Fig. III.1 – Index de jointure binaire

Le premier n-uplet de la table Ventes est joint avec un n-uplet de la table Client correspondant à un client habitant Poitiers. Par conséquent, la case correspondante à la ville de Poitiers de la première ligne de cet index est mise à 1, les deux autres cases sont mises à 0.

III.1.5.2) Stratégie d'exécution en présence des IJB

Une requête de jointure en étoile est caractérisée par un ensemble de sélections sur les tables de dimension, suivies de jointures avec la table des faits. Si tous les attributs objets des prédicats de sélection sont indexés, alors l'exécution d'une requête de jointure passe par les étapes suivantes [12]:

1. Effectuer une réécriture de la requête qui consiste à séparer chaque jointure dans une sous-requête. Chaque sous-requête représente l'ensemble des sélections effectuées sur chaque table de dimension.

2. Pour chaque sous-requête, utiliser les IJB définis sur la table de dimension pour trouver un vecteur de bits représentant les n-uplets de la table des faits référencés par la sous-requête.

3. Effectuer une opération AND entre les vecteurs obtenus à partir des sous-requêtes pour trouver un seul vecteur référençant tous les n-uplets référencés par la requête.

4. Utiliser le vecteur résultat pour accéder à la table des faits et récupérer les n-uplets référencés par la requête globale.

III.2) Problème de sélection des index de jointure binaires

Nous présentons dans cette section une formalisation du problème de sélection des IJB, la complexité de sélection une configuration IJB.

III.2.1) Formalisation

La sélection d'une configuration d'IJB vise à optimiser la performance d'un ensemble de requêtes de jointure en étoile. Nous considérons l'espace de stockage réservé aux index comme une contrainte du problème de sélection des IJB. En conséquence, le problème de sélection d'une configuration d'IJB peut être formalisé comme suit [12] :

Étant donné :

- Un entrepôt de données modélisé par un schéma en étoile ayant un ensemble de tables de dimension $D = \{D_1, D_2, ..., D_d\}$ et une table des faits F,

- une charge de requêtes $Q = \{Q_1, Q_2, ..., Q_m\}$, où chaque requête Q_j a une fréquence d'accès f_j, et

- Une capacité de stockage S ;

Le problème de sélection des index de jointure consiste à trouver une configuration d'index CI minimisant le coût d'exécution de Q en respectant la contrainte de stockage $(Taille(CI) \leq S)$.

III.2.2) Complexité

La sélection d'une configuration d'IJB est généralement une tâche difficile comparée à d'autres types d'index. Cela est dû à plusieurs considérations :

– Un IJB est définis sur un ensemble d'**attributs indexable** qui représentent un sous-ensemble des attributs non clés des tables de dimension. Un attribut est indexable s'il est utilisé par un prédicat de sélection. Dans le contexte d'entrepôts de données, le

nombre d'attributs indexables peut être important, vu le nombre de tables de dimension et le nombre d'attributs non clés de chaque table.

– Un IJB peut être défini sur un attribut (mono-attribut) ou sur plusieurs attributs issus de différentes tables de dimension (multi-attributs), ce qui augmente le nombre d'index de jointure possibles.

– Un attribut indexable peut figurer dans plusieurs IJB car ces derniers peuvent être non -disjoints.

– La taille d'un IJB est proportionnelle à sa cardinalité. Un attribut de forte cardinalité rend l'index volumineux, donc difficile à stocker et à maintenir.

Soit $A = \{A_1, A_2, ..., A_K\}$ un ensemble d'attributs indexables candidats pour la sélection d'une configuration d'IJB. Chaque IJB dans cette configuration est constitué d'un sous-ensemble d'attributs de A, par conséquent cette configuration constitue une partition de *A* en un ensemble de groupes. Chaque groupe d'attributs représente un IJB potentiel. Nous pouvons considérer deux scenarii :

1. *Sélection d'un seul index de jointure : si on veut utiliser un seul IJB, le nombre de possibilité est donné par :*

$$N = \binom{k}{1} + \binom{k}{2} + \cdots + \binom{k}{k} = 2^k - 1$$

Si le nombre d'attributs indexables est égal à 5 (K = 5), alors le nombre d'index possible est égal à 31.

2. *Sélection de plus d'un index de jointure : pour sélectionner plus d'un IJB, le nombre de possibilités est donné par :*

$$N = \binom{2^k - 1}{1} + \binom{2^k - 1}{2} + \cdots + \binom{2^k - 1}{2^k - 1} = 2^{2^k - 1}$$

Par exemple, si le nombre d'attributs indexables est égal à 5, alors N = 2^{31}> (1, 2 × 109).

III.2.3) Travaux de sélection d'une configuration index

Les travaux existant pour la sélection d'une configuration d'index comportent généralement deux étapes :

1. La détermination des attributs candidats à l'indexation : qui peut être faite manuellement par un administrateur ou automatiquement.
2. la construction d'une configuration d'index : construire un ensemble d'index à partir des attributs indexables candidats et l'ensemble des requêtes définies sur l'entrepôt de données.

Plusieurs travaux ont été proposes dans la sélection automatique des configurations d'index, pour réduire la complexité de sélection de la configuration d'index, qui est base sur l'élagage de l'espace de recherche .parmi ces travaux il existe deux qui se basent sur l'approche d'extraction des motifs fréquents pour élaguer l'espace de recherche index. Nous présentons dans cette section la structure de ces 02 approches car l'approche proposé dans ce cadre de travail utiliser la même technique.

III.2.3.1) travaux de Aouiche

Le travail d'Aouiche et al. [28] est parmi les rares travaux qui traitent le problème de sélection des IJB dans le contexte d'entrepôts de données modélisés par un schéma en étoile. L'approche proposée se base sur une technique de recherche des motifs fréquents fermés pour élaguer l'espace de recherche des index de jointure ils utilisant l'algorithme Close pour les générer. Un algorithme glouton est utilisé pour la sélection d'une configuration d'index. La figure III.2 représente les principales étapes de l'approche proposée.

Dans une 1ère étape une charge de requêtes est exploitée afin d'en extraire une configuration d'index améliorant le temps d'accès aux données. Le reste des étapes ont procédé comme suit :

1. *Extraction de la charge de requêtes :* la charge de requêtes est extraite à partir du journal des transactions sauvegardé et maintenu automatiquement par le SGBD.
2. *Analyse de la charge :* la charge de requête obtenue est analysé afin d'extraction l'ensemble des attributs indexables. Ces attributs sont ceux qui font l'objet de prédicats de sélection dans les clauses WHERE des requêtes.
3. *Construction d'un contexte de recherche des motifs fermés :* Soit un ensemble des requêtes $Q= \{Q_1, Q_2,\ldots, Q_m\}$, et un ensemble des attributs d'index candidats $A= \{A_1, A_2,\ldots, A_l\}$, le contexte d'extraction est une matrice qui a des lignes représentant les requêtes, et des colonnes représentant les attributs indexables. L'existence d'un attribut indexable dans une requête est symbolisée par un et son absence par zéro.

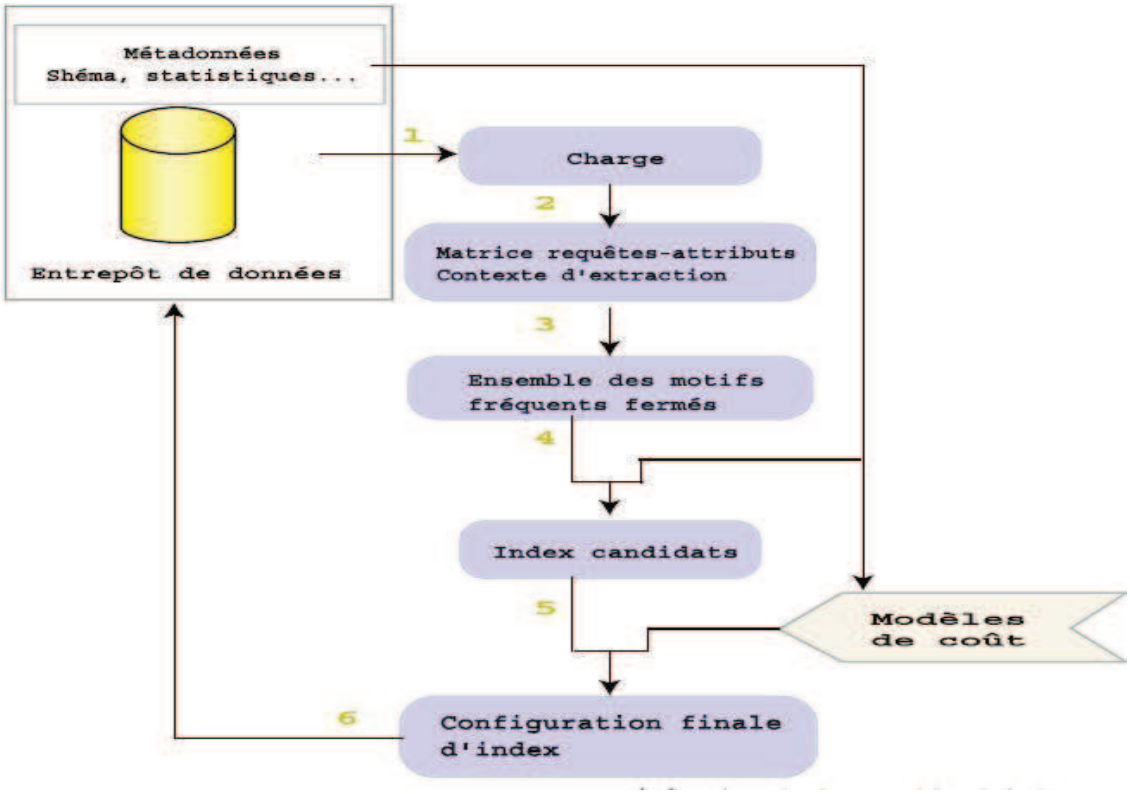


Fig. III.2 – Architecture de fonctionnement de l'approche de Aouiche

4. *Application de l'algorithme close sur ce contexte :* l'algorithme CLOSE est appliqué sur le contexte d'extraction afin d'extraire l'ensemble des motifs fréquents. Chaque motif extrait est composé d'un ensemble d'attributs de l'entrepôt de données. Un motif possède la forme suivante : < Table1.Attribut$_i$, Table2.Attribut$_j$, …, Tablen.Attributk >.
5. *Construction de l'ensemble des index candidats :* l'ensemble des index candidats est construit à partir des motifs fréquents fermés résultant de l'application de l'algorithme CLOSE. Le but de cette étape est de vérifier si les attributs contenus dans chaque motif fréquent permettent de créer un IJB sur l'entrepôt.
6. *Construction de la configuration d'index finale :* A partir de l'ensemble d'index généré dans l'étape précédente, un algorithme glouton est appliqué pour sélectionner une configuration d'index finale. Cet algorithme procède en plusieurs itérations. Dans la première itération, la fonction objective est calculée pour chaque index candidat. L'index I_{max} maximisant la fonction objectif est choisi pour former la configuration initiale. Durant chaque itération, un nouvel index vérifiant la même condition est ajouté à la configuration courante. L'algorithme s'arrête dans les cas suivants :

- ✓ Aucune amélioration de la fonction objectif n'est possible ;
- ✓ Tous les index ont été sélectionnés ;
- ✓ L'espace de stockage disponible est saturé.

La fonction objective utilisée repose sur un modèle de coût qui permet de calculer la taille des index sélectionnés ainsi que le coût d'exécution des requêtes en présence de ces index.

III.2.3.2) l'approche de Bellatreche et al.

III.2.3.2.1) Motivation

Les travaux de Bellatreche et al. [29] Présentent une amélioration des travaux de Aouiche et al [28]. Ils ont montré par un contre exemple les limites de l'approche Aouiche qui utilise uniquement la fréquence des attributs d'élaguer l'espace de recherche pour le problème de sélection IJB. En effet, les IJB sont créés pour optimiser des jointures entre la table des faits et les tables de dimension. En utilisant l'approche de Aouiche et al. , l'algorithme peut éliminer des index sur des attributs non fréquemment utilisés mais qui appartiennent à des tables de dimension volumineuses, ce qui ne permet pas d'optimiser une opération de jointure. Pour pallier ce problème, Bellatreche et al. Proposent d'inclure d'autres paramètres dans la génération des motifs fréquents comme la taille des tables de dimension, la taille de la page système, etc.

III.2.3.2.2) DynaClose et DynaCharm

Pour balayer la limitation de l'approche d'Aouiche et al. Les auteurs en Bellatreche et al. [28] proposent deux algorithmes DynaClose et DynaCharm qui constituent une adaptation des algorithmes Close et Charm. Au lieu d'utiliser le support comme critère de détermination des motifs fréquents, les algorithmes proposés reposent sur une fonction fitness permettant de pénaliser chaque motif fréquent en prenant en compte les paramètres cités ci-dessus.

Pour un motif fréquent m_i, cette fonction est définie comme suit :

$$Fitness(m_i) = \frac{1}{n} \times \left(\sum_{j=1}^{n} \alpha_i \times sup_j \right)$$

Où n représente le nombre d'attributs non clés A_j dans m_i. sup_j représente le support de A_j et α_j est un paramètre de pénalisation défini par l'équation suivante :

$$\alpha_j = \frac{|D_j|}{|F|}$$

Où $|D_j|$, $|F|$ représentent respectivement le nombre de page nécessaire pour stocker la table de dimension D_j et la table des faits F^1.

Etant donné un support minimum *minsup*, une valeur minimum de la fonction fitness *minfitness* est calculée comme suit :

$$minfit = \frac{minsup}{|F|} \times \left\lceil \left(\sum_{j=1}^{d} \frac{|D_j|}{d} \right) \right\rceil$$

où *d* représente le nombre de tables de dimension.

III.3) Modèle de coût

Nous avons vu dans la section III.2.2 que le problème de sélection d'une configuration IJB est NP-Complet [30], d'où la nécessité de définir des algorithmes heuristiques pour résoudre ce problème. Ces algorithmes parcourent un ensemble d'index candidat possibles et sélectionnent le meilleur index. L'estimation de la qualité de chaque index nécessite la définition d'un modèle de coût qui guide la sélection des index.

III.3.1) les types de modèle de cout utilisé

Les modèles sont classes en deux catégories :

1- modèle basé sur une fonction mathématique adhoc : La première catégorie de modèles [31,32] définit une fonction de coût qui prend en entrée une requête et un plan d'exécution et donne en sortie le coût de ce plan. Ce coût est calculé en utilisant des formules qui prennent en compte un certain nombre de paramètres et de statistiques collectées sur la base de données. Pour simplifier l'élaboration de la fonction de coût, un certain nombre de d'hypothèses sont considérées.

2- modèle basé sur l'optimiseur des requêtes : ils font appel à l'optimiseur de requêtes du SGBD utilisé pour estimer le coût d'une requête [33,34]. L'optimiseur reçoit la requête, il évalue les différents plans d'exécution et retourne le meilleur plan avec son coût. Le fait d'utiliser l'optimiseur de requêtes rend le calcul du coût plus fiable mais engendre deux inconvénients majeurs : (1) le coût que l'optimiseur estime dépend du SGBD utilisé et (2) faire souvent appel à l'optimiseur engendre un coût d'exécution supplémentaire et dégrade la qualité de l'optimiseur qui passe plus de temps à estimer le coût d'exécution des requêtes qu'à les exécuter.

[1] Le nombre de pages occupé par une table D est calculé comme suit : $|D| = \left\lceil \frac{\|D\| \times LD}{Ps} \right\rceil$, où ||D||, LD, Ps représentent le nombre de instances de T, la longueur d'une instance de T et de la taille de page, respectivement.

Dans le cadre de notre travail, nous avons choisit d'utilisé le modèle de cout théorique. Nous avons utilisé le même modèle de coût proposé dans [12].

III.3.2) principe de modèle de cout théorique

Nous détaillons dans cette section les différents paramètres et formules de modèle de cout qui nous avons utilisé

III.3.2.1) Paramètres utilisés dans le modèle de coût

Dans ce modèle de coût trois types de données sont utilisé comme paramètres : (1) données de l'entrepôt, (2) données du système physique et (3) données de la charge de requêtes [12].

Les données de l'entrepôt englobent un nombre de paramètres comme le nombre de n-uplets de chaque table, la taille de chaque n-uplet, le nombre de pages stockant une table, etc. Certains paramètres physiques sont aussi nécessaires pour l'élaboration de notre modèle de coût comme la taille du buffer et la taille de la page système. La taille du buffer est très importante pour sauvegarder les résultats intermédiaires en mémoire et réduire ainsi le nombre d'E/S nécessaires. La taille de la page système sert à calculer le nombre de pages nécessaires pour stocker une table ou un résultat intermédiaire. Les données de la charge de requêtes concernent les fréquences d'accès, les facteurs de sélectivité des prédicats de sélection, les facteurs de sélectivité des jointures, etc. Le tableau III.1 montre les principaux paramètres utilisés dans nos formules de coût.

Paramètre	Signification
I	Un index de jointure binaire
|RowId|	Taille de l'identificateur de ligne en bits
$\|A_j\|$	Cardinalité de l'attribut Aj.
|F|	Cardinalité de la table des faits F
||F||	Nombre de pages nécessaires pour stocker F
Taille(I)	Taille d'un index I
PS	Taille de la page système
D_i	i^{eme} table de dimension
B	Taille de buffer en nombre de pages
TS_R	taille d'un n-uplet de la table *R*
fs(R ⋈ S)	facteur de sélectivité de la jointure entre *R* et *S*
|R ⋈ S|	nombre de n-uplets du résultat de la jointure entre *R* et *S*

Tab. III.1 – Paramètres utilisés dans le modèle de coût

III.3.2.2) Coût de stockage d'un IJB

Un IJB construit sur un ensemble d'attributs de dimension, stocke pour chaque n-uplet de la table des faits son identificateur de ligne (*RowID*) ainsi qu'un ensemble de vecteurs binaires représentant chacune une valeur des attributs indexés. L'espace de stockage d'un IJB dépend de deux paramètres : le nombre de n-uplets de la table des faits et la cardinalité des attributs indexables

Soit un index *I* définis sur n attributs *A1, A2, ..., An*. L'espace de stockage en octets de *I* est calculé par la formule suivante :

$$Tatlle(I) = \frac{(|RowId| + \sum_{j=1}^{n} |Aj|) \times |F|}{8}$$

III.3.2.3) Coût d'exécution

L'exécution d'une requête dans le contexte de base de données ou entrepôt de données centralisés engendre deux coûts différents : (1) coût de chargement et de sauvegarde de données via ou sur le disque qu'on appelle le coût d'entrées/sorties et (2) le coût CPU de calcul. Étant donné que le coût CPU est nettement inférieur au coût d'E/S, la plupart des modèles de coût proposés le considèrent négligeable.

III.3.3) Les Scénarios de cout d'exécution

Le modèle de coût permet donc à partir d'une requête et d'un index de donner le nombre d'E/S nécessaires pour l'exécution de cette requête sur un entrepôt de données selon l'index de jointure. Selon les attributs indexés dans CI, nous pouvons considérer t rois scenarios pour l'exécution de Q : (1) Aucun attribut indexable de Q n'est couvert par CI, (2) tous les attributs indexables de Q sont couverts par CI et (3) quelques attributs indexables de Q sont couverts par CI.

III.3.3.1) Scénario 1

Aucun attribut de Q n'est indexé dans CI : Dans le cas d'absence d'IJB utilisés par Q, toutes les jointures de Q peuvent être calculées en appliquant des méthodes classiques comme la jointure par hachage. [Annexe A]. Le cout d'exécution d'une requête est compose de 3 couts [12] :

1- Cout de la première jointure : ce coût correspond au coût de jointure par hachage entre la table des faits et la première table de dimension D_{min}[2]. Ce coût est donné par :

$$C_{PJ} = 3 \times (\|F\| \times \|D_{min}\|)$$

2- *Coût de calcul des résultats intermédiaires :* ce coût correspond au coût d'une jointure par hachage d'un résultat intermédiaire courant et une table de dimension pour former un autre résultat intermédiaire. Ce coût est composé du coût de hachage de la table de dimension ainsi que le coût de sauvegarde et de chargement du résultat intermédiaire précédant. Une fonction *Tdisp* a utilisé pour détermine si le tampon est suffisant pour sauvegarder le résultat intermédiaire ou non. Elle possède la signature suivante : *TDisp (B, RES) : entier* où *B* désigne la taille du tampon et *RES* est un résultat intermédiaire. Cette fonction est définie comme suit : $Tdisp(B, RES) = \begin{cases} 1 \text{ si } \|RES\| \leq B \\ 0 \quad \text{sinon} \end{cases}$

Le coût de calcul des résultats intermédiaires est donné par la formule suivante :

$$C_{RI} = \sum_{j=2}^{k} \left[2 \times Tdisp(B, RES_i) \times \left(\|RES_{j-1} - B + 1\| + 3 \times \|D_{min_j}\| \right) \right]$$

Où D_{min_j} désigne la table de dimension impliquée par la jointure de sélectivité minimum.

3- *Coût des groupements et des agrégations :* Le coût des agrégations et des groupements dépend de la taille du dernier résultat intermédiaire RES_k. Si ce résultat tient en mémoire, nous supposons que ces deux opérations se font en mémoire (aucun coût n'est ajouté). Dans le cas contraire, un coût de sauvegarde et de chargement des données qui ne peuvent pas être gardées en mémoire, dois être ajouté. La formule de coût est alors comme suit :

$C_{AG} = n \times Tdisp(RES_k) \times (|RES_k| - B + 1)$, où *n* est un entier qui dépend du type de la requête, il est défini comme suit :

– *n = 2* si la requête contient seulement des agrégations

– *n = 4* si la requête contient des groupements et des agrégations.

[2] La table de dimension qui a le plus petit facteur de sélectivité

III.3.3.2) Scénario 2

Tous les attributs de Q sont indexés dans CI : ce cas représente la situation idéale où toutes les jointures dans Q ont été pré-calculées dans CI. L'exécution de Q dans ce cas passe par deux étapes importantes : le chargement des index, ensuite l'accès aux données. Par conséquent, deux coûts sont considérés : le coût de chargement des index et le coût d'accès aux données.

- *Coût de chargement des index :* le coût de chargement d'un IJB noté CC(I) correspond au nombre de pages lues pour le charger. Il est calculé par la formule suivante :

$$CC(I) = \frac{Taille(I)}{PS}$$

Le coût de chargement de l'ensemble d'index utilisés par une requête Q correspond à la somme des coûts de chargement de ces index.

- *Coût d'accès aux n-uplets :* soit N_t le nombre de n-uplets de la table de faits référencés par la requête Q. Le coût de lecture (CL) de ces n-uplets est donné par la formule suivante :

$$CL = \|F\| \left(1 - e^{-\frac{N_t}{\|F\|}}\right)$$

Où ||F|| désigne le nombre de pages nécessaires pour stocker la table des faits F.

III.3.3.3) Scénario 3

Quelques attributs de Q sont indexés dans CI : Dans ce scénario, l'exécution de Q se fait en deux phases. Dans la première phase, les index utilisés par Q sont chargés et utilisés pour trouver un ensemble de n-uplets de la table des faits. Le coût de cette phase est calculé comme celui du scénario 2. Il correspond au coût de chargement des index utilisés ainsi que celui de chargement des n-uplets de faits. Dans la deuxième phase, les jointures non encore effectuées (à cause de l'absence d'IJB les pré-calculant dans CI) sont réalisées. Ces jointures sont effectuées entre les n-uplets de faits, résultats de la première étape, et les tables de dimension non encore jointes. Le coût de cette étape est calculé en utilisant le modèle que nous avons utilisé dans le scénario 1.

Conclusion

Les IJB sont des structures redondantes permettant de pré-calculer les jointures dans un entrepôt de données modélisé par un schéma en étoile. Ces index sont caractérisés par une représentation binaire permettant l'utilisation des opérations logiques pour évaluer des conjonctions ou disjonctions de prédicats contenus dans les requêtes de jointure en étoile. Ces index sont généralement recommandés pour les attributs de faible cardinalité.

Nous avons étudié dans ce chapitre le problème de sélection d'une configuration d'IJB sous une contrainte d'espace de stockage. Nous avons formalisé la sélection d'IJB comme un problème d'optimisation et nous avons décrit sa complexité par rapport au nombre d'attributs indexables.

Nous avons détaillé le modèle de coût utilisé en présentant les différentes formules et les scénarios possibles.

Chapitre IV

Sélection configuration IJB à base Tolérance de Faute

Introduction

Dans ce chapitre nous proposons les différentes étapes de notre nouvelle approche de sélection d'une configuration finale index qui est basée sur la logique floue (tolérance aux fautes), on inspirant des deux travaux que nous avons présentés dans le chapitre III [28,29] pour l'extraction des motifs fréquents.

IV.1 Motivation

Pour exprimer la motivation de notre approche, nous présentons l'exemple suivant [3].

Exemple 4.1 *Considérez les données de l'achat du client sur 50 produits (P1, P2,..., P50). La table 4.1 représente les comptes de clients avec les registres de l'achat correspondants pour P1 à P5 (1 pour a "acheté", 0 pour n'a pas acheté", les autres produits ne sont pas d'intérêt pour cet exemple). Supposons sue le nombre total de clients (lignes) de la base de données soyez 10,000. Pour simplicité, suppose qu'aucuns autres clients de la base de données n'ont acheté ces 5 produits. Remarquez que pour toute valeur du support minimum k > 0, l'itemset {P1..., P5} n'est pas fréquent. En fait, aucun des 5 items ne paraîtrait dans toute énumération de l'itemset fréquente pour un niveau du support de 5%.Cependant, note que 5.7% des transactions contiennent 4 des 5 produits. Si produits P1,..., P5 sont marques différentes de soda alors 5.7% des clients achètent une portion considérable des 5 marques.*

Le nombre	P1	P2	P3	P4	P5	Autres produits
100	1	1	1	1	0	…
100	0	1	1	1	1	…
80	1	1	1	0	1	…
90	1	0	1	1	1	…
200	1	1	0	1	1	…
…	…	…	…	…	…	…

Tab IV.1 – Comptes de motif des données du client-achat

*Ce motif peut être utile pour l'analyste des données mais serait non découvert par les approches de l'itemset fréquentes traditionnelles dû à la définition sévère de support. En relaxant la définition d'itemsets fréquent pour être erreur-tolérant (Faute tolérant), on pourrait identifier ce groupe de clients qui, achetez "la plupart" des produits {P1,..., P5}. Cependant, si on autorise une erreur de 0.2%, c.-à-d. on prend en compte les transactions qui manque au plus 0.2*5 item (1 ou 0 item), et si on prend un supmin=5.7%, alors on peut dire que l'itemset {P1..., P5} est fréquent avec une erreur tolérant (Faute tolérant) de 0.2%.*

De cette idée de relaxation, nous pensons que l'application l'approche de MFTF dans l'étape d'élagage des attributs candidat pour la sélection d'une configuration index sera efficace, les 02 travaux de Aouiche et Ballatreche [29,28] basent sur l'application d'un algorithme classique (Close) qui est basé sur la définition stricte du support, et comme nous avons vue dans l'exemple 4, cette définition ne permet pas de découvrir certains attributs, qui peut être dans notre cas, efficace de les choisir comme index (peut n'importe la cardinalité des tables qui contiennent ces attributs). Appliquer l'approche de MFTF, nous espérons de proposer une autre solution pour résoudre le problème de sélection d'une C.I.

IV.2) Démarche de sélection automatique d'index

Nous procédons comme suit pour proposer une configuration d'index :

1. analyse de la charge pour extraire les attributs indexables.
2. construction du contexte de recherche des motifs fréquents.
3. application de l'algorithme MFTF sur ce contexte.
4. construction de la configuration d'index candidats.
5. construction de la configuration finale d'index.

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

IV.2.1) Analyse de la charge

Les requêtes SQL présentes dans la charge sont traitées par un analyseur syntaxique automatique afin d'en extraire tous les attributs susceptibles d'être des supports d'index. Ces attributs sont ceux présents dans les clauses *Where* des requêtes. Ces attributs servent à la recherche dans les requêtes d'interrogation.

IV.2.2) Construction du contexte d'extraction

À partir des attributs extraits dans l'étape précédente, nous construisons une matrice (requêtes-attributs) qui a pour lignes les requêtes de la charge et pour colonnes les attributs à indexer [29]. Les valeurs de cette matrice est donnée par :

$$Utiliser(Qi, Aj) = \begin{cases} 1 \; si \; la \; requete \; Q_i \; utilise \; pour \; la \; selection \; l'attribut A_j \\ 0 \; sinon \end{cases}$$

Exemple 4.2 : *pour bien comprendre cette procédure, considérons la table 2.1. Cette table contient 5 requêtes qui sont exécuté sur un entrepôt de donnée modéliser par un schéma en étoile qui contient une table de fait Sales, et 02 tables de dimension Channels (notée Ch) et Custumer (C).*

1	*select S.channel id, sum(S.quantity sold) from S, Ch* *where S.channel id=Ch.channel id and Ch.channel desc='Internet'* *group by S.channel id*
2	*select S.channel id, sum(S.quantity sold), sum(S.amount sold) from S, Ch* *where S.channel id=Ch.channel id and Ch.channel desc ='Catalog'* *group by S.channel id*
3	*select S.channel id, sum(S.quantity sold),sum(S.amount sold) from S, Ch* *where S.channel id=Ch.channel id and Ch.channel desc ='Partners'* *group by S.channel id*
4	*select S.cust id, avg(quantity sold) from S, C* *where S.cust id=C.cust id and C.cust gender='M' group by S.cust id*
5	*select S.cust id, avg(quantity sold) from S, C* *where S.cust id=C.cust id and C.cust gender='F'* *group by S.cust id*

Tab. IV.2 – Requêtes de description

Afin de faciliter la construction de la matrice de contexte, nous renommons les attributs indexables comme suit: Sales.cust_id = A_1, Customers.cust_id=A_2; Customers.cust_gender = A3, Channels.channel_id=A_4; Sales.channel-id = A5; Chaînes . channel_desc = A6. La matrice est donnée ci-dessous.

	A_1	A_2	A_3	A_4	A_5	A_6
Q_1	0	0	0	1	1	1
Q_2	0	0	0	1	1	1
Q_3	0	0	0	1	1	1
Q_4	1	1	1	0	0	0
Q_5	1	1	1	0	0	0
Support	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{3}{5}$	$\frac{3}{5}$	$\frac{3}{5}$

IV.2.3) application de l'approche MFTF

Pour appliquer l'algorithme MFTF, nous avons utilise l'algorithme proposé par Ardian et al. [1] (qui nous avons détaillé dans chapitre 2). Cet algorithme est base sur 2 étapes principales :

- Utilisation de la fonction qui borne l'espace de recherche, à cause de l'explosion du nombre des itemsets.
- Utilisation de la programmation linéaire pour calculer le Ft-support d'un itemset.

IV.2.4) Construction de l'ensemble d'index candidats

Après la génération de l'ensemble des motifs fréquents, une phase de génération d'index candidats est nécessaire, nous procédons comme suit :

IV.2.4.1) Application de la fonction fitness :

Au lieu d'utiliser le support comme critère de détermination des motifs fréquents, nous inspirons de l'approche de (Bellatreche et al.) nous utilisons la même fonction fitness permettant de pénaliser chaque motif fréquent qui contient des attributs relatifs a des petites tables de dimensions, en prenant compte des cardinalités des tables de dimensions et la taille de la page système.

IV.2.4.2) Purification des motifs fréquents :

Apres l'application de la fonction fitness, une étape de purification permet d'éliminer les motifs qui ne peuvent pas générer un index de jointure. Dans le chapitre 3, nous avons présenté qu'un IJB est construit entre une table de faits et des tables de dimension en fonction des attributs non-clés. Trois scénarios de purification sont considérés:

1. le MF généré contient uniquement les attributs clés (des tables de dimension) ou de clés étrangères de la table de fait
2. le MF donnée a un certain nombre d'attributs clés plus élevé que le nombre d'attributs non-clés. Pour un attribut non-clé, nous avons besoin de deux attributs

clés pour construire un IJBs. Généralement, pour *N* attributs de la sélection, 2 × N attributs-clé sont nécessaires

3. la MF ne contient que des attributs non-clés.

Les motifs fréquents qui nous intéressent sont ceux du deuxième scenario et qui serons pris en compte dans notre démarche.

IV.2.5) la sélection de la configuration finale

L'étape de purification permet de générer un ensemble d'attributs indexables candidats. Cet ensemble est défini par l'union des attributs non clés appartenant aux motifs fréquents générés. A partir de l'ensemble d'attributs candidats, nous avons utilise l'algorithme glouton proposé dans [29], où nous avons proposé d'ajouté une fonction auxiliaire qui permet de trier les attributs selon leurs cardinalité et selon la taille de la table de dimension a laquelle ils appartiennent. L'entrée de cet algorithme comprend: (a) un schéma en étoile, (b) un ensemble de requêtes: $Q = \{Q_1, Q_2, ..., Q_m\}$, (c) PURIJB (représentant les motifs fréquents purifiée), et (d) une contrainte de stockage *S*.

Algorithme 1 : Algorithme glouton pour la sélection IJBs

Soit IJB_j un IJB définit sur l'attribut A_j. Taille (IJB_j) est le cout de stockage du IJB_j

Entrée: *Q*: un ensemble de requêtes, *PURIJB* : MF purifies, et une contrainte de stockage S.

Sortie : $Config_{finale}$: ensemble IJB sélectionnés.

Début

$Config_{finale} = IJB_{min}$;

$S: = S - taille\ (IJB_{min})$;

$PURIJB := PURIJB - A_{min}$; A_{min} : *est l'attribut utilisée pour définir* IJB_{min}.

$IJBSET = Trier\_selon\_cardinalite\_taille\_table(PURIJB)$;

Tant que ($taille\ (Config_{finale}) \leq S$) *faire*

Pour tout $A_j \in IJBSET$ faire

Si ($COST\ [Q, (Config_{finale} \cup IJB_j)]) < COST\ [Q, Config_{finale}]$)

Et (($Taille\ (Config_{finale} \cup IJB_j) \leq S$)) alors

$Config_{finale} := Config_{finale} \cup IJB_j$;

$Taille\ (Config_{finale}) := Taille\ (Config_{finale}) + Taille\ (IJB_j)$;

L'algorithme glouton (Voir algorithme 1) commence par l'index défini sur l'attribut ayant la cardinalité minimum, on ajoute ensuite le reste des index candidats, triant selon la fonction qu'on a ajouté, d'une façon itérative jusqu'à ce que l'espace de stockage autotomisé soit consommé. L'algorithme glouton est basé sur un modèle de coût mathématique, pour notre implémentation, nous avons utilisé le modèle de coût proposé par Boukhalfa [12] (III.3)

IV.5) Etudes expérimentales

Nous avons effectué des expériences pour comparer les trois Approches : DynaCharm, Dynaclose et Fault-tolerant en utilisant le modèle de coût mathématique présenté dans la section III.5. Nous avons implémenté tous les algorithmes en Java en utilisant l'IDE Netbeans sur une machine Toshiba ayant une capacité de 2 Go. Avant présenter les différentes expériences, nous présentons l'entrepôt de données utilisé dans les tests ainsi que la charge des requêtes définie sur cet entrepôt.

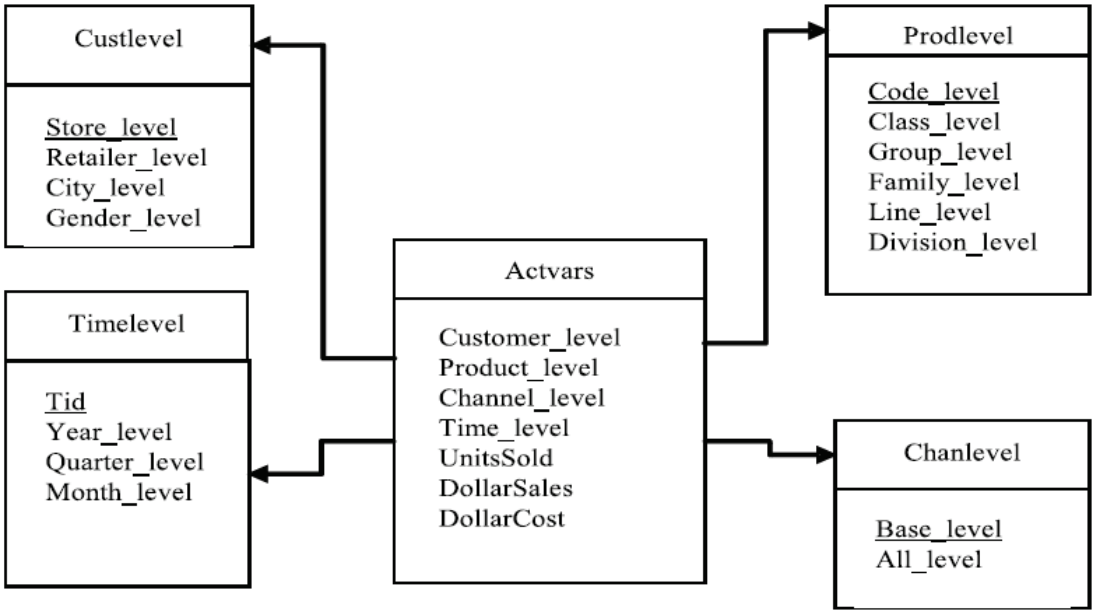


Figure IV.1 – Schéma de l'entrepôt

IV.5.1) L'entrepôt de données

L'entrepôt de données que nous avons utilises est modélisé sous forme d'un schéma en étoile (Figure IV.1). Il est constitue d'une table de fait Actavars et de 4 tables de dimensions (Chanlevel, Custlevel, Prodlevel, Timelevel). La table IV.3 résume les caractéristiques de chaque table.

Table	Nombre d'enregistrements	Taille d'un enregistrement
Actvars	24 786 000	74
Chanlevel	9	24
Custlevel	900	24
Prodlevel	9000	72
Timelevel	24	36

Tab. IV.3 – Caractéristiques des tables de l'entrepôt

IV.5.2) Charge de requêtes

Sur l'entrepôt de données décrit ci-dessus, nous avons considéré 60 requêtes de recherche composées d'un seul bloc (pas de requêtes imbriquées). Ces requêtes contiennent une jointure d'un seul bloc entre la table de Fait et une ou plusieurs tables de dimension. Chaque requête est caractérisée par sa description, sa fréquence d'accès ainsi que par l'ensemble des prédicats de sélection qu'elle utilise. L'ensemble des requêtes utilise 45 prédicats de sélection définis sur 12 attributs de sélection (ClassLevel, GroupLevel, FamilyLevel, LineLevel, DivisionLevel, YearLevel, MonthLevel, QuarterLevel, RetailerLevel, CityLevel, GenderLevel et ALLLevel) dont les cardinalités sont respectivement : 605, 300, 75, 15, 4,2, 12, 4, 99, 4, 2, 5. Chaque prédicat possède un facteur de sélectivité calculé sur l'entrepôt réel. Nous avons considéré que l'identificateur de ligne (RowId) est codé sur 10 octets et que la taille du buffer est 100.

IV.5.3) Evaluation

Nous avons effectué plusieurs expériences en utilisant le modèle de coût théorique. Ces expériences visent à comparer les performances de chaque stratégie de sélection. Nous avons implémenté trois algorithmes de sélection :

1. l'algorithme DynaClose et DynaCharm ;
2. l'algorithme de sélection basé sur l'approche MFTF ;

Dans la première expérimentation, nous avons considéré plusieurs valeurs de l'espace de stockage nécessaire pour les index sélectionnés (de 0.5 à 4 Go). La figure IV.2 montre les résultats obtenus avec un minsup=0.03.

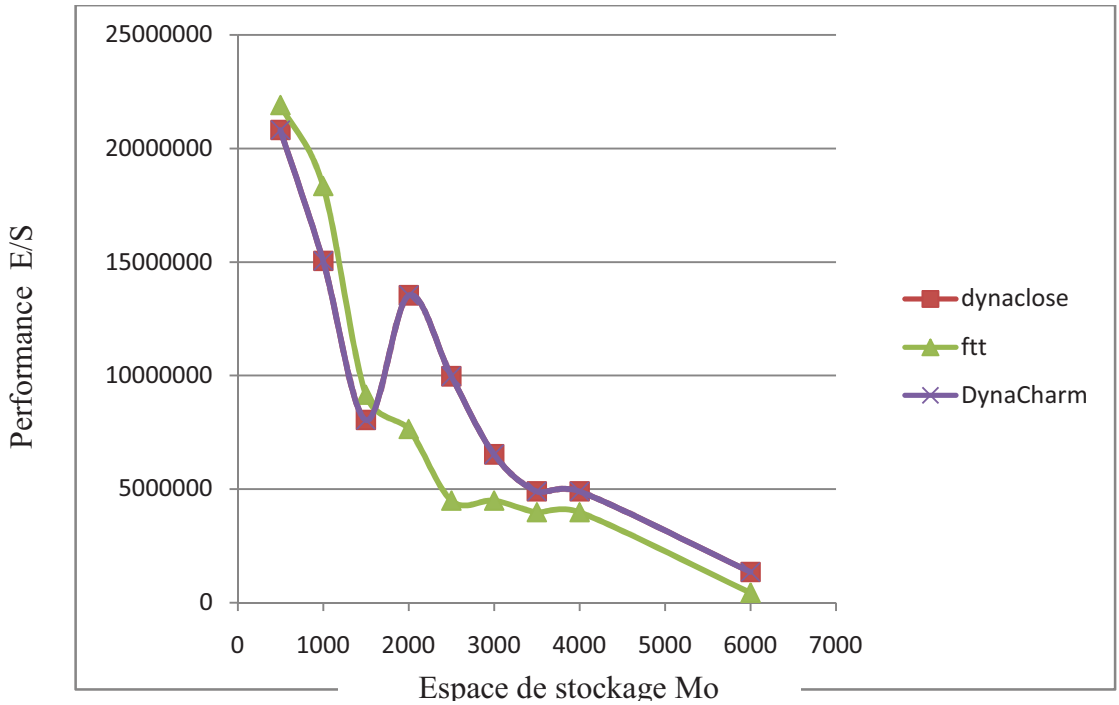


Figure IV.2 – Comparaison de performance en fonction d'espace de stockage

Avec cette valeur de minsup, les 2 Algorithme DynaClose et DynaCharm ont retourné 7 attributs candidat, par contre l'approche de MFTF a découvrit 8 attributs candidat.

Attribut retourné	Leur cardinalité	Cardinalité de leur table
GENDER_LEVEL (MFTF)	2	900
DIVISION_LEVEL	4	9000
CITY_LEVEL	4	900
LINE_LEVEL	15	9000
FAMILY_LEVEL	75	9000
GENDER_LEVEL	99	900
GROUP_LEVEL	300	9000
CLASS_LEVEL	600	9000

Tab IV.4 – les attributs candidats découvrerts avec un minsup=2

Les cardinalité des tables de dimension indexée montrent l'effet d'application de la fonction fitness.

La figure IV.2 montre bien que la performance de l'approche de MFTF est plus petite par rapport à les 2 autres approches dans l'intervalle d'espace [500-1500] Mo, cela revient au fait que l'approche de MFTF a sélectionné l'attribut GENDER_LEVEL car il a la petite cardinalité, par contre les deux autre approche n'ont pas découvrit cet attribut, alors

ils ont sélectionné l'attribut DIVISION_LEVEL pour la construction de la configuration finale (car il représente l'attribut qui a la cardinalité minimale). La grande différence entre la taille de deux tables qui contiennent ces deux attributs expliquent bien cette différence en performance. A partie de la valeur 2000 MO d'espace, l'approche de MFTF a augmenté leur performance, parce que nous avons utilisé une fonction qui trie les attributs selon leur cardinalité, cela permet de sélectionner plus d'index pour la configuration, car la taille d'un index de faible cardinalité est petit. Lorsque on atteint un espace de stockage très grand, les 3 approches donnent une configuration qui contient tout les attributs candidats, a cet effet l'approche de MFTF a retourné une bonne performance car le nombre des index candidat découvrit est plus grand que les 02 autre approches.

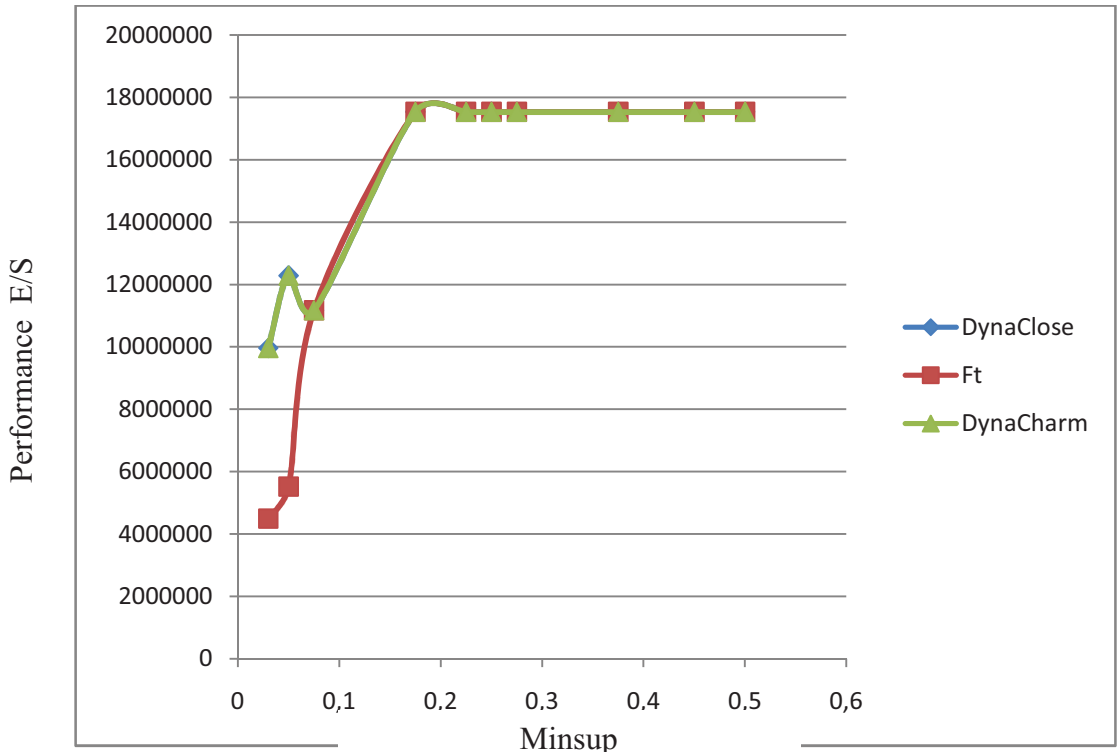


Figure IV.3 – effet de minsup sur la performance

Pour analyse l'impact de minsup sur le résultat de sélection d'une C.I, et sur l'espace occupé pour chaque configuration, nous avons effectué des expériences avec des différentes valeurs de minsup, où pour chaque valeur nous avons lancé les trois algorithmes et nous avons calculé le coût d'exécution des requêtes ainsi que la taille des index générés. Les figures IV.3 et IV.4 montrent les résultats obtenus. Il est clairement visible que lorsque le minsup est petit, beaucoup d'index sont créés et occupent donc plus d'espace de stockage. Cela est causé par le fait qu'une petite valeur de minsup implique que la majorité des motifs sont considérés fréquents. Lorsque minsup est

grand, peu d'index sont générés, et par conséquent moins d'espace de stockage est occupé. Nous proposons donc de choisir une valeur de minsup garantissant un bon compromis entre la performance et l'espace de stockage.

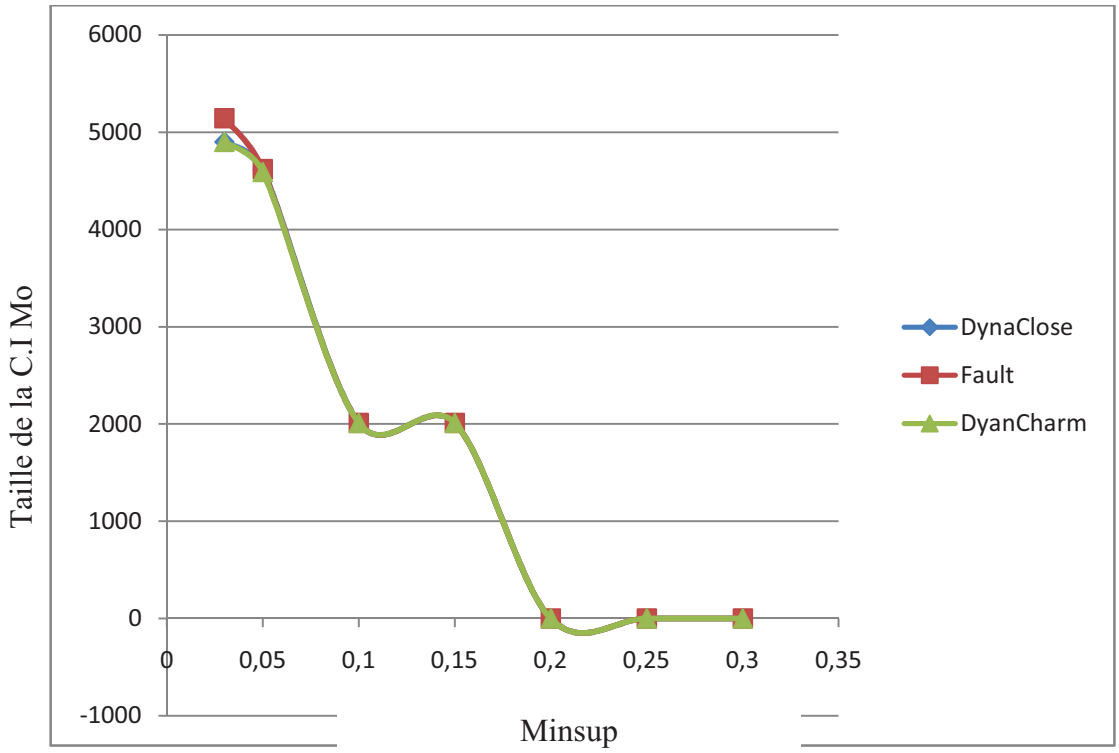


Figure IV.4 – effet de minsup sur la taille de la configuration final

Dans la dernière expérience, nous avons modifié la cardinalité des attributs indexables, où nous avons considéré que tous les attributs ont la même cardinalité et nous avons choisi plusieurs valeurs (de 200 à 2000) et pour chaque valeur nous avons exécuté les trois algorithmes. La figure IV.5 montre les résultats obtenus dans cette expérimentation. La performance se détériore considérablement avec l'augmentation des cardinalités. Cela est dû à l'augmentation de la taille des index sélectionnés qui provoque la saturation rapide de l'espace de stockage réservé et un coût de chargement énorme des index créés. Lorsque la cardinalité dépasse un certain seuil, soit l'espace de stockage disponible ne permet de sélectionner aucun index, soit les index sélectionnés sont très volumineux, ce qui rend l'utilisation de ces index non avantageuse.

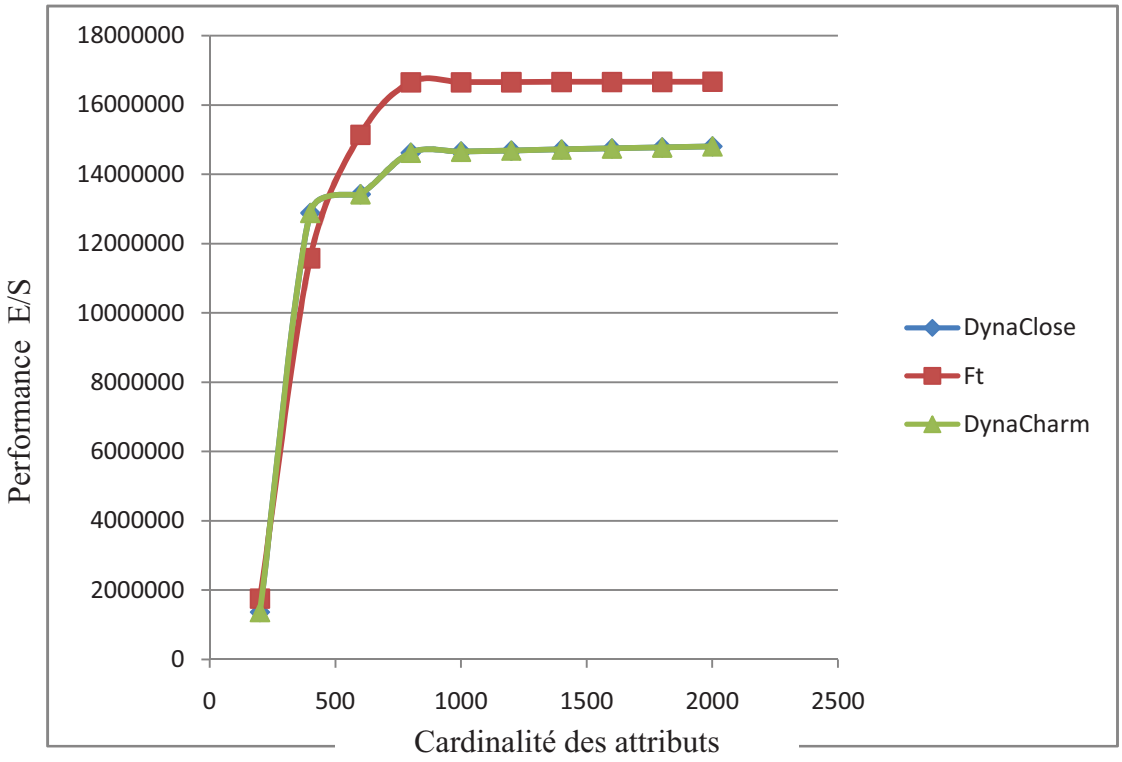


Figure IV.6 – effet de cardinalité sur la performance

IV.4) L'application de validation

Afin de réaliser le but de ce travail, nous avons programmé une application JAVA, pour effectuer les différents tests proposés dans ce mémoire.

IV.4.1) Principe de fonctionnement de l'application

Afin de réaliser les objectifs de ce mémoire, notre application est divisée en trois parties principales:

IV.4.1.1) La sélection automatique des attributs candidat : dans cette partie un analyseur lexicale des requêtes est programmé. Cet analyseur prendre en charge la charge de requête a optimisé et il retourne tout les attributs qui seront pris en considération pour la phase de sélection de la configuration finale.

IV.4.1.2) la sélection d'une configuration index : cette partie donne la main a l'utilisateur de choisit l'un des algorithmes d'optimisation présenté de ce mémoire (DynaClose, DynaCharm, MFTF), pour sélectionné une configuration finale d'index. Comme nous avons cité dans le chapitre 2, le calcule de FT-support d'un itemset est fait à base de la programmation linéaire en nombre entier, afin d'implémenté le modèle

linéaire, nous avons fait recoure à la bibliothèque java ceplex.jar[1], qui nous permet de résoudre le programme linéaire.

VI.4.1.2) validation : cette partie contient l'affichage de différente expérience effectuée. Pour cela nous avons utilisé la fameuse bibliothèque JAVA JFreeChart[2].

VI.4.2) Les diagrammes UML

Pour implémenter notre application, nous avons modélisé notre application, avec UML. Cette modélisation inclure trois diagramme : diagramme de cas utilisation, diagramme de séquence, et diagramme de classes.

VI.4.2.1) diagramme de cas utilisation

L'acteur principal de notre application est l'administrateur de l'entrepôt de donnée. L'application lui fournie les 02 actions suivantes :

1) la sélection d'une configuration index, qui inclus le chargement et l'analyse lexicale de la charge de requêtes, le chargement de schéma de l'entrepôt de donnée sur lequel les requêtes sont exécuté, et la sélection une de trois approches à utilisé.
2) Effectué une comparaison entre les 3 approches proposé pour la sélection.

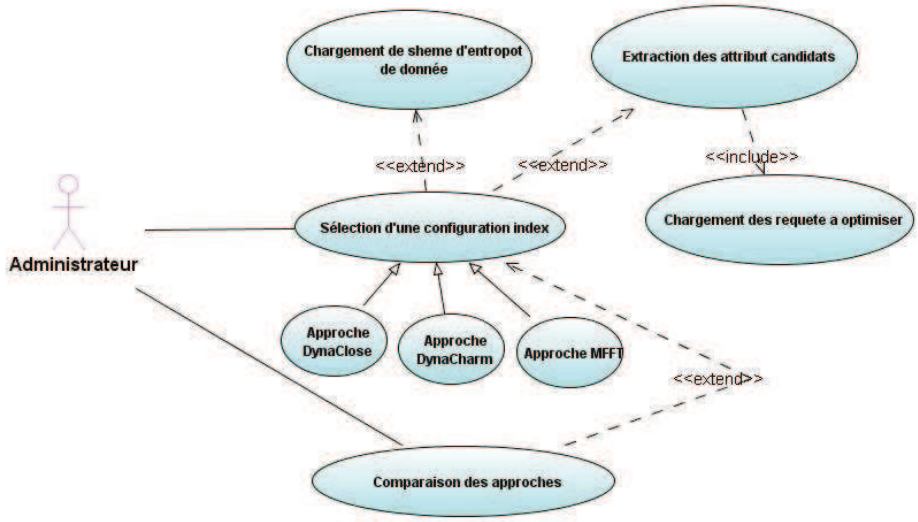


Figure IV.7 – Diagramme de cas utilisations

[1] http://www.lkn.ei.tum.de/arbeiten/faq/man/ILOG/CPLEX/cplex75/
[2] http://www.jfree.org/jfreechart/

VI.4.2.2) diagramme de séquence

L'administrateur doit charger la charge de requêtes a optimisé et charger le schéma de l'entrepôt de donnés sur lesquels ces requêtes sont exécuté. Après le chargement, il fait l'appelle à la méthode *selection* de la classe *Optimiseur*. Cette fonction reçoit comme entré, les paramètres de sélection (minsup, les 3 paramètres de relaxation), l'espace de stockage, et l'approche à utilisé. La classe *Optimiseur* a fait l'appeler à deux méthodes *selection_index()* et *selection_CIF* de les deux classes *Approche* et *Greedy_algorithme* respectivement, pour calculer les index candidat et la sélectionner de la configuration final

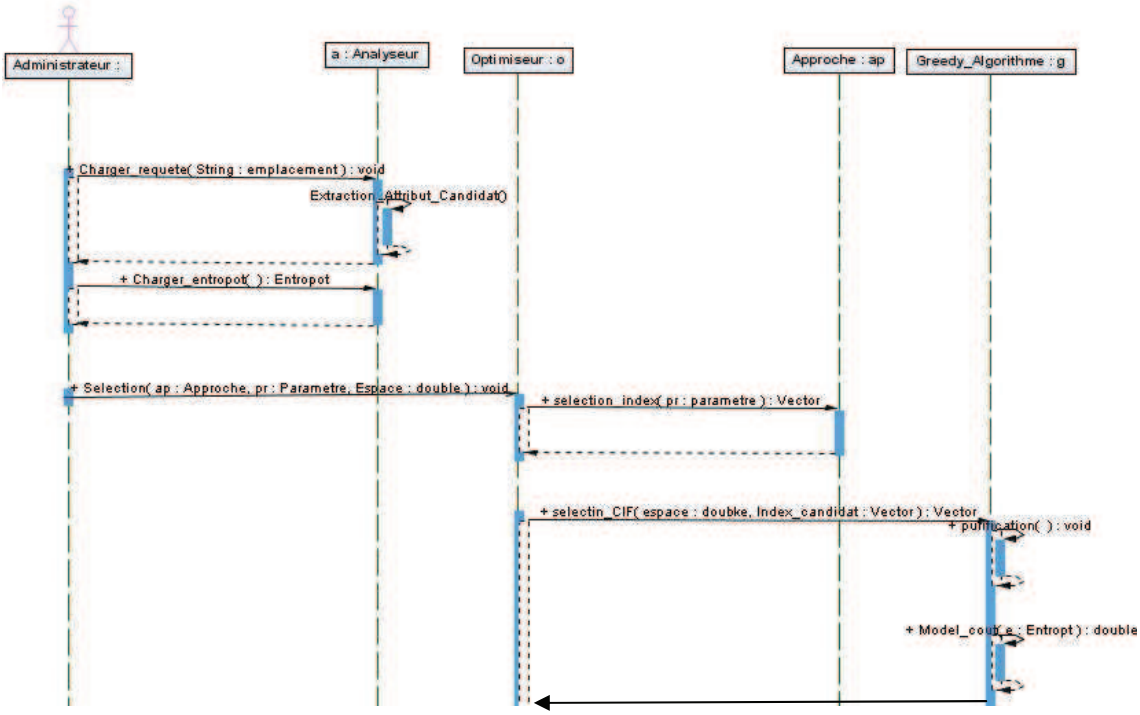


Figure IV.8 – Diagramme de séquence

VI.4.2.3) diagramme de classes

La figure IV.9 Présente le diagramme de classe associé au diagramme de séquence présenté dans la figure IV.8.

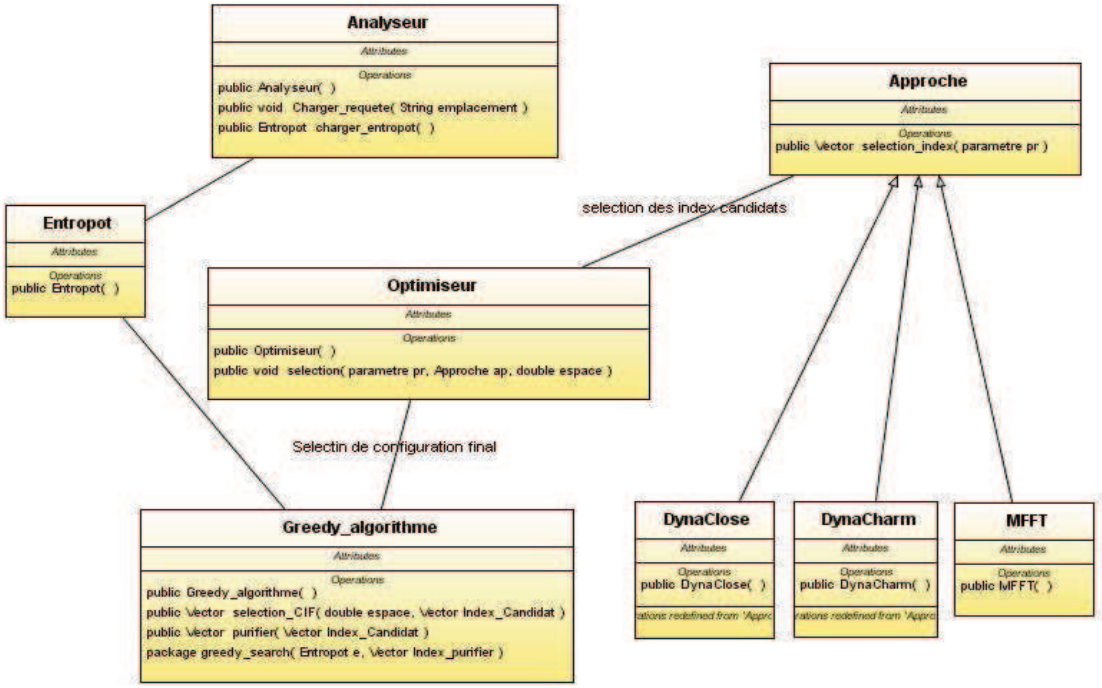


Figure IV.9 – Diagramme de classes

Conclusion

Dans ce chapitre nous avons décrit l'approche que nous avons proposée, avec la présentation des différentes étapes que nous avons considérées pour la génération de la configuration finale d'index.

Ensuit nous avons présenté nos résultats pour valider notre approche, afin d'évaluer l'efficacité de cette dernière. Une conception de l'application que nous avons implémentée a été faite dans la dernière section.

Conclusion générale

Conclusion Générale

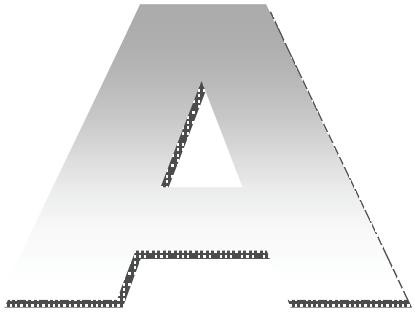
Notre travail nous a permis de découvrir le domaine des entrepôts de données et les différentes techniques d'optimisation. Nous avons décrit les index binaires de jointure, avec les problèmes liés à leurs sélections ainsi que les différents travaux proposés pour résoudre ces problèmes.

Dans ce travail, nous avons essaye d'utiliser la technique d'index binaire de jointure qui est considérée comme une technique très pertinente pour l'optimisation des requêtes en étoile (OLAP) utilisées dans le cas des entrepôts de données.

Nous avons proposé une approche de sélection d'une configuration finale d'index binaires de jointure. Notre approche permet de sélectionner une configuration initiale optimisant chaque requête séparément. La configuration finale obtenue est composée de plusieurs index qui permettent d'optimiser l'ensemble de requêtes.

Nous avons effectué plusieurs expérimentations sur notre approche et sur des approches déjà proposés telles que DynaClose, DynaCharm afin de prouver la qualité de notre approche par rapport à ces dernières.

A la fin nous avons déduit que l'approche que nous avons proposé a prouvé son efficacité quand à l'optimisation des performances des entrepôts de données. L'idée de proposer une nouvelle approche pour sélectionner une combinaison d'index de jointure binaire est très originale et prometteuse. Les résultats présentés dans ce mémoire montrent l'efficacité de notre approche tout en contribuant à un axe de recherche très prometteur et d'actualité.



La formulation avec la programmation linéaire en nombres entiers

1. " Tu peut lancer le satellite Z_1 seulement si tu as choisi un propulseur compatible Z_2."

$$Z_1 \leq Z_2$$

2. " Z_3 peut être produit si et seulement si une machine Z_1 et un ouvrier Z_2 sont disponible."

$$Z_3 \leq Z_1$$

$$Z_3 \leq Z_2$$

$$Z_3 + 1 \leq Z_1 + Z_2$$

3. " Le projet Z_3 peut être installé si seulement si le projet Z_1 ou le projet Z_2, ou bien les deux projets sont installés."

$$Z_3 \leq Z_1 + Z_2$$

$$Z_3 \geq Z_1$$

$$Z_3 \geq Z_2$$

4. " La ligne d'emballage Z_3 peut recevoir un produit de la ligne de traitement Z_1 ou de la ligne de traitement Z_2."

$$Z_3 \leq Z_1 + Z_2$$

$$Z_3 \geq Z_1 - Z_2$$

$$Z_3 \geq -Z_1 + Z_2$$

$$Z_3 \leq 2 - Z_1 + Z_2$$

5. "Non Z."

$$1 - Z_1$$

6. " A partir de l'ensemble $\{Z_1, Z_2, \ldots, Z_k\}$, choisir au maximum un élément."

$$\sum_{i=1}^{k} Z_k \leq 1$$

7. " A partir de l'ensemble $\{Z_1, Z_2, , Z_k\}$, choisir exactement un élément."

$$\sum_{i=1}^{k} Z_k = 1$$

8. " A partir de l'ensemble $\{Z_1, Z_2, \ldots, Z_k\}$, choisir au minimum un élément."

$$\sum_{i=1}^{k} Z_k \geq 1$$

9. " A partir de l'ensemble $\{Z_1, Z_2, \ldots, Z_k\}$, choisir au minimum quatre éléments."

$$\sum_{i=1}^{k} Z_k \leq 4$$

10. " Si tu construiras un entrepôt Z =1, tu peux stocker dans cet entrepôt jusqu'à 13 tonnes de X."

$$X \leq 13Z$$

11. " Si tu construiras un entrepôt Z =1, tu peux stocker dans cet entrepôt au moins 56 tonnes de X mais pas plus de 141 tonnes."

$$X \geq 56Z \; X \leq 141Z$$

Bibliographie

[1] A. K. Poernomo and V. Gopalkrishnan. Towards efficient mining of proportional fault-tolerant frequent itemsets. *KDD '09* Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, 2009.

[2] A. K. Poernomo and V. Gopalkrishnan. Mining statistical information of frequent fault-tolerant patterns in transactional databases. In *ICDM*, pages 272–281, 2007.

[3] C. Yang, U. M. Fayyad, and P. S. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *KDD*, pages 194–203, 2001.

[4] J. Liu, S. Paulsen, X. Sun, W. Wang, A. B. Nobel, and J. Prins. Mining approximate frequent itemsets in the presence of noise: Algorithm and analysis. In *SDM*, 2006.

[5] A. K. Poernomo and V. Gopalkrishnan. Efficient computation of partial-support for mining interesting itemsets. In *SDM*, 2009.

[6] T. Calders and B. Goethals. Quick inclusion-exclusion.In *KDID*, pages 86–103, 2005.

[7] Bay S. D., « The UCD KDD Archive [http://kdd.ics.uci.edui]», 1999, Irvine, CA : University of California, Departement of information and computer Science.

[8] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme and L. Lakhal. Pascal : un algorithme d'extraction des motifs fréquents. 2001.

[9] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. VLDB '94 Proceedings of the 20th International Conference on Very Large Data Bases, 1994.

[10] N. Pasquier, Y. Bastide, R. Taouil and L. Lakhal. Discovering Frequent Closed Itemsets for Association Rules. ICDT, pages 398-416, 1999.

[11] J. Han, J. Pei and Y. Yin Mining frequent patterns without candidate generation. ACM SIGMOD Record Homepage Volume 29 Issue 2, June 2000 ACM New York, NY, USA.

[12] K. Boukhalfa. De la conception physique aux outils d'administration et de tuning des entrepôts de données. Ph.d. thesis Ecole nationale supérieure de mécanique et d'aérotechnique Poitier juillet 2009.

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

[14] L. Toumi. Auto-administration dans les VLDB : Problèmes d'optimisation des structures. Magister thesis, Université Lumière Lyon 2, December 2005.

[15] W. H. Inmon. Building the Data Warehouse. Third Edition, Wiley Computer Publishing 2002.

[16] R. Kimball. The Data Warehouse Toolkit. John Wiley and Sons, 1996.

[17] O. Test. Modélisation et manipulation d'entrepôt de données complexes et historisées. Thèse de doctorat en informatique, Université Paul Sabatier, 2000.

[18] H. Gupta, V. Harinarayan, A. Rajaraman and J. Ullman. Index selection for OLAP. Proceedings of international Conference on Data Engineering (ICDE), pages 208-219, April 1997.

[19] I. Rahem, K. Boukhalfa, A. Mouhous et Z. Alimazighi. Sélection des index de jointure Binaires pour les entrepôts de données : Adaptation des algorithmes de fragmentation Vertical et Outil d'assistances. USTHB – Alger, 2009.

[20] H. Gupta. Selection of views to materialize in a data warehouse. Proceedings of the 6th International Conference on Database Theory (ICDT '97), pages 98–112, 1997.

[21] L. Bellatreche, K. Boukhalfa, and H. I. Abdalla. Saga : A combination of genetic and simulated annealing algorithms for physical data warehouse design. in 23rd British National Conference on Databases, (212-219), July 2006.

[22] S. Navathe and M. Ra. Vertical partitioning for database design : a graphical algorithm. ACM SIGMOD, pages 440–450, 1989.

[23] 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.

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

[25] R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite. The Data Warehouse Lifecycle Toolkit : Expert Methods for Designing, Developing, and Deploying Data Warehouses. John Wiley and Sons, 1998.

[26] M. Golfarelli, D. Maio, and S. Rizzi. Conceptual design of data warehouses from e/r schemes. in the 31th Hawaii Conference on System Sciences, January 1998.

[27] P. O'neil. Multi-table joins through bitmapped join indioces. SIGMOD, 24(03), 1995.

[28] K. Aouiche, J. Darmont, O. Boussaid, and F. Bentayeb. Automatic Selection of Bitmap Join Indexes in Data Warehouses. 7th International Conference on Data Warehousing and Knowledge Discovery (DAWAK 05), August 2005.

[29] L. Bellatreche, R. Missaoui, H. Necir, and H. Drias. A data mining approach for selecting bitmap join indices. Journal of Computing Science and Engineering, 2(1) :206–223, 2008.

[30] A. Salleb. Recherche de motifs fréquents pour l'extraction de règle d'association et de caractérisation. Thesis l'Université d'Orléans Discipline Informatique.

[31] S. Choenni, H. Blanken, and T. Chang. Index selection in relational databases. In 5th International Conference on Computing and Information (ICCI 93), Ontario, Canada, pages 491–496, 1993.

[32] J. Kratica, D. Ljubig, and D. T o˘sig. A Genetic Algorithm for the Index Selection Problem. Lecture Notes in Computer Science, 2611 :281–291, 2005.

[33] S. Chaudhuri and V. Narasayya. An efficient cost-driven index selection tool for microsoft sql server. Proceedings of the International Conference on Very Large Databases, pages 146–155, August 1997.

[34] M. Frank, E. Omiecinski, and S. Navathe. Adaptive and automated index selection in rdbms. In 3rd International Conference on Extending Database Technology (EDBT 92), Vienna, Austria, pages 277–292, 1992.

[35] Han J., KamberM., *Data Mining : Concepts and Techniques*, Morgan Kaufmann, septembre 2000.

Résumé: la conception physique des entrepôts de données relationnels est basée essentiellement sur la sélection d'un ensemble d'index afin de réduire le cout d'exécution des requêtes OLAP complexe qui contient des jointures très couteux. Les index de jointure binaire (IJB) sont très adaptés pour réduire le cout d'exécution de ces jointures. Ils sont définis sur la table de fait en utilisant un ou plusieurs attributs de tables de dimension. Sélectionner une configuration d'IJB pour réduire le cout d'exécution d'un ensemble de requête est reconnu comme un problème NP-Complet. Peu d'algorithmes ont été proposés pour résoudre ce problème. Dans ce cadre de travail, nous avons proposé une approche de sélection d'une C.I.J.B qui est basé sue une technique de data mining. Notre approche permet de sélectionner une C.I en appliquant l'approche de MFTF[1] . La configuration finale obtenue est composée de plusieurs index qui permettent d'optimise le cout d'exécution de l'ensemble de requête. Nous avons effectué plusieurs expérimentations sur notre approche et sur des approches déjà proposés tel que DynaClose, DynaCharm afin de prouber la qualité de notre algorithme par rapport à ces dernières.

Mots Clés : entrepôts de données-Index de jointure binaire-tolérance de faute- Data mining-requêtes OLAP.

Abstract: the physical design of relational data warehouse is essentially based on the selection of a set of indexes to reduce the coast of complex OLAP queries that contained costly join. The binary join indexes are very suitable to reduce the cost of implementation of these joints. They are defined on the fact table using one or more attributes from dimension tables. Select a configuration BJI to reduce the cost of execution a set of queries is known as a NP-complete problem. Few algorithms have been proposed to solve this problem. In this work, we proposed an approach for selecting a C.I.B.J.I that is based on a technique of data mining. Our approach, to select a C.I, uses the approach of Fault-Tolerant Frequent Itemsets. The final configuration obtained is composed of several indexes that optimize the cost of execution of all queries. We conducted several experiments on our approaches already proposed such as DynaClose, DynaCharm to prove the quality of our algorithms with respect to latter

Keywords: data warehouse-Index joins binary-fault tolerant-data mining-OLAP queries

ملخص: إن تصميم مستودعات المعلومات الارتباطية يستند أساسا على اختيار مجموعة من المؤشرات للحد من تكلفة طلابيات المعقدة OLAP. إن مؤشرات الربط التنائي مناسبة جدا للحد من هذه التكلفة. إنها معرفة على جدول الأعمال باستعمال مسند أو عدة مستندات من جداول الأبعاد. إن اختيار تشكيلة نهائية لمؤشر ربط تنائي لكي تقلل من تكلفة مجموعة من الطلابيات يعرف كمشكلة صعب جدا. عدد قليل من الخوارزميات اقترحت للحد من هذه المشكلة. في هذا العمل اقترحنا نهج جديد لاختيار تكوين مؤشر تنائي يعتمد على تقنيات DATA MINING. منهجنا يعتمد أساسا على استخراج MFTF. لقد أجرينا عدد من التجارب لمقارنة منهجنا مع المنهجين DynaClose و DynaCharm لكي نثبت فعالية المنهج المقترح.

الكلمات الرئيسية : مستودعات المعلومات, مؤشر الربط التنائي, خطأ متسامح, استخراج المعلومات, الطلابيات OLAP.

[1] Motif fréquent au tolérance de faute