DynaClose : Une approche de data mining pour la sélection des index de jointure binaires dans les entrepôts de données

Hamid Necir*, Ladjel Bellatreche**, Rokia Missaoui***

* Université de Bab Ezzouar BP 32 El Alia Bab Ezzouar ALGERIE
ncrhmd@yahoo.fr
** Université de Poitiers - LISI/ENSMA – FRANCE
bellatreche@ensma.fr
*** Université du Québec en Outaouais (UQO) – CANADA
rokia.missaoui@uqo.ca[1]

Résumé. L'indexation est l'une des techniques d'optimisation redondantes qui accélère les requêtes OLAP. Deux types d'index sont disponibles : les mono-index (B-tree, index binaire, projection, etc.) et les multi-index (index de jointure). Pour un entrepôt représenté par un schéma en étoile, les index de jointure binaires sont souvent utilisés pour accélérer les requêtes de jointure en étoile connues pour leur nombre important d'opérations de jointure. La sélection des index de jointure binaires est un problème difficile vu le nombre important des attributs candidats participant à la construction des index. Pour surmonter cette difficulté, nous proposons la démarche suivante : (1) nous adaptons d'abord un algorithme de fouille de données, appelé, *Close* qui permet de générer un ensemble d'itemsets fermés fréquents qui représentent les attributs candidats pour le processus de sélection des index. (2) Une fois les attributs candidats générés, nous proposons un algorithme itératif qui sélectionne un ensemble d'index de jointure binaires en prenant en compte l'ensemble des attributs candidats. Ces index doivent minimiser le coût d'exécution d'un ensemble de requêtes fréquentes et respecter une contrainte de stockage. Finalement, notre approche est validée par une étude expérimentale en la comparant avec les solutions existantes.

1 Introduction

Les entrepôts de données et les bases de données de grande taille sont souvent accédés par des requêtes complexes et coûteuses en terme de temps de calcul par le fait qu'elles nécessitent des opérations de jointure. Les entrepôts de données sont souvent représentés par un schéma en étoile constitué par une table des faits et de tables de dimension. Les requêtes typiques définies sur ce schéma sont appelées les *requêtes de jointure en étoile* (star join queries) qui ont les caractéristiques suivantes : (1) elles possèdent des jointures multiples entre la table des faits ayant une taille importante et les tables de dimension, (2) elles n'ont aucune jointure entre les tables de dimension (toute opération de jointure passe par la table

[1] Ce travail a été réalisé lors d'un congé sabbatique à l'Université Blaise Pascal à Clermont-Ferrand en France.

centrale qui est la table des faits) et (3) chaque table de dimension impliquée dans une opération de jointure possède plusieurs prédicats de sélection sur ses *attributs descriptifs*.

Plusieurs structures ont été proposées pour optimiser les opérations de jointure. Ces dernières peuvent être classées en deux catégories : (1) les structures d'optimisation non redondantes et (2) les structures d'optimisation redondantes. Dans la première catégorie, nous pouvons citer les différentes implémentations de l'opération de jointure : *les boucles imbriquées, le tri fusion et le hachage*. Cependant, ces techniques ne sont efficaces que lorsque la jointure concerne deux tables. Les structures redondantes sont plus efficaces pour accélérer les opérations de jointure sur plusieurs tables (ONeil et al., 1997). Comme leur nom l'indique, ces structures nécessitent un espace de stockage et un coût de maintenance. On peut ainsi citer les index de jointure (Valduriez et al., 1984), les vues matérialisées (Gupta, 1999), (Rizzi et al., 2003), etc.

Dans cet article, nous nous intéressons à une structure redondante, à savoir les index de jointure. Un index de jointure peut être vu comme une jointure pré-calculée (Valduriez, 1984). Créé à l'avance, il est implémenté par une relation d'arité 2. Ce genre d'index est souhaité pour les requêtes OLTP, car elles possèdent souvent des jointures entre deux tables (Red Brick Systems, 1997). Dans le contexte des entrepôts de données, Red Brick (Red Brick Systems, 1997) a proposé un nouvel index de jointure, appelé, *index de jointure en étoile* (star join index), adapté aux requêtes de jointure en étoile. Il peut contenir toute combinaison de clés étrangères de la table des faits. L'index de jointure binaire est une variante de l'index de jointure en étoile. Etant donné que les requêtes de jointure en étoile possèdent des opérations de jointure suivies par des opérations de sélection, un bitmap représentant les n-uplets de la table de faits est créé pour chaque valeur distincte de l'attribut de la table dimension sur lequel l'index est construit. Le $i^{ème}$ bit du bitmap est égal à 1 si le n-uplet correspondant à la valeur de l'attribut indexé peut être joint avec le n-uplet de rang i de la table de faits. Dans le cas contraire, le $i^{ème}$ bit est à zéro. Les index de jointure binaires sont efficaces pour les requêtes de type COUNT, AND, OR, NOT, d'où leur implémentation dans les SGBDs commerciaux, comme Oracle, SQL server, et DB2. Notons que les index binaires sont recommandés pour des attributs de faible cardinalité comme *sexe*. Notons que dans un contexte décisionnel, les requêtes d'analyse se font généralement sur des indicateurs (attributs) dont le domaine est restreint. Une autre caractéristique offerte par les index de jointure binaires est la compression (Johnson, 1999), d'où une réduction de leur espace de stockage et la possibilité de les stocker en mémoire centrale.

La sélection des index de jointure binaires est un problème difficile (Chaudhuri et al. 2004). Cette difficulté est due aux différents choix d'attributs des tables de dimension qui peuvent participer dans le processus de leur construction. Pour réduire le nombre des attributs potentiels dans la construction des index, certains travaux ont proposé d'utiliser les techniques de fouille de données (Agrawal et al. 1994), (Netz et al., 2000) dans le but de générer les itemsets (motifs) fréquents qui constitueront les attributs candidats dans le processus d'indexation. La principale limitation de ces travaux est qu'ils considèrent *seulement* la fréquence d'accès des requêtes dans le processus de génération des motifs fréquents. Ce choix peut être discutable, car nous avons fait le lien entre ce problème et le problème de sélection d'un schéma de fragmentation verticale[2]. Les travaux sur la fragmentation verticale ont montré la limite des algorithmes de fragmentation basés sur la

[2]La fragmentation verticale d'une table consiste à décomposer les attributs de cette table en plusieurs fragments.

fréquence (pour plus de détails voir le travail de (Fung et al., 2003)). En se basant sur ce constat, nous proposons un algorithme de sélection de motifs fréquents qui prend en considération des paramètres autres que la fréquence d'accès, comme la taille des tables (la tables des faits et les tables de dimension) concernées par l'indexation, car le coût d'une opération de jointure dépend fortement de la taille des tables jointes (Getoor et al., 2001).

L'article est organisé en six sections : La section 2 présente une formalisation du problème de sélection d'index de jointure et explore les approches d'indexation existantes en montrant leurs principes et leurs insuffisances sur un exemple. La section 3 présente notre approche de sélection des motifs fréquents fermés, appelée, *DynaClose* en adaptant l'algorithme Close. La section 4 présente notre algorithme de sélection des index de jointure binaires sous une contrainte d'espace de stockage. La section 5 présente la partie expérimentale qui valide notre approche de sélection des index de jointure. Enfin la section 6 conclut l'article en récapitulant les résultats principaux et en suggérant des travaux futurs.

2 Index de jointure binaire : concepts et approche de sélection existantes

Actuellement, la plupart des SGBDs commerciaux supportent l'utilisation des index de jointure binaires. Ils peuvent être définis sur un seul ou plusieurs attributs *non clés* de tables de dimension, comme le montre l'exemple suivant :

CREATE BITMAP INDEX sales_c_gender_bjix
ON sales(customers.cust_gender)
FROM sales, customers
WHERE sales.cust_id = customers.cust_id

qui crée un index de jointure binaire entre la table des faits *sales* et la table de dimension *customers* sur l'attribut de sélection *gender*. Le résultat de cet index est une table avec trois colonnes, représentant respectivement, le *numéro de ligne de la table des faits* (*Row Identifier*) participant dans la jointure, une *colonne binaire* pour la valeur de sexe féminin et une *colonne binaire* pour la valeur de sexe masculin.

Supposons que nous ayons la requête suivante :

SELECT count(*) FROM sales, customers
WHERE gender='F' and *sales.cust_id = customers.cust_id.*

L'index de jointure *sales_c_gender_bjix* peut être utilisé pour accélérer cette requête, en utilisant le "hint" suivant[3] :

SELECT **/*+ INDEX(sales_c_gender_bjix*)*/** count(*) FROM sales, customers
WHERE gender='F' and *sales.cust_id = customers.cust_id.*

Dans ce cas, la requête utilise uniquement cet index et sans aucun accès à la table des faits, ce qui représente un gain considérable (on économise une opération de jointure).

Notons que sur chaque attribut de sélection non clé défini dans les requêtes, un index de jointure pourrait être construit. Etant donné que les requêtes de jointure en étoile possèdent un nombre important d'opérations de sélection (voir Section 1) et qu'un index de jointure

[3]Un "hint" est une directive qui force l'optimiseur de requêtes à choisir un plan d'exécution.

binaire peut être construit sur un ou plusieurs attributs de sélection, le nombre d'index de jointure binaires potentiels devient très large.

Formellement, le problème de sélection des index de jointure binaires est formulé comme un *problème d'optimisation* sous la forme suivante : étant donné :

(1) un entrepôt de données modélisé par un schéma en étoile formé d'une table des faits F et de $D = \{D_1,...,D_d\}$ tables de dimensions,

(2) un ensemble de requêtes d'interrogation $Q = \{q_1,...,q_m\}$ (les plus fréquentes) avec leurs fréquences d'accès $f = \{f_1,...,f_m\}$,

(3) une contrainte de stockage S.

L'objectif est de sélectionner un ensemble d'index réduisant le coût d'exécution de requêtes et respectant la contrainte de l'espace de stockage S.

Plusieurs approches ont été proposées pour traiter ce problème, nous les détaillons dans la section suivante.

2.1 Approches existantes de sélection d'index de jointure binaires

Plusieurs travaux de recherche ont montré l'utilité de l'indexation et proposent différentes approches pour la sélection d'index dans le contexte des bases de données ou des entrepôts de données. Ces travaux ont traité le problème de sélection d'index comme le problème de sac à dos, où plusieurs types d'algorithmes ont été proposés (des algorithmes itératifs, la programmation linéaire, etc.) (Chaudhuri, et al., 2004). Ces algorithmes sont basés sur des *modèles de coût* calculant la performance des index sélectionnés. Deux types de modèles sont disponibles : (1) des modèles de coût de l'optimiseur de requêtes de SGBD utilisé, comme dans les travaux du groupe de base de données de Microsoft (Chaudhuri et al, 1997) et les travaux du groupe de bases de données d'IBM (DB2) (Rao et al. 2002) et (2) des modèles de coût mathématiques utilisés pour valider les algorithmes proposés (Choenni et al, 1993), (Gundem, 1999), (Feldman et al., 2003).

D'autres types de travaux ont proposé d'utiliser des techniques de fouille de données afin de réduire la complexité des algorithmes de sélection des index de jointure (Aouiche, 2005). L'idée de base de ces algorithmes est que plus un attribut ou un groupe d'attributs est fréquemment présent dans la charge de requêtes plus il est intéressant de le(s) considérer dans le processus de sélection des index. Ensuite, des algorithmes de sélection sont exécutés pour générer les index finaux. Pour mieux comprendre ces travaux, quelques définitions s'imposent.

Définition 1. Un motif (itemset) fréquent : soient I = i, …, *m* un ensemble de *m* items et B = t_1, …, t_n une base de données (contexte) de *n* transactions. Chaque transaction est composée d'un sous ensemble d'items. Un sous ensemble I' de taille k est appelé un *k*-itemset.

Définition 2. Le support d'un motif I', noté sup(I'), est la proportion de transactions de B qui contiennent I'.

Définition 3. Motif fréquent. Soit un seuil $minsup \in [0,1]$, appelé le support minimum. Un motif est dit *fréquent* si : $\sup(P) \geq minsup$.

Définition 3. Motif fréquent fermé. Un motif fermé est un ensemble maximal d'items communs à un ensemble d'objets. Un motif i $\subseteq$ I tel que le support(i) $\geq$ minsup est appelé motif fréquent fermé.

Dans le contexte de sélection des index de jointure binaires, les transactions sont des requêtes fréquentes et les items sont les attributs utilisés dans les requêtes.

Trois grandes approches ont été proposées pour l'extraction des motifs fréquents (Bastide et al. 2002) : La première consiste à parcourir itérativement par niveaux l'ensemble des motifs. Durant chaque itération un ensemble de motifs candidats est créé en joignant les motifs fréquents découverts durant l'itération précédente. Les supports de ces motifs sont calculés et les motifs non fréquents sont supprimés. L'algorithme de référence basé sur cette approche est l'algorithme Apriori (Agrawal et al., 1994). La seconde approche est basée sur l'extraction des motifs fréquents maximaux dont tous les sur-ensembles sont non fréquents et tous les sous-ensembles sont fréquents. La troisième approche, représentée par l'algorithme *Close* (Parsquier, 1999), est basée sur le treillis de Galois. Les motifs fermés fréquents (et leurs supports) sont extraits de la base de données en réalisant un parcours par niveaux. Notons que Close a été utilisé dans (Aouiche, 2005) dans la sélection des index de jointure. Notre travail rentre dans cette catégorie de travaux de sélection d'index.

2.2 Limites de l'approche itemsets fermés fréquents pour la sélection d'index

Pour montrer les limites des approches existantes, considérons l'exemple suivant. Soit un ensemble de cinq requêtes définies sur un schéma en étoile composé de deux tables de dimension *channels* et *customers* et une table des faits *sales* (figure 1). La taille des tables (en termes d'instances) est :

|*sales*| = 46 521, | *channels*| = 5 et | *customers*| = 30 000.

(1) Select sales.channel_id, sum(sales.quantity_sold) from sales, channels where sales.channel_id=channels.channel_id and *channels.channel_desc*='Internet' group by sales.channel_id;
(2) select sales.channel_id, sum(sales.quantity_sold), sum(sales.amount_sold) from sales, channels where sales.channel_id=channels.channel_id and *channels.channel_desc* ='Catalog' group by sales.channel_id;
(3) select sales.channel_id, sum(sales.quantity_sold),sum(sales.amount_sold) from sales, channels where sales.channel_id=channels.channel_id and *channels.channel_desc* ='Partners' group by sales.channel_id;
(4) Select sales.cust_id, avg(quantity_sold) from sales, customers where sales.cust_id=customers.cust_id and **customers.cust_gender**='M' group by sales.cust_id;
(5) Select sales.cust_id, avg(quantity_sold) from sales, customers where sales.cust_id=customers.cust_id and **customers.cust_gender**='F' group by sales.cust_id;

FIGURE 1 : Exemple de charge de requêtes

Supposons que minsupp=3 (en valeur absolue). Un algorithme de sélection des index candidats en utilisant une approche basée sur l'extraction des itemsets fermés fréquents,

comme Close utilisée dans (Aouiche, 2005), retourne un index de jointure binaire construit sur la table *Channels* et *Sales* sur l'attribut *channels.channel_desc* (figurant 3 fois dans les prédicats de sélection des requêtes). L'index de jointure entre la table de dimension *customers* et la table des faits *sales* (sur l'attribut *gender*) est élagué car il n'est pas fréquent (il ne figure que deux fois). Mais ce dernier pourrait réduire le coût d'exécution de requête car il est défini sur une table de dimension plus importante que la table *channels* (30 000 vs. 5). Cet exemple montre l'insuffisance des travaux utilisant les approches d'extraction des itemsets fréquents ayant comme métrique d'élagage la fréquence d'apparition des attributs dans les requêtes.

Nous affirmons que pour une bonne sélection d'index de jointure, d'autres paramètres comme la taille des tables doivent être pris en considération.

3 Notre approche de sélection des motifs fréquents

L'exemple de la section précédente a démontré les insuffisances des approches d'extraction des itemsets fréquents dans le contexte de sélection des index de jointure binaires. En conséquence, il faut les adapter afin de prendre en compte les paramètres comme la taille des tables. Pour ce faire, nous proposons un nouveau paramètre, appelé, *fitness* qui sera utilisé comme une métrique d'élagage. Il est défini :

$$fitness = \frac{\sum_{i=1}^{n} \sup_i \times \alpha_i}{n}$$

où n représente le nombre d'attributs non clés dans chaque itemset fermé fréquent extrait et $\alpha_i = \frac{|D_i|}{|F|}$. Notons que la valeur de $fitness$ est calculée à chaque itération. Cette valeur (fitness) pénalise les itemsets fréquents qui sont définis sur des tables de dimension de petite taille.

Notre approche utilise une charge formée par un ensemble de requêtes. Cet ensemble nous permet d'établir un contexte d'extraction auquel un algorithme d'extraction des itemsets fermés fréquents est appliqué afin d'avoir un ensemble d'index candidats. Par la suite, nous appliquons un algorithme de sélection des index de jointure binaires. Cet algorithme est dirigé par un modèle de coût représentant le nombre d'entrées sorties nécessaires pour exécuter l'ensemble de requêtes. Notre démarche se déroule en 5 étapes : (1) le prétraitement de requêtes, (2) la construction du contexte d'extraction, (3) l'extraction de la liste des itemsets fermés fréquents, (4) l'établissement de la liste des index candidats, et (5) la génération des index. Ces étapes sont détaillées dans les sections suivantes.

3.1 Prétraitement des requêtes

Les requêtes présentes dans la charge sont traitées afin d'en extraire tous les attributs susceptibles d'être candidats pour l'indexation. Ces derniers sont ceux présents dans les clauses WHERE des requêtes.

3.2 Construction du contexte d'extraction

La matrice d'usage des attributs constitue notre contexte d'extraction. Les lignes et les colonnes de cette matrice représentent les différentes requêtes et les attributs obtenus par l'étape de prétraitement des requêtes, respectivement. A chaque requête Q_i et chaque attribut A_j, on associe une valeur d'usage de l'attribut qui est égale à 1 si la requête utilise l'attribut A_j, 0 sinon.

Exemple 1

Considérons l'exemple de charge de requête de la Figure 1. Soient les abréviations suivantes : sales.cust_id=A1 ; customers.cust_id=A2; customers.cust_gender =A3 ; channel_id=A4; sales.channel_id=A5; chanels.channel_desc=A6. Nous obtenons la matrice d'usage des attributs suivante :

	A1	**A2**	**A3**	**A4**	**A5**	**A6**
Q1	0	0	0	1	1	1
Q2	0	0	0	1	1	1
Q3	0	0	0	1	1	1
Q4	1	1	1	0	0	0
Q5	1	1	1	0	0	0
SUPPORT	2/5	2/5	2/5	3/5	3/5	3/5

FIGURE 2 : Contexte d'extraction

3.3 DynaClose : extraction des itemsets fermés fréquents

Dans (Aouiche, 2005), une procédure de sélection d'index de jointure binaires est proposée en se basant surr l'algorithme Close (Pasquier et al., 1999). Nous utilisons une démarche assez similaire, à la différence que la nôtre utilise une métrique d'élagage basée à la fois sur les supports des motifs et sur la taille des tables. Rappelons que les approches existantes élaguent selon la fréquence des items par rapport à minsup. DynaClose élague le même nombre d'itemsets non fréquents élagués par l'approche classique (Aouiche, 2005), mais en se basant sur la fonction de *fitness* au niveau des fréquents fermés (qui formeront les index candidats), contrairement à l'approche classique où l'élagage se fait au niveau des générateurs. On rappelle que le générateur Z d'un itemset fermé X est un sous-ensemble minimal de X tel que sa fermeture est égale à X (Pfaltz et al., 2002).

DynaClose parcourt l'ensemble de générateurs des motifs fermés fréquents par niveaux. Dans la première étape, nous initialisons les 1-générateurs aux 1-itemsets qui représentent l'union de tous les attributs probablement indexables (ce sont les éléments des colonnes représentant la matrice du contexte). A chaque itération, l'algorithme considère un ensemble de k-itemsets générateurs.

Si à une étape K d'extraction on obtient un itemset fréquent fermé qui n'a aucun attribut non clé, dans ce cas, notre fonction de fitness sera égale au support. Notons qu'on n'élague pas à l'étape K un itemset fréquent fermé qui n'a pas d'attribut non clé car il pourrait former un itemset fermé fréquent avec un attribut non clé dans l'étape (K+1).

Nous réitérons ce processus jusqu'à ce qu'on ne puisse plus générer de k-générateurs, ce qui constituera notre critère d'arrêt. Les motifs fermés fréquents obtenus sont triés selon leur fonction de fitness.

Exemple 2

Pour illustrer notre démarche, nous l'appliquons sur l'exemple 1 de la section 3.1. La matrice de 1-générateurs est la suivante.

1-Générateur	support	α	*fitness*	1-itemset fermé
A1	0,4	0,6448	0,12896	A1, A2, A3
A2	0,4	0,6448	0,12896	A1, A2, A3
A3	0,4	0,6448	0,12896	A1, A2, A3
A4	0,6	0,0003	0,0001	A4, A5, A6
A5	0,6	0,0003	0,0001	A4, A5, A6
A6	0,6	0,0003	0,0001	A4, A5, A6

FIGURE 3 : Matrice de 1-générateurs

Si minsup=0,6 la démarche d'extraction classique (Aouiche et al., 2005) élague trois attributs, à savoir A1, A2 et A3 car ils ne sont pas fréquents (leur support est inférieur à minsup). Notre approche doit élaguer trois attributs comme dans l'approche classique. Pour identifier ces attributs, nous prenons ceux qui ont une faible valeur de fitness, à savoir, A4, A5 et A6. La deuxième étape de notre approche consiste à générer les 2-itemsets fréquents à partir des 1-générateurs fréquents. Chaque attribut de 1-génération est associé à tous les autres 1-items set fréquents. En conséquence, tous les 2-itemsets sont inclus dans le 1-itemsets fermés fréquents, à savoir A1A2A3 généré précédemment et l'algorithme s'arrête.

Lorsque deux itemsets possèdent la même valeur fitness, on privilégie celui qui a la plus grande valeur de support.

3.4 Etablissement de la liste des index candidats

La liste des itemsets fermés fréquents générée par *DynaClose* doit être épurée pour éviter les index erronés. Cette élimination concerne tout itemset fréquent fermé ne respectant pas les caractéristiques de l'index binaire de jointure en étoile. Rappelons qu'un index de jointure est construit entre la table des faits et les tables de dimension sur leurs attributs non clés. Nous distinguons trois cas d'épuration :

1. Si DynaClose génère des itemsets fréquents formés seulement de clés primaires des tables de dimension ou des clés étrangères de la table des faits, ces derniers seront éliminés dans la liste des itemsets fréquents.
2. Les itemsets fréquents ne respectant pas les caractéristiques des index de jointure en étoile. Prenons l'exemple suivant : considérons un itemset fréquent formé de (customers.cust_gender, sales.cust_id, customers.cust_id, sales.prod_id, products.prod_id) contenant trois attributs clés et un non clé. Si on veut construire l'index de jointure binaire, nous obtenons :

```
CREATE BITMAP INDEX sales_c_gender_p_cat_bjix
ON sales(customers.cust_gender)
FROM sales, customers, products
```

WHERE sales.cust_id = customers.cust_id
AND sales.prod_id = products.prod_id

Cette instruction est fausse car on a un attribut dans la clause ON et deux jointures dans le clause WHERE. Normalement, on doit avoir une seule jointure.

3. Si un itemset fréquent ne contient que les attributs non clés.

3.5 Génération des index

Rappelons que les itemsets fréquents générés par DynaClose représentent des attributs d'index candidats. Pour des contraintes de stockage et de maintenance, les attributs de sélection figurant dans les itemsets fréquents ne peuvent pas tous être utilisés pour l'indexation. Notre problème de sélection des index de jointure binaires est alors formalisé de la façon suivante : étant donné :

- Un entrepôt de données modélisé par un schéma en étoile formé d'une table des faits F et de $D = \{D_1,...,D_d\}$ tables de dimensions.
- Un ensemble de requêtes les plus fréquentes $Q = \{q_1,...,q_m\}$ avec leurs fréquences d'accès $f = \{f_1,..., f_m\}$.
- Une configuration d'index de jointure binaires candidats générés par DynaClose $I = \{I_1,..., I_p\}$
- Une capacité de stockage S pour stocker les index sélectionnés.

L'objectif est de sélectionner un ensemble d'index réduisant le coût d'exécution de requêtes et satisfaisant la contrainte S.

3.5.1 Le modèle de coût d'exécution de requêtes en présence d'index

Pour évaluer la qualité des index sélectionnés, nous avons développé un modèle de coût calculant le nombre d'entrées sorties (IOs) nécessaires dans l'évaluation de l'ensemble de requêtes.

L'évaluation d'une requête q_i en présence des index de jointure binaires s'exécute selon les trois scénarii suivants :

- *Pas de correspondance* **:** Aucun index n'est pertinent pour la requête. Pour évaluer le coût de cette requête, nous utilisons la jointure par hachage. Notons que le coût de cette dernière entre deux tables T_j et T_i en terme d'entrée sorties est donné par : $3 \times (|T_i| + |T_j|)$ (Ramakrishman, 1998), où $|T_i|$ représente le nombre de pages occupées par la table T_i.
- *Correspondance totale* **:** C'est le cas où un ou plusieurs index de jointure couvrent toutes les opérations de jointure de la requête. Pour estimer le coût de cette requête, nous utilisons le modèle de coût proposé par (Aouiche, 2005).
- *Correspondance partielle* **:** C'est le cas où l'index ne couvre que certaines tables. Il est alors utilisé pour traiter les tables qu'il couvre (comme pour la correspondance totale). Le résultat est combiné avec celui obtenu en effectuant

des jointures classiques sur les tables non couvertes par l'index (comme dans le cas de non correspondance).

Ainsi le coût total d'exécution de l'ensemble des requêtes est donné par :

$CT = \sum_{i=1}^{m} f_i \times C(q_i)$, où $C(q_i)$ représente le coût d'exécution de la requête q_i.

4 Algorithme itératif de sélection

Dans cette section, nous décrivons notre algorithme de sélection des index de jointure binaires. Notre algorithme est basé sur un modèle de coût mathématique comme dans (Cheonni et al., 1993) (voir section 2.1). Cet algorithme itératif a deux étapes principales :

1. **Etape de présélection** : éliminer tous les index de jointure binaires qui ne respectent pas la contrainte d'espace disque S. Notons par $S(I_j)$ le coût de stockage nécessaire pour l'index I_j.
2. **Etape de sélection de la configuration finale** : dans cette étape, nous commençons par une configuration initiale qui comprend l'index réduisant le plus le coût d'exécution de requêtes. Ensuite, nous enrichissons d'une manière itérative cette configuration en considérant d'autres index, tant que le coût de stockage n'est pas dépassé. Les détails de l'algorithme sont décrits ci-dessous.

Etape 1 : présélection

1. Construction d'une matrice de coût C, tel que C[i,j] correspond au coût d'exécution de la requête i en utilisant l'index $I_j \in config_{candidats}$

2. Calcul du coût total d'exécution pour l'ensemble Q pour chaque index I_j : $C_{j,Q} = \sum_{i=1}^{m} C[i, j]$ (en utilisant le modèle de coût dans la section 3.5).

3. Eliminer tout index ne respectant pas la contrainte d'espace S :

Pour tout index $I_j \in config_{candidats}$ avec $S(I_j) > S$ faire

$config_{candidat} = config_{candidat} - I_j$

$S(config_{finale}) = 0.$

Choisir un index $I_{\min} \in config_{candidat}$ avec $C(I_{\min}) = \min(C_{j,Q})$

$config_{finale} := I_{\min}$;

$S(config_{finale}) = S(I_{\min})$;

$config_{candidat} = config_{candidat} - I_{\min}$;

Etape 2 : Sélection de la configuration finale d'index

Tant que $S(config_{finale}) \leq S$ faire

Pour tout index $I_j \in config_{candidat}$ faire

Si (($config_{finale} \cup I_j$) $\leq C(config_{finale})$ et $S(config_{finale} \cup I_j) \leq S$) alors

$config_{finale} := config_{finale} \cup I_j$

$config_{candidat} = config_{candidat} - I_j$

$S(config_{finale}) = S(config_{finale}) + S(I_j)$

FIN.

5 Evaluation expérimentale

Notre étude expérimentale utilise un schéma en étoile composé d'une table des faits Sales et de cinq tables dimensions *Customers*, *Products*, *Times*, *Promotions* et *Channels* dont la taille a été indiquée en sous-section 2.2. Nous avons utilisé aussi une charge formée de 40 requêtes décisionnelles comme dans (Aouiche, 2005). Notre programme de simulation est implémenté avec le langage Java exécuté sous PC Pentium IV, d'une fréquence de 1,5 Ghz et une mémoire de 256 Mo.

Nos expérimentations se sont déroulées selon trois scénarii : (1) l'identification des valeurs de minsup qui permettent de donner des itemsets fréquents, (2) l'évaluation de notre démarche sans considération pour l'espace de stockage, en testant les 40 requêtes sur l'entrepôt non indexé, indexé avec l'approche classique (basée sur les fréquences seulement, indexé avec notre approche et (3) l'évaluation de notre approche avec la contrainte d'espace de stockage où les trois cas précédents sont considérés (entrepôt non indexé, indexé avec l'approche classique et indexé avec notre approche).

Nous avons d'abord effectué des expérimentations pour fixer la valeur de minsup pertinente pour l'extraction d'itemsets fermés fréquents et les index de jointure. Les résultats ont montré que la valeur 0,05 constitue une borne qui permet d'avoir le nombre le plus important d'itemsets fermés fréquents et la valeur 0,5, où aucun index n'est généré.

5.1 Evaluation sans contrainte d'espace

La Figure 4 montre les performances des trois algorithmes d'indexation. Avec un Minsup=0,05, nous avons obtenu un gain de 69,66 % pour les deux stratégies sur la stratégie non indexée. Ce taux paraît très important étant donné le nombre élevé d'itemsets fermés extraits et de ce fait le nombre important d'index de jointure à considérer. Notre approche améliore le coût d'exécution de l'ensemble de requêtes pour des valeurs petites de minsup. Cette amélioration se dégrade pour des grandes valeurs de minsup qui diminuent le nombre d'index. Notre approche avec pénalité améliore légèrement les performances (par rapport à l'approche d'Aouiche), principalement pour les valeurs de minsup se situant entre 0,075 et 0,375.

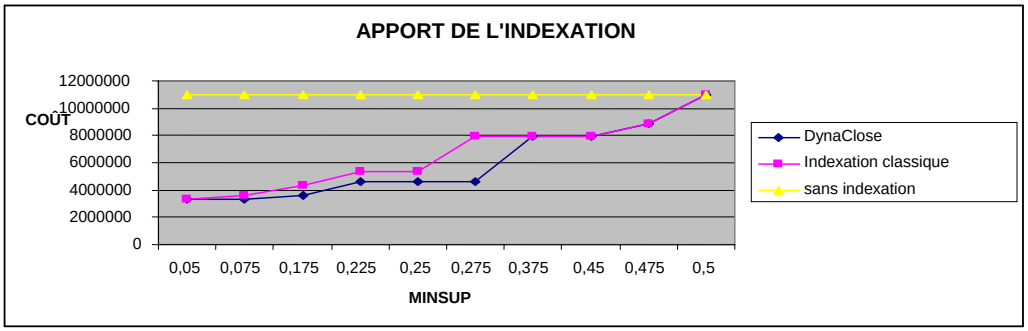

FIGURE 4 : Comparaison de DynaClose avec les approches existantes

5.2 Evaluation de DynaClose avec contrainte d'espace

Notons que l'ensemble d'index de jointure candidats généré par DynaClose occupe un espace de stockage de 146 Mo[4]. Cette valeur est très élevée si on la compare à la taille de la table des faits qui est de 372,17 Mo. En conséquence, nous avons exécuté l'algorithme de sélection d'index de jointure binaires avec une contrainte de stockage, à savoir 20 Mo. Nous avons considéré l'ensemble de 40 requêtes et les mêmes valeurs de minsup que précédemment (voir Figure 4).

La Figure 5 montre une nette amélioration avec un gain de 33,56 % pour un espace de stockage de près de 7 fois plus petit que l'espace initial (146 Mo).

Notre approche fournit une infime amélioration des performances par rapport à l'approche classique (Aouiche et al.) uniquement pour minsup=0,075 (un pourcentage de 33,56% pour 27,4% pour l'approche classique).

Pour conclure, notre algorithme nous permet de faire un meilleur choix entre différents index candidats. Cependant, d'autres tests expérimentaux sont nécessaires.

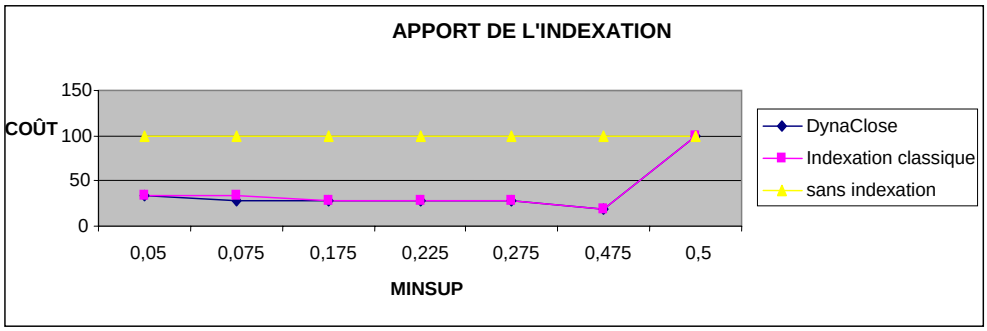

FIGURE 5 : Pourcentage de gain avec contrainte d'espace

[4]Nous avons adapté le modèle défini dans (Bellatreche, et al., 2002) pour estimer l'espace de stockage nécessaire pour les index de jointure sélectionnés.

6 Conclusion

Dans cet article, nous avons proposé une démarche qui utilise une technique de fouille de données pour sélectionner automatiquement une configuration d'index à partir d'une charge de requêtes. Notre stratégie de sélection procède en deux grandes étapes. Compte tenu du nombre important des attributs des tables de dimension qui peuvent participer à la construction des index de jointure binaires, nous utilisons l'algorithme DynaClose pour en réduire le nombre. Le critère d'élagage diffère de l'approche d'Aouiche qui utilise uniquement la fréquence d'apparition des motifs. Nous enrichissons ce critère en prenant en compte un facteur pénalisant relatif au nombre de n-uplets des différentes tables de dimension par rapport à la table des faits.

Sur cet ensemble d'attributs, nous avons proposé un algorithme itératif de sélection des index de jointure sous la contrainte de l'espace de stockage. Cet algorithme est basé sur un modèle de coût qui calcule le nombre d'entrées-sorties nécessaires pour exécuter un ensemble de requêtes fréquentes.

Notre approche a été validée par des expérimentations. Les résultats ont permis de tirer des résultats préliminaires assez encourageants. Toutefois, d'autres expérimentations sur des bancs d'essai plus volumineux et plus complexes s'imposent.

Il serait également intéressant de voir l'impact de la sélection des motifs maximaux (plutôt que fermés) et du raffinement de la fonction de pénalité (intégrant d'autres facteurs telles que la sélectivité de l'attribut à indexer et la sélectivité des jointures) sur le choix des index et sur la performance des requêtes.

Comme il existe des similarités entre le problème de la sélection d'index et celui du choix d'autres structures d'optimisation comme les vues matérialisées et la fragmentation horizontale (Sanjay et al., 2004) (Bellatreche, 2000), il serait intéressant d'adapter notre démarche à la matérialisation de vues et à la fragmentation dans les entrepôts de données.

Références

Agrawal, R., Srikant, R. (1994). Fast Algorithms for Mining Association Rules in Large Databases. International Conference on Very Large Databases (VLDB'1994), pp. 487-499, September, 1994, Santiago de Chile.

Aouiche K. (2005) Techniques de fouille de donnée pour l'optimisation automatique des performances des entrepôts de données, Thèse de doctorat, Université Lumière Lyon2.

Aouiche, K., Darmont, J., Boussaid, O., Bentayeb, F. (2005). "Automatic Selection of Bitmap Join Indexes in Data Warehouses", International Conference on Data Warehousing and Knowledge Discovery (DAWAK'05), pp. 64-73, August 2005, Copenhagen, Denmark.

Bastide Y., Taouil R., Pasquier N., Stumme G., Lakhal L. (2000a). Pascal: un algorithme d'extraction des motifs fréquents, Technique et Science Informatique, 21(1/2002), pp. 65-95, 2002.

Bastide Y., Taouil R., Pasquier N., Stumme G., Lakhal L., (2000b) Mining frequent patterns with counting inference », SIGKDD Explorations, 2(2), 2000, p. 66-75.

Bellatreche L. (2000). Utilisation des vues matérialisées, des index et de la fragmentation dans la conception logique et physique d'un Entrepôt de données, Thèse de Doctorat, Université de Clermont-ferrant II, France.

Bellatreche, L., Karlapalem, K., Li, Q. (2000). Evaluation of materialized view indexing in data warehousing environments. International Conference on Data Warehousing and Knowledge Discovery (DAWAK'00), pp. 57-66, September 2000, London, UK.

Chauhuri, S., Datar, M., Narasayya, V. R. (2004). Index selection for databases: a hardness study and a principled heuristic solution. IEEE Transactions Knowledge on Data Engineering, 16(11), pp. 1313-1323, November, 2004.

Chaudhuri. S, Narasayya, V. R. (1997). An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. International Conference on Very Large Data Bases (VLDB 1997), pp. 146-155, August, 1997, Athens, Greece.

Choenni S., Blanken H.M., Chang T. (1993). On the Selection of Secondary Indices in Relational Databases. Data Knowledge Engineering, 11(3), pp. 207–238.

Feldman Y.A., Reouven J. (2003). A knowledge–based approach for index selection in relational databases, Expert System with Applications, 25(1), pp. 15–37, July.

Fung, C.-W., Karlapalem, K., Li, Q. (2003). Cost-driven vertical class partitioning for methods in object oriented databases, VLDB Journal, 12(3), pp. 187-210, October, 2003.

Getoor, L., Taskar, B., Koller, D. (2003). Selectivity Estimation using Probabilistic Models. International Conference on Management of Data (SIGMOD'01), pp. 461-472, June 2001, San Diego, California, USA.

Gundem T.I. (1993). Near optimal multiple choice index selection for relational databases. Computers & Mathematics with Applications, 37(2), pp.111–120, 1993.

Gupta H. (1999). Selection and maintenance of views in data warehouse. Ph.d. thesis, Stanford University, September 1999.

Johnson, T. (1999). Performance measurements of compressed bitmap indices. International Conference on Very Large Data Bases (VLDB99), pp. 278-289, September 1999, Edinburgh, Scotland

Netz, A., Chaudhuri, S., Bernhardt, J., Fayyad U. (2000). Integration of Data Mining with Database Technology, International Conference on Very Large Data Bases (VLDB'00), pp. 719-722, September, 2000, Cairo, Egypt.

O'Neil. P, D. Quass. (1997). Improved query performance with variant indexes. International Conference on Management of Data (SIGMOD'1997), pp. 38-49, May 1997, Arizona, USA.

Pasquier. N., Bastide Y., Taouil R., Lakhal L. (1999). Discovering Frequent Closed Itemsets, Proceedings of 7th International Conference on Database Theory (ICDT'99), pp. 398-416, January 1999, Jerusalem, Israel.

Pfaltz JL., Taylor CM. (2002). Scientific Discovery through Iterative Transformations of Concept Lattices. Workshop on Discrete Applied Mathematics in conjunction with the 2nd SIAM International Conference on Data Mining, pages 65-74, Arlington, USA, 2002.

Rao, J., Zhang, C., Megiddo, N., Lohman, G.M. (2002). Automating physical database design in a parallel database, International Conference on Management of Data (SIGMOD'02), pp. 558-569, June, 2002, Wisconsin, USA.

Ramakrishman. R.(1998). Database Management Systems. WCB/McGraw Hill, 1998.

Red Brik System (1997). Star schema processing for complex queries. White Paper. July, 1997

Rizzi S., Saltarelli E. (2003). View Materialization vs. Indexing : Balancing Space Constraints in Data Warehouse Design », in 15th International Conference on Advanced Information Systems Engineering (CAiSE 2003), pp. 502–519, June 2003, Klagenfurt, Austria.

Sanjay. A, Narasayya, V. R., Yang, B. (2004). Integrating vertical and horizontal partitioning into automated physical database design. Proceedings of the International Conference on Management of Data (SIGMOD04), pp. 359–370, June 2004, Paris.

Stumme G, Taouil R., Bastide Y., Pasquier N., Lakhal L. (2001). Intelligent structuring and reducing of association rules with formal concept análisis. Advances in Artificial Intelligence, Joint German/Austrian Conference on AI, p. 335-350, September 2001, Vienna, Austria.

Valduriez, P., Gardarin, G. (1984). Join and Semijoin Algorithms for a Multiprocessor Database Machine. ACM Transaction Database Systems 9(1), pp. 133-161, June, 1984.

Summary

Indexing is a redundant technique used to speed up OLAP queries. Two main types of indexing techniques are available: mono-index (B-tree, bitmap index, projection, etc.) and multi-index (join indexes). Bitmap join indices are usually used to accelerate queries defined on a star schema having a large number of join and selection operations. Selecting an optimal set of bitmap join indexes is a very hard problem due to the number of candidate attributes that can be used in the index selection process. To deal with this problem, we propose the following methodology: (1) an adaptation of a data mining algorithm called Close that generates frequent closed itemsets representing candidate attributes for the index selection process. (2) A selection algorithm that identifies indices using a subset of attribute candidates. These indices should minimize the query processing cost and satisfy the storage constraint. Finally, we validate our proposed algorithm using an experimental evaluation.