Vers l'auto-sélection des index dans les entrepôts de données: une approche basée sur la recherche des motifs fréquents maximaux

ZIANI Benameur - OUINTEN Youcef

Laboratoire d'Informatique et de Mathématiques (LIM)
Université Amar Télidji
Laghouat Algérie
{bziani, ouinteny}@mail.lagh-univ.dz

RÉSUMÉ. La sélection des index dans les entrepôts de données est l'une des tâches les plus importantes à réaliser par un administrateur afin d'optimiser les performances du système en minimisant le temps d'accès aux données. Dans ce travail nous proposons une approche pour la sélection automatique des index de jointure binaires basée sur la recherche des motifs fréquents maximaux. Dans une première étape, nous avons sélectionné et implémenté un algorithme de recherche des motifs fréquents maximaux (*fpmax*)[2] afin de pouvoir valider notre approche. L'impléméntation réalisée est ensuite appliquée à une charge de requêtes pour générer automatiquement une configuration d'index. Enfin, nous avons appliqué la charge de requêtes sur un entrepôt de données test crée à l'aide du banc d'essai ABP-1. Les expérimentations menées montrent que la configuration d'index générée par l'algorithme implémenté permet un important gain de performance.

ABSTRACT. Index sélection is one of the fastidious tasks that a data warehouse administrator has to perform in order to reduce time access to data, and hence improve the performance of a system. This paper deals with an automatic selection of Bitmap Join Indices based on mining maximal frequent itemsets. The algorithm (*fpmax*)[2] for mining maximal frequent itemsets was selected and implemented in java. This implementation is then applied on a given workload to automatically determine an index configuration. The queries of the workload are executed using the generated configuration on a test data warehouse created from the ABP-1 benchmark. The obtained results show that the index configuration generated by the implemented algorithm allowed a significative improvement of the execution time of the workload.

MOTS-CLÉS : Entrepôt de données, index de jointure binaire, motifs fréquents maximaux

KEYWORDS : Data Warehouse, Bitmap join index, Maximal frequent itemsets

1. Introduction

Les entrepôts de données sont souvent modélisés selon un schéma en étoile contenant une table de faits centrale volumineuse et un ensemble de tables de dimension représentant les descriptions des faits[19][22]. Les tables de dimension contiennent un nombre important d'attributs représentant des données qualitatives supportant un ensemble de processus d'analyse. La table des faits contient un nombre très important d'instances. Chaque n-uplet dans la table des faits contient deux types d'attributs : (1) un ensemble de clés étrangères référençant les tables de dimension et (2) un ensemble de mesures qui peuvent être agrégées pour effectuer certains traitements. Les requêtes définies sur ce schéma sont appelées requêtes de *jointure en étoile*. Ces requêtes comportent un nombre de prédicats de sélection définis sur des tables de dimension et des prédicats de jointures entre la table de faits et les tables de dimension. De telles requêtes sont très coûteuses en temps de calcul. La réduction de ce temps devient cruciale lorsque ces jointures opèrent sur des tables volumineuses. Une des techniques utilisées pour optimiser les requêtes de jointure en étoile est l'*indexation*. Dans le contexte des entrepôts de données l'indexation constitue une option importante dans la phase de conception physique[23]. Plusieurs techniques d'indexation ont été proposées dans le cadre des entrepôts de données relationnels. Certaines de ces techniques héritent de celles proposées dans le cadre des base de donées traditionnelles comme les *arbres B*. D'autres sont proposées pour optimiser les sélections définies sur des attributs de faibles cardinalités comme les *index binaire*. Les *index de jointure binaires* ont été proposés pour optimiser simultanément les sélections et les jointures. Le problème de sélection d'index est connu NP-Complet[10]. De ce fait, il n'existe pas des algorithmes proposant une solution optimale en un temps fini. Plusieurs travaux de recherche proposent des solutions proches de la solution optimale en utilisant des heuristiques réduisant la complexité du problème. Les algorithmes proposés pour la sélection d'index comportent généralement deux étapes : (1) la détermination des attributs candidats à l'indexation et (2) la construction d'une configuration finale d'index.

Dans ce travail nous présentons une approche de sélection automatique des index de jointure binares basée sur la technique de recherche des motifs fréquents maximaux. Il est organisé comme suit : Dans la deuxième section nous présentons brièvement les principaux travaux proposés pour résoudre le problème de la sélection d'index de jointure binaires. Nous mettons l'accent sur les travaux exploitant les techniques de fouille de données pour résoudre ce problème. La troisième section expose la technique de recherche des motifs fréquents maximaux. Nous exposons également les raisons pour lesquelles nous avons choisi l'algorithme *fpmax* que nous avons implémenté en java. La quatrième section présente les expérimentations réalisées pour valider notre approche. Pour terminer, nous concluons et indiquons quelques perspectives dans la cinquième section.

2. sélection des index dans les entrepôts de données

2.1. Introduction

Le problème de sélection d'index consiste à construire une configuration d'index optimisant le coût d'exécution d'une charge de requêtes donnée sous certaines contraintes comme l'espace de stockage alloué par le système aux index crées. La sélection automatique des index a fait l'objet de plusieurs études[16][20][11][15][13][3]. L'objec-

tif étant de faciliter les tâches manuelles de l'administrateur du système. La sélection des index peut être réalisée à partir d'un ensemble d'index candidats. Ces index candidats sont extraits à partir d'une charge de requêtes en faisant appel à l'expertise de l'administrateur[16][8][3] ou d'une manière automatique en réalisant une analyse syntaxique des requêtes de la charge [7][21][13]. D'autre travaux ont adapté les algorithmes génétiques au problème de sélection d'index[15]. Dans le contexte des entrepôts de données, les index de jointure binaires sont bien adaptés pour optimiser les requêtes de jointures en étoile[24]. Cependant, la sélection de ces index est un problème difficile[25] vu le nombre important des attributs potentiels candidats pour la construction des index. Afin de réduire ce nombre, des approches récentes[3][6] proposent l'utilisation des techniques de fouille de données (Data Mining) pour générer l'ensemble des index candidats. Elles ont exploité en particulier la technique de recherche des motifs fréquents fermés pour la génération des index candidats. L'idée de base est inspirée de la technique de recherche des motifs fréquents, très utilisée en fouille de données. Si un attribut ou un groupe d'attributs est fréquemment présent dans un ensemble de requêtes, il est considéré comme intéressant dans le processus de sélection des index. La réalisation d'une configuration finale d'index à partir des index candidats utilisent généralement des modèles de coût[3].

2.2. Approches basés sur les motifs fréquents fermés

Le problème de sélection des index de jointure binaire peut être formulé comme suit :

– Etant donné :

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

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

3) Une contrainte de stockage S.

– L'objectif est de trouver une configuration d'index CI minimisant le coût d'exécution de Q et respectant la contrainte de stockage $(Taille(CI) \leq S)$.

Aouiche et al.[3] ont traité ce problème en proposant une approche basé sur la technique de recherche des motifs fréquents fermés avec l'algorithme *Close* pour élaguer l'espace de recherche des index de jointure binaires. Ils prennent en considération la fréquence d'utilisation d'un attribut dans une charge de requêtes comme facteur déterminant dans sa candidature à la configuration finale des index. Bellatreche et al.[6][17] ont montré à travers un exemple que la fréquence d'accès seule ne permet pas de sélectionner un ensemble d'index efficace. En effet, les index de jointure binaires sont crées pour optimiser des jointures entre la table des faits et les tables de dimension. L'approche de Aouiche et al. 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. Les auteurs proposent l'algorithmes *DynaClose* qui est une adaptation de l'algorithme *Close*. L'algorithme proposé s'appuit sur une fonction permettant de pénaliser les attributs fréquents définis sur des tables de petites tailles. Vu le nombre important des motifs généré par la technique des motifs fréquents fermés, nous proposons d'exploiter les motifs fréquents maximaux. Ces derniers sont moins nombreux que les motifs fréquents fermés ce qui va réduire l'espace de recherche des index candidats. En plus, un motif fréquent maximal a la particularité que tous ses sur-ensembles

sont non fréquents. Cette particularité va permettre de générer des index moins nombreux et plus pertinents.

3. Recherche des motifs fréquents maximaux

Le problème de la découverte des motifs fréquents est un problème important dans le domaine de la fouille de données. Il est introduit pour la première fois par Agrawal et al. avec l'algorithme Apriori[18]. Cet algorithme identifie les i-motifs fréquents à la ième itération puis génère les $(i+1)$-motifs fréquents à patir des i-motifs fréquents identifiés. Cependant, la phase de génération de candidats reste très coûteuse, plus particulièrement si le nombre de motifs est important et/ou la taille des motifs est grande. A titre d'exemple, s'il existe un motif fréquent de longueur l, tous les 2^l sous-ensembles de ce motif sont générés. C'est pourquoi il est plus intéréssant de se focaliser sur la recherche des motifs fréquents maximaux. Un motif fréquent maximal est un motif fréquent dont tous les sur-ensembles sont non fréquents.

Le problème de recherche des motifs fréquents maximaux peut être formulé comme suit : Soit $I = \{i_1, i_2, \ldots, i_m\}$ un ensemble de m symboles distintcts appelés *articles* ou *items* et $T = \{t_1, t_2, \ldots, t_n\}$ un ensemble de n transactions qui constituent la base d'origine. Un ensemble $X = \{i_1, i_2, \ldots, i_k\} \subseteq I$ est appelé un *motif* ou *k-motif* s'il contient k articles. Le support d'un motif X est le nombre de transactions de la base T contenant X divisé par le nombre total des transactions

$$Sup(X) = \frac{|\{t \in T / X \subseteq t\}|}{|T|}$$

Soit *minsup* un seuil minimum pour le support. Un itemset X est dit fréquent si $Sup(X) \geq minsup$. L'ensemble des motifs fréquents est ainsi :

$$MF = \{X \subseteq I | Sup(X) \geq minsup\}$$

Un motif fréquent est dit *maximal* si tous ses sur-ensembles sont non fréquents. L'ensemble des motifs fréquents maximaux est alors :

$$MFM = \{X \subseteq I | (Sup(X) \geq minsup) \wedge (\nexists Y \subseteq I | Sup(Y) \geq minsup \wedge X \subset Y)\}$$

Il est clair que $MFM \subseteq MF$. L'objectif des travaux sur la recherche des motifs fréquents maximaux est de déterminer d'une manière efficace l'ensemble MFM. Les tableaux 1 et 2 présentent respectivement des exemples d'une base de transactions et de motifs fréquents maximaux recherhés dans cette base. La majorité des algorithmes de recherche

TID	Items
1	ACTW
2	CDW
3	ACTW
4	ACDW
5	ACDTW
6	CDT

Tableau 1. *Exemple de transactions*

Taille du Motif Maximal	Motif. Maximal(Min_Sup=2)
1	–
2	–
3	CDT
4	ACDW, ACTW

Tableau 2. *Exemple de Motifs fréquents maximaux*

des motifs fréquents maximaux s'inspirent de la méthode FP-Growth[14]. Cette méthode permet de générer les motifs fréquents en parcourant seulement deux fois la base

des transactions en s'appuyant sur une structure arborescente : le FP-Tree (Frequent Pattern Tree). Les meilleurs algorithmes de recherche des motifs fréquents maxiamux sont présentés dans le workshop sur les différentes implémentations dans ce domaine (FIMI'04)[1] qui été organisé conjointement avec ICDM'04 (Fourth IEEE International Conference on Data Mining). L'algorithme *fpmax* est présenté comme étant le plus performant. Il a été ainsi choisi pour notre propre implémentation. Nous avons présenté dans[2] le principe de fonctionnement de cet algorithme ainsi qu'une implémentation en java.

4. Approche de sélection

L'approche proposée est constituée des étapes suivantes :

1) sélection d'une charge de requêtes, supposée représentative, de l'activité du du système ;

2) strcucturation des attributs indéxables contenus dans la charge sous forme d'une base tansactionnelle ou les requêtes représentent les transactions et les attributs représentent les motifs. Ceci représente notre contexte d'extraction des motifs fréquents maximaux ;

3) génération des index candidats par l'algorithme *fpmax* implémenté en java ;

4) Comparaison des temps de réponses de l'exécution des requêtes sans index puis avec les index générés par l'impléméntation de *fpmax*.

4.1. Expérimentations

Afin de tester l'approche proposée nous avons procédé à une étude expérimentale qui s'est déroulée selon les étapes suivantes :

1) Imlpémentation de l'algorithme *fpmax* en java. A part sa portabilité, java est choisi pour sa gestion automatique de la mémoire. Cette caractéristique et cruciale car les structures de données manipulées sont principalement des listes chaînées et des arbres. Avant d'être appliquée au contexte de sélection, notre implémentation est testée sur un ensemble de données réelles et synthétiques disponibles à cet effet[1]. Les performances de notre implémentation comparées aux meilleures implémentations du workshop cité ci-dessus[1] sont présentées dans [2].

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

Tableau 3. *Caractériqtiques des tables de l'entrepôt utilisé*

2) Création d'un entrepôt de données à l'aide du banc d'essai ABP-1[9]. Cet entrepôt est composé d'une table de faits *Actvars* et quatre tables de dimension *ProdLevel*, *TimeLevel*, *CustLevel* et *ChanLevel*. La table 3résume les caractéristiques des tables formant l'entrepôt. Nous avons considéré une charge de 60 requêtes de jointure en étoile définies sur cet entrepôt. La charge des requêtes a été choisie pour couvrir l'ensemble

des attributs de l'entrepôt. On a considéré un ensemble de requêtes utilisant plusieurs prédicats de sélection définis sur un ou plusieurs attributs. Ces requêtes appartiennent à plusieurs catégories : requêtes du type *count(*)* avec et sans agrégations, requêtes utilisant les fonctions d'agrégations comme *Sum, Min, Max*, requêtes ayant des attributs de dimension dans la clause *SELECT*,...etc.

3) Après génération de l'entrepôt nous avons crée le contexte d'extraction. Il s'agit d'un fichier texte sous forme d'une matrice ou chaque ligne désigne une requête de la charge. Les colonnes définissent les attributs candidats pour la procédure d'indexation. A ce contexte nous avons appliqué notre imlpementation de l'algorithme *fpmax*. Nous avons choisi un seuil minimum de 90% afin de sélectionner les attributs candidats qui figurent ensemble dans la quasi totalité des requêtes. Ceci signifie qu'ils sont très utilisés dans l'historique du système et sont ainsi des candidats intéréssents pour l'opération d'indexation.

4) Dans la dernière étape nous avons procédé à l'exécution de la charge des requêtes sur l'entrepôt généré selon deux sénarii (1) sans création des index (2) après création des index générés par l'algorithme *fpmax*. Les outils Oracle 10g et Aqua Data Studio 7.0 ont été utilisé pour exécuter la charge sur l'entrepôt crée.

4.2. Résultats

Pour les deux sénarii considérés, nous avons calculé le temps d'exécution de chaque requête de la charge. La figure 1 montre les temps enregistrés pour l'exécution des re-

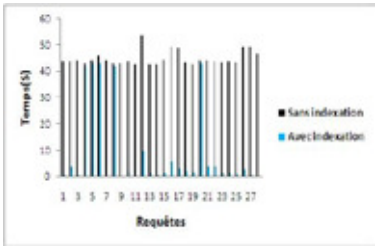

Figure 1. *Temps d'exécution des requêtes avec et sans indexation*

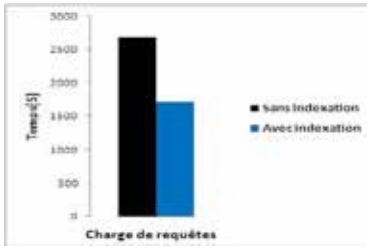

Figure 2. *Temps global d'execution de la charge avec et sans indexation*

quêtes qui ont bénéficié des index crées. Nous remarquons que pour une bonne partie de la charge le temps d'exécution est nettement amélioré après création des index. La figure 2 illuste le gain de temps global d'exécution pour l'ensemble de la charge considérée. Les résultats obtenus confirment la grande utilité des index de jointure binaires pour les requêtes du type *Count(*)*. Le temps d'exécution des ces requêtes est nettement inférieur avec l'utilisation des index générés car ce type de requêtes ne nécessite aucun accès aux données (l'accès aux index crées suffit pour répondre à ces requêtes). Les requêtes les moins bénéficiaires des index générés sont celles nécéssitant des jointures supplémentaires entre la table des faits et les tables de dimension.

5. Conclusions et Perspectives

Les index de jointures binaires sont des structures d'optimisation permettant de précalculer les jointures en étoile. Nous avons présenté dans ce papier une approche basée

sur la recherche des motifs fréquents maximaux pour la sélection des index de jointure binaires dans les entrepôts de données relationnels. Un algorithme de recherche des motifs fréquents maximaux a été choisi et implémenté afin de pouvoir tester et valider notre approche. Les premiers résultats montre un interêt particulier des index préconisés par l'algorithme implémenté. Notons que dans cette première expérimentation nous n'avons pas considéré ni la contrainte d'espace ni d'autre paramètre de sélection. Il serait intéréssant de prendre en considération plusieurs paramètres pour générer la configuration finale d'index. Nous pensons en particulier aux facteurs de sélectivité et aux cardinalités des attributs ou encore les tailles des tables de dimension. Une amélioration possible consisite à considérer un modèle de coût prenant en considération tous ces facteurs. Ce modèle permettera une évaluation théorique de la stratégie proposée qui sera ensuite validée sous un SGBD tel que Oracle.

6. Bibliographie

[1] http ://fimi.cs.helsinki.fi/

[2] B. ZIANI, Y. OUINTEN, « Mining Maximal Frequent Itemsets : a java implementation of FP-MAX algorithm », *6th International Conference on Innovations in Information Technology (IIT09)*, 2009.

[3] K. AOUICHE, « Techniques de fouille de données pour l'optimisation automatique des performances des entrepôts de données », *Université Lumière Lyon 2*, 2005,

[4] S.G. AZEFACK, « Sélection automatique des index dans les entrepôts de données », *Université Lumière Lyon 2*, 2007.

[5] L. BELLATRECHE, « Techniques d'optimisation des requêtes dans les datawerhouses », *LISI/ENSMA*, 2003.

[6] L.BELLATRECHE, R. MISSAOUI, H. NECIR, « A data Mining Approach for Selecting Bitmap Join Indices », *Journal of Computer Science and Enginnering V(N)*, 2007.

[7] V.R. NARASAYYA, S. CHAUDHURI, « An efficient cost-driven index sélection tool for Microsoft QSL Server », *23rd International Conference on Very Large Data Bases*, 1997.

[8] T. CHANG, S. CHOENNI, H.M BLANKEN, « Index sélection in relational data bases », *5th International Conference on Computing and Information (ICCI93)*, 1993.

[9] O. COUNCIL, « Apb-1 OLAP benchmark, release ii. http ://www.olapcouncil.org/research/resrchly.htm »,

[10] C.D, « The difficulty of optimum index sélection », *ACM transactions on Database Systems (TODS)*, 1978.

[11] Y. FELDMEN, J. REOUVEN, « A knowledge-based approach for index sélection in relational databases », *Expert System sith Applications*, n° 25, 2003.

[12] G. GRAHNE, J. ZHU, « Efficiently using prefix-trees in mining frequent itemsets », *IEEE ICDM Workshop on Frequent Itemsets Mining Implementations*, 2003.

[13] S. RIZZI, J. GOLFARELLI, E. SALTARELLI, « Index sélection for Datawarhousing », *4th International Workshop on Design end Management of Data Warhouses (BMDW2002)*, 2002.

[14] J. PEI, J. HAN, Y. YIN, « Mining Frequent Patterns without candidate generation », *ACM SIGMOD 2000*, 2000.

[15] I. LJUBIC, J. KRATICA, D. TOSIC, « A genetic algorithm for index sélection problem », *Application of Evolutionary Computing*, n° 2611, 2003.

[16] E. OMIECINSKI, M. FRANK, S. NAVATHE, « Adaptive and automated index sélection in RDBMS », *3rd International Conference on Extending Database Technology (EDBT92)*, n° 580, 1992.

[17] H. NECIR, L.BELLATRECHE, R. MISSAOUI, « DynaClose : Une approche de Data Mining pour la sélection des index de jointures binaires dans les entrepôts de données », *EDA*, 2007.

[18] A.N SWAMI, R. AGRAWAL, T. IMIELINSKI, « Mining Association Rules between sets of items in lagre Databases », *SIGMOD*, 1993.

[19] R. KIMBALL, M. ROSS, « The Data Warehouse Toolkit : The complete guide to dimentional modeling », *John Wiley Sons*, 2002.

[20] S. CHAUDHURI, S. AGRAWAL, V. NARASAYYA, « Automated sélection of materialized views and indexes in SQL databases », *26th International Conference on Very Large Databses (VLDB2000)*, 2000.

[21] D. ZILIO, G. LOHMAN, A. SKELLEY, G. VALENTIN, M. ZULIANI, « Db2advisor : An optimizer smart enough to recommend its own indexes », *16th International Conference on Data Engineering (ICDE 2000)*, 2000.

[22] W. INMON, « Building the Data Warehouse », *John Wiley Sons*, 2002.

[23] S. CHAUDHURI, V.R. NARASAYYA, « Self-tuning database systems : A decade of progress », *International Conference on Very Large Data Bases*, 2007.

[24] T. STOHR, H. MARTENS,, E. RAHM « Multi-dimensional database allocation for parallel data warehouses », *International Conference on Very Large Data Bases*, 2000.

[25] S. CHAUHURI, M Datar,, V.R. NARASAYYA « Index selction for database : A hardness study and a principled heuristic solution », *IEEE Transaction Knowledge on Data Enginnering*, 2004