HAL open science

Vers l'auto-administration des entrepôts de données

Kamel Aouiche, Jérôme Darmont, Le Gruenwald

To cite this version:

Kamel Aouiche, Jérôme Darmont, Le Gruenwald. Vers l'auto-administration des entrepôts de données. Revue des Nouvelles Technologies de l'Information, 2003, 1, pp.1-12. hal-01448644

HAL Id: hal-01448644

https://hal.science/hal-01448644v1

Submitted on 28 Jan 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Vers l'auto-administration des entrepôts de données

Kamel Aouiche*, Jérôme Darmont*
Le Gruenwald**

*Laboratoire ERIC, Équipe BDD
Université Lumière – Lyon 2
5 avenue Pierre Mendès-France
69676 BRON Cedex
{kaouiche, jdarmont}@eric.univ-lyon2.fr
**School of Computer Science
University of Oklahoma
Norman, OK 73019, USA
ggruenwald@ou.edu

Résumé. Avec le développement des bases de données en général et des entrepôts de données (*data warehouses*) en particulier, il est devenu primordial de réduire la fonction d'administration de base de données. L'idée d'utiliser des techniques de fouille de données (*data mining*) pour extraire des connaissances utiles des données elles-mêmes pour leur administration est avancée depuis quelques années. Pourtant, peu de travaux de recherche ont été entrepris dans ce domaine. L'objectif de cette étude est de rechercher une façon d'extraire, à partir des données stockées, des connaissances utilisables pour appliquer de manière automatique des techniques d'optimisation des performances, et plus particulièrement d'indexation. Nous avons réalisé un outil qui effectue une recherche de motifs fréquents sur une charge donnée afin de calculer une configuration d'index permettant d'optimiser le temps d'accès aux données. Les expérimentations que nous avons menées ont montré que les configurations d'index générées par notre outil permettent des gains de performance de l'ordre de 15% à 25% sur une base et un entrepôt de données tests.

1 Introduction

L'utilisation courante de bases de données requiert un administrateur qui a pour rôle principal la gestion des données au niveau logique (définition de schéma) et physique (fichiers et disques de stockage), ainsi que l'optimisation des performances de l'accès aux données. Avec le déploiement à grande échelle des systèmes de gestion de bases de données (SGBD), minimiser la fonction, d'administration est devenu indispensable (Weikum et al. 2002).

L'une des tâches importantes d'un administrateur est la sélection d'une structure physique appropriée pouvant améliorer les performances du système en minimisant les temps d'accès aux données (Finkelstein et al. 1988). Les index sont des structures physiques permettant un accès direct aux données. Le travail d'optimisation des performances de l'administrateur se porte en grande partie sur la sélection d'index et de

vues matérialisées (Agrawal et al. 2001, Gupta 1999). Ces structures jouent un rôle particulièrement important dans les bases de données décisionnelles (BDD) tels que les entrepôts de données, qui présentent une volumétrie très importante et sont interrogés par des requêtes complexes.

Depuis quelques années, l'idée est avancée d'utiliser les techniques de fouille de données (*data mining*) pour extraire des connaissances utiles des données elles-même pour leur administration (Chaudhuri 1998). Cependant, peu de travaux de recherche ont été entrepris dans ce domaine jusqu'ici. C'est pourquoi nous avons conçu et réalisé un outil qui utilise la fouille de données pour proposer une sélection (configuration) d'index pertinente.

Partant de l'hypothèse que l'utilité d'un index est fortement corrélée à la fréquence de l'utilisation des attributs correspondants dans l'ensemble des requêtes d'une charge donnée, la recherche de motifs fréquents (Agrawal et al. 1993) nous a semblé appropriée pour mettre en évidence cette corrélation et faciliter le choix des index à créer. L'outil que nous présentons dans cet article exploite le journal des transactions (ensemble de requêtes résolues par le SGBD) pour proposer une configuration d'index.

Dans la suite de cet article, nous présentons tout d'abord brièvement les principales méthodes de sélection automatique d'index existantes dans la Section 2. Nous détaillons ensuite notre proposition dans la Section 3, avant d'exposer quelques résultats expérimentaux dans la Section 4. Nous concluons finalement cet article et présentons nos perspectives de recherche dans la Section 5.

2 État de l'art

Le problème de la sélection d'un ensemble d'index optimal pour une base de données a été étudié depuis les années 70. Trois grandes approches ont été proposées (Chaudhuri et al. 1997). La première exploite le schéma de la base de données (clés primaires et étrangères) et des statistiques rudimentaires ("petite" ou "grande" table), sans prendre en compte la charge supportée par le système. La deuxième approche est basée sur l'utilisation de systèmes experts, la base de règles définissant un "bon" modèle physique. Cette approche souffre de la complexité de modélisation de cette base de règles. Enfin, la troisième approche utilise l'optimiseur de requêtes du SGBD pour estimer le coût de différentes configurations d'index candidats.

Les travaux les plus récents en matière de sélection d'index utilisent cette dernière approche. (Frank et al. 1992) propose un outil d'aide à la décision pour l'administrateur qui, à partir d'une charge et d'une configuration d'index données, fournit une estimation du coût global de chaque index grâce à un dialogue constant avec l'optimiseur de requêtes. L'outil de sélection d'index IST - *Index Selection Tool* (Chaudhuri et al. 1997, Chaudhuri 1998, Agrawal et al. 2000), développé par Microsoft au sein du SGBD SQL Server, exploite une charge et en extrait une configuration d'index candidats mono-attribut. Un algorithme glouton permet de sélectionner les meilleurs index de cette configuration grâce à des estimations de coût effectuées par l'optimiseur de requêtes. Le processus est ensuite réitéré pour générer des index sur deux attributs à partir des index mono-attributs, et ainsi de suite pour les index multi-attributs de taille supérieure.

3 Extraction de motifs fréquents pour la sélection d'index

3.1 Principe

L'approche que nous proposons, dont le principe est représenté à la Figure 1, exploite le journal des transactions (fichier log) pour extraire une configuration d'index. Les requêtes présentes dans le journal constituent une charge, qui est traitée par un analyseur de requêtes SQL qui extrait tous les attributs susceptibles d'être indexés (attributs indexables).

Nous construisons ensuite une matrice dite "requêtes-attributs" qui représente le contexte d'extraction des motifs fréquents. Pour obtenir ces derniers, nous avons choisi l'algorithme Close (Pasquier et al. 1999a), car il calcule l'ensemble des fermés (au sens de la connexion de Galois (Pasquier et al. 1999b)) fréquents, qui est un générateur pour tous les motifs fréquents et leur support. Dans la plupart des cas, le nombre des fermés fréquents est sensiblement moins important que la totalité des motifs fréquents obtenue en sortie des algorithmes classiques tels qu'Apriori (Agrawal et al. 1994). Dans notre contexte, l'utilisation de Close nous permet d'obtenir une configuration d'index candidats moins volumineuse (tout en demeurant aussi significative) en un temps de calcul plus court. Finalement, nous sélectionnons au sein de la configuration d'index candidats les plus pertinents et les créons.

3.2 Extraction de la charge

Nous supposons que nous disposons d'une charge similaire à celle présentée dans la Figure 2. Cette charge peut normalement être obtenue à partir du journal des transactions du SGBD hôte ou bien grâce à une application externe telle que Log Explorer (Lumigent Technologies 2002).

3.3 Extraction des attributs indexables

Pour réduire le temps de réponse d'une recherche dans une base de données, il est judicieux de construire des index sur les attributs utilisés pour effectuer cette recherche. Ces attributs sont ceux qui font partie des clauses WHERE, ORDER BY, GROUP BY et HAVING des requêtes SQL (Chaudhuri et al. 1997).

L'analyseur syntaxique que nous avons conçu opère sur tous types de requête SQL (sélection et mise à jour) imbriquée ou non, et en extrait tous les attributs indexables. Par exemple, l'analyse de la requête de la Figure 3 renvoie comme résultat les attributs suivants : *part.partkey, lineitem.partkey, part.brand, part.container,* et *lineitem.quantity.* Ce processus est appliqué à toutes les requêtes de la charge.

3.4 Construction du contexte d'extraction des fermés fréquents

Nous construisons ensuite une matrice "requêtes-attributs" (Figure 4) dont les lignes représentent les différentes requêtes de la charge et les colonnes représentent l'ensemble de tous les attributs indexables identifiés à l'étape précédente. Cette matrice

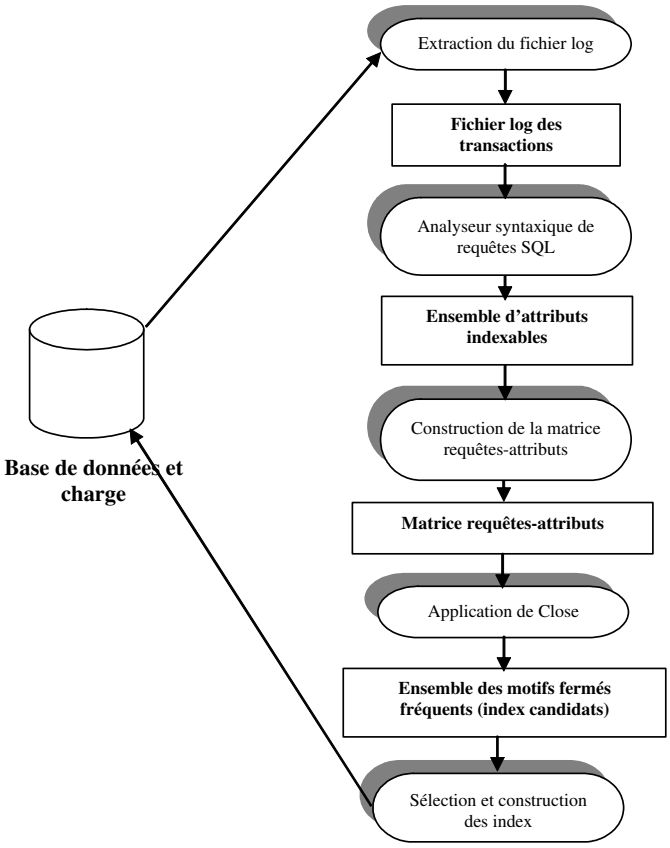

FIG. 1 – Principe de la sélection des index en utilisant les motifs fréquents

sert à associer à chaque requête les attributs indexables qu'elle comporte. L'existence d'un attribut dans une requête est symbolisée par 1 et son absence par 0.

3.5 Extraction des motifs fréquents fermés

L'algorithme Close réalise un parcours en largeur du treillis des fermés pour en extraire les fermés fréquents et leurs supports. Il utilise en entrée un contexte d'extraction tel que celui de la Figure 4, sur lequel sont réalisés des balayages successifs. Un paramètre régulateur (seuil), le support minimal (*minsup*), permet de déterminer quels sont les fermés fréquents (c'est-à-dire, de support supérieur ou égal à *minsup*).

L'application de Close sur le contexte de la Figure 4 fournit en sortie l'ensemble de fermés fréquents (et leurs supports) suivant, pour un support minimal de 2/6 : {(AC, 3/6), (BE, 5/6), (C, 5/6), (ABCE, 2/6), (BCE, 4/6)}.

```
Q1 :  SELECT * FROM T1, T2 WHERE A BETWEEN 1 AND 10 AND C=D
Q2 :  SELECT * FROM T1, T2 WHERE B LIKE '%this%' AND C=5 AND E<100
Q3 :  SELECT * FROM T1, T2 WHERE A=30 AND B>3 GROUP BY C HAVING
      SUM(E)>2
Q4 :  SELECT * FROM T1 WHERE B>2 AND E IN (3, 2, 5)
Q5 :  SELECT * FROM T1, T2 WHERE A=30 AND B>3 GROUP BY C HAVING
      SUM(E)>2
Q6 :  SELECT * FROM T1, T2 WHERE B>3 GROUP BY C HAVING SUM(E)>2
```

FIG. 2 – Exemple de charge

```
SELECT SUM(lineitem.extendedprice) / 7.0 FROM lineitem, part
WHERE part.partkey = lineitem.partkey
AND part.brand = ' :1' AND part.container = ' :2'
AND lineitem.quantity < (
       SELECT 0.2 * AVG(lineitem.quantity) FROM lineitem
       WHERE lineitem.partkey = part.partkey )
```

FIG. 3 – Exemple de requête SQL

3.6 Construction des index

Le nombre d'index candidats obtenu avec notre approche est d'autant plus important que la charge en entrée est volumineuse. En pratique, il est donc impossible de construire tous les index proposés. Le temps de construction, puis plus tard de mise à jour de tous ces index serait pénalisant. Il faut donc prévoir des méthodes ou des procédés de filtrage permettant de réduire le nombre d'index à générer.

Dans le contexte des bases de données décisionnelles, plus particulièrement dans celui des entrepôts de données, la construction d'un index est un problème fondamental vue la volumétrie importante des tables. Il semble donc plus intéressant de construire des index sur les tables volumineuses, l'apport d'un index sur une table de petite taille

	Attributs				
Requêtes	A	B	C	D	E
Q1	1	0	1	1	0
Q2	0	1	1	0	1
Q3	1	1	1	0	1
Q4	0	1	0	0	1
Q5	1	1	1	0	1
Q6	0	1	1	0	1

FIG. 4 – Contexte d'extraction pour la charge de la Figure 2

s'avérant peu significatif.

Certaines connaissances statistiques comme la cardinalité des attributs à indexer peuvent également être exploitées pour construire les index. La cardinalité d'un attribut est le nombre de valeurs distinctes de cet attribut dans une relation donnée. Si la cardinalité est très grande, l'index dégénère à un parcours séquentiel et si elle est très petite, l'indexation n'apporte pas d'amélioration significative (Vanichayobon et al. 1999). Il faut donc construire des index sur des attributs à cardinalité "moyenne".

Dans cette première étude, nous nous intéressons plus particulièrement au volume des tables. Pour cela, nous avons établi deux stratégies pour construire les index. La première consiste à construire systématiquement tous les index proposés (méthode naïve, qui demeure applicable quand le nombre d'index est réduit). Dans ce cas, chaque fermé fréquent correspond à un index à créer. La seconde stratégie prend en considération le volume des tables auxquelles se réfère l'index proposé. Dans ce cas, l'administrateur de la base de données défini des tables dites volumineuses et seuls les index portant sur des attributs de ces tables sont construits.

3.7 Comparaison avec les méthodes existantes

Contrairement aux méthodes de sélection d'index développées récemment (Section 2), l'outil que nous proposons présente l'originalité de ne pas communiquer avec l'optimiseur de requêtes du SGBD. Habituellement, les dialogues entre l'outil de sélection des index et l'optimiseur sont pénalisants et doivent être minimisés. En effet, le temps de calcul du coût d'une configuration d'index est d'autant plus important que la charge est conséquente, ce qui est typiquement le cas. Notre méthode à base de recherche de motifs fréquents est également gourmande en temps de calcul, mais il nous est actuellement difficile de déterminer quelle approche engendre la surcharge la plus lourde pour le système.

Cependant, nous nous intéressons plus encore à la qualité des index générés. En utilisant la recherche des fréquents fermés, notre outil extrait directement un ensemble d'index mono-attribut et multi-attributs. Nous ne construisons donc pas *a priori* une configuration d'index mono-attribut initiale et n'avons pas besoin d'utiliser une heuristique pour construire par itérations successives les index candidats multi-attributs, comme c'est le cas avec IST (Chaudhuri et al. 1997). Nous pensons que cette approche évite la génération et l'évaluation du coût d'index non pertinents.

4 Expérimentations

Afin de valider notre démarche, nous l'avons appliquée sur une base et un entrepôt de données tests. Notre objectif est ici de vérifier si nos propositions s'avèrent intéressantes en pratique, plus que d'effectuer de véritables tests de performance. Nous avons choisi le banc d'essais décisionnel TPC-R (TPC 1999) pour nos expérimentations sur base de données relationnelle, car c'est un standard qui nous permettra par la suite de comparer aisément notre approche aux autres méthodes existantes. Nous l'avons configuré pour générer une base de données de 1 Go.

En revanche, il n'existe pas à notre connaissance de banc d'essais standard pour les entrepôts de données. Aussi avons-nous travaillé sur un petit magasin de données développé par ailleurs au sein de notre laboratoire. Nous avons de plus conçu une charge décisionnelle spécialement adaptée pour l'appliquer sur ce magasin (Aouiche 2002). Le magasin de données d'accidentologie est composé d'une table de faits *Accidents* et des tables dimensions : *Lieux, Conditions, Dates* et *Responsables*. Il occupe une place disque de 15 Mo.

```
// Exécution à froid (sans chronométrage)
POUR Chaque requête de la charge FAIRE
    Exécuter la requête courante
FIN POUR
// Exécution à chaud
POUR i = 1 A nombre_de_répétitions FAIRE
  POUR Chaque requête de la charge FAIRE
    Exécuter la requête courante
    Calculer le temps de réponse de la requête courante
  FIN POUR
FIN POUR
Calculer le temps de réponse moyen global
```

Fig. 5 – Protocole de test

La base de données de TPC-R et le magasin de données d'accidentologie ont été implantés au sein du SGBD SQL Server 2000. Le protocole de test que nous avons adopté est présenté à la Figure 5. Cet algorithme est exécuté pour différentes valeurs du paramètre *minsup* (support minimal) de Close. En pratique, ce paramètre nous permet de limiter le nombre d'index à générer en ne sélectionnant que ceux qui sont les plus fréquemment sollicités par la charge.

4.1 Expérimentations avec base TPC-R

Les résultats obtenus sont représentés à la Figure 6 et 7. Elles montrent que nous réalisons un gain de performance quelle que soit la valeur de *minsup*. Le gain maximum en temps de réponse est la différence entre le temps de réponse moyen sans index et le plus petit temps de réponse moyen avec index. Il est proche de 22% dans le premier cas et de 25% dans le deuxième cas. Les gains moyens en temps de réponse sont de 14,4% et 13,7%, respectivement. Dans le premier cas, le temps de réponse s'améliore jusqu'à ce que *minsup* atteigne 15%, puis se dégrade de manière continue. Dans le deuxième cas, il reste à sa valeur minimale dans un large intervalle (valeur de *minsup* comprise entre 20% et 50%) avant de se dégrader brutalement. Le nombre important d'index à générer dans le premier cas peut expliquer ce comportement. Considérer uniquement les index associés à des tables volumineuses permet de réduire le nombre d'index générés et évite la création d'index sur les "petites" tables (puisqu'ils n'apportent qu'un faible bénéfice).

Finalement, pour les grandes valeurs de *minsup*, le temps de réponse moyen dans les deux cas est proche de celui obtenu sans générer aucun index. Ce cas de figure est prévisible, dans le sens où, pour un *minsup* très grand, aucun index ou très peu d'index sont générés. Dans le deuxième cas, cet état est atteint rapidement car très peu d'index sont construits, ce qui explique le gain moyen inférieur.

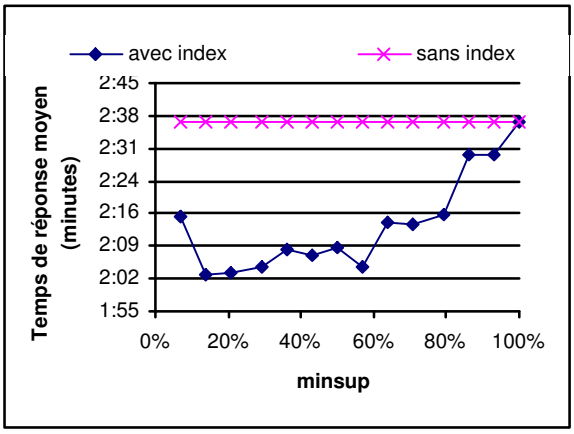

FIG. 6 – Résultats avec TPC-R — Tous les index

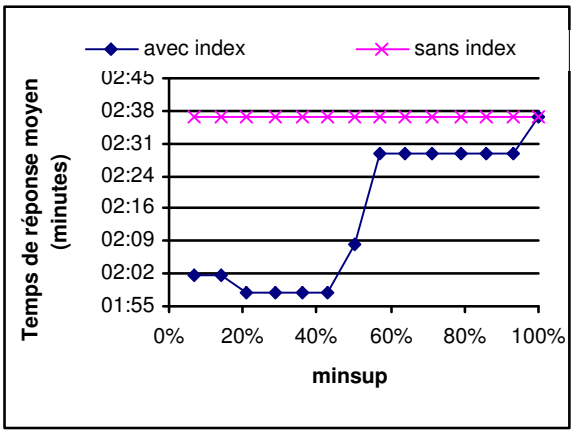

FIG. 7 – Résultats avec TPC-R — Index sur des tables volumineuses

4.2 Expérimentations sur le magasin de données

Pour cette série de tests, nous avons appliqué le même protocole (Figure 5). Cependant, nous n'avons pas employé la stratégie de création d'index sur des tables

volumineuses, étant donné que toutes les tables de notre magasin de données test ont une taille similaire.

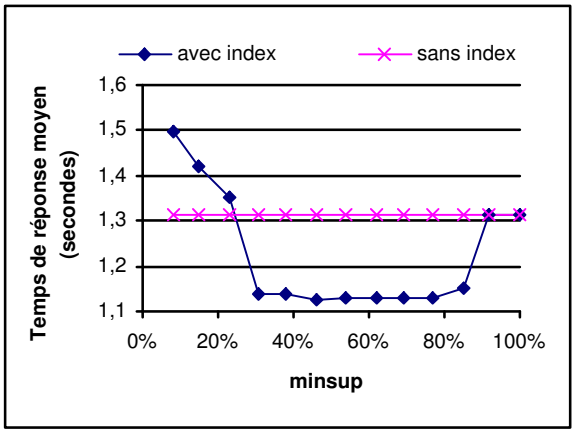

Fig. 8 – Résultats avec le magasin de données d'accidentologie

Les résultats obtenus sont représentés à la Figure 8. Le gain de performance maximum est proche de 15%. Le gain moyen est de 6,4%. La Figure 8 montre que construire les index est plus coûteux que de ne pas le faire pour les valeurs de *minsup* comprises entre 10% et 25%. Ceci peut s'expliquer par le nombre important d'index à générer, qui induit une surcharge importante. En outre, comme le magasin de données (de 15 Mo) tient entièrement en mémoire centrale, les index ne sont utiles qu'à son premier chargement. Dans ce contexte, de nombreux index peu utilisés doivent être chargés, ce qui pénalise les performances globales.

Le meilleur gain en temps de réponse apparaît pour les valeurs de *minsup* comprises entre 30% et 85%, quand le nombre d'index est tel que la surcharge due à la génération d'index est moins importante que le gain de performance obtenu au chargement du magasin de données. Pour les plus grandes valeurs de *minsup*, le temps de réponse se dégrade et devient proche de celui obtenu sans index car aucun ou peu d'index sont générés.

Une autre explication possible aux performances inférieures dans le cas du magasin de données par rapport à la base de données de TPC-R peut venir de la structure de l'index créé. Les index *bitmap* et les index de jointure en étoile sont les plus adaptés pour un entrepôt de données (O'Neil et al. 1995). Or, les index créés par défaut dans SQL Server sont des variantes des B-arbres.

5 Conclusions et perspectives

Nous avons présenté dans cet article une nouvelle approche pour la sélection automatique d'index dans un SGBD. L'originalité de notre travail repose sur l'extraction de motifs fréquents pour déterminer une configuration d'index. Nous nous basons en effet

sur l'intuition que l'importance d'un attribut à indexer est fortement corrélée avec sa fréquence d'apparition dans les requêtes présentes dans une charge. Par ailleurs, l'utilisation d'un algorithme d'extraction de fréquents tel que Close nous permet de générer des index mono-attribut et multi-attributs à la volée, sans avoir à mettre en œuvre un processus itératif permettant de créer successivement des index multi-attributs de plus en plus gros à partir d'un ensemble d'index mono-attribut.

Nos premiers résultats expérimentaux montrent que notre technique permet effectivement d'améliorer le temps de réponse de 20% à 25% pour une charge décisionnelle appliquée à une base de données relationnelle (banc d'essais TPC-R). Nous avons par ailleurs proposé deux stratégies effectuer une sélection parmi les index candidats : la première crée systématiquement tous les index candidats et la deuxième ne crée que les index associés à des tables dites volumineuses. La deuxième stratégie apporte une meilleure amélioration car elle propose un compromis entre l'espace occupé par les index (le nombre d'index créés est limité à ceux qui sont définis sur des attributs de tables volumineuses) et l'intérêt de la création d'un index (il est peu intéressant de créer un index sur une petite table).

Nous avons également réalisé des tests sur un petit magasin de données d'accidentologie auquel nous avons appliqué une charge décisionnelle ad hoc. Le gain en temps de réponse, de l'ordre de 14%, est moins important que dans le cas de TPC-R. Cela peut être expliqué par le fait que les index créés par défaut par SQL Server sont des variantes des B-arbres et non des index *bitmap* et des index de jointure en étoile, qui seraient plus adaptés pour un entrepôt de données.

Notre travail démontre que l'idée d'utiliser des techniques de fouille de données pour l'auto-administration des SGBD est prometteuse. Il n'est cependant qu'une première approche et ouvre de nombreuses perspectives de recherche. Une première voie consisterait à améliorer la sélection des index en concevant des stratégies plus élaborées que l'utilisation exhaustive d'une configuration ou l'exploitation de renseignements relativement basiques concernant la taille des tables. Un modèle de coût plus fin au regard des caractéristiques des tables (autres que la taille), ou encore une stratégie de pondération des requêtes de la charge (par type de requête : sélection ou mise à jour), pourraient nous aider dans cette optique. L'utilisation d'autres méthodes de fouille de données non-supervisées telles que le regroupement (*clustering*) pourraient également fournir des ensembles de fréquents moins volumineux.

Par ailleurs, il paraît indispensable de continuer à tester notre méthode pour mieux évaluer la surcharge qu'elle engendre pour le système, que ce soit en terme de génération des index ou de leur maintenance. Il est notamment nécessaire de l'appliquer pour des entrepôts de données de grande taille et en tirant partie d'index adaptés. Il serait également très intéressant de la comparer de manière plus systématique avec l'outil IST développé par Microsoft, que ce soit par des calculs de complexité des heuristiques de génération de configurations d'index (surcharge) ou des expérimentations visant à évaluer la qualité de ces configurations (gain en temps de réponse et surcharge due à la maintenance des index).

Finalement, étendre ou coupler notre approche à d'autres techniques d'optimisation des performances (vues matérialisées, gestion de cache, regroupement physique, etc.) constitue également une voie de recherche prometteuse. En effet, dans le contexte

des entrepôts de données, c'est principalement en conjonction avec d'autres structures physiques (principalement les vues matérialisées) que l'indexation permet d'obtenir des gains de performance significatifs (Gupta 1999, Agrawal et al. 2000, Agrawal et al. 2001).

Références

Agrawal R., Imielinski T. et Swami A.N. (1993), Mining Association Rules between Sets of Items in Large Databases, SIGMOD Record, 22(2), pp 207-216.

Agrawal R. et Srikant R. (1994), Fast Algorithms for Mining Association Rules, Proceedings of the 20th International Conference on Very Large Data Bases, Santiago, Chile, pp 487-499.

Agrawal S., Chaudhuri S. et Narasayya V.R. (2000), Automated Selection of Materialized Views and Indexes in SQL Databases, Proceedings of the 26th International Conference on Very Large Data Bases, Cairo, Egypt, pp 496-505.

Agrawal S., Chaudhuri S. et Narasayya V.R. (2001), Materialized View and Index Selection Tool for Microsoft SQL Server 2000, Proceedings of the ACM SIGMOD International Conference on Management of Data, Santa Barbara, USA.

Aouiche K. (2002), Accidentology datamart schema and workload, http ://bdd.univ-lyon2.fr/download/charge-accidentologie.pdf.

Chaudhuri. et Narasayya V.R. (1997), An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server, Proceedings of the 23rd International Conference on Very Large Data Bases, Athens, Greece, pp 146-155.

Chaudhuri S. (1998), Data Mining and Database Systems : Where is the Intersection ?, Data Engineering Bulletin, 21(1), pp 4-8.

Chaudhuri S. et Narasayya V.R. (1998), AutoAdmin 'What-if' Index Analysis Utility, Proceedings of the ACM SIGMOD International Conference on Management of Data, Seattle, USA, pp 367-378.

De Marchi F., Lopes S. et Petit J.M. (2002), Efficient Algorithms for Mining Inclusion Dependencies, Proceedings of the 8th International Conference on Extending Database Technology, Prague, Czech Republic, LNCS, Vol. 2287, pp 464-476.

Finkelstein S. J., Schkolnick M. et Tiberion P. (1988), Physical Database Design for Relational Databases, TODS, (13)1, pp 91-128.

Frank M. R., Omiecinski E. et Navathe S.B. (1992), Adaptive and Automated Index Selection in RDBMS, Proceedings of the 3rd International Conference on Extending Database Technology, Vienna, Austria, LNCS, Vol. 580, pp 277-292.

Gupta H. (1999), Selection and maintenance of views in a data warehouse, PhD thesis, Stanford University.

Lopes S., Petit J.M. et Lakhal L. (2000), Efficient Discovery of Functional Dependencies and Armstrong Relations, Proceedings of the 7th International Conference on Extending Database Technology, Konstanz, Germany, LNCS, Vol. 1777, pp 350-364.

Lumigent Technologies (2002), Log Explorer for SQL Server, http ://www.lumigent.com.

O'neil P. et Graefe G. (1995), Multi-table joins through bitmapped join indices, SIGMOD Record, (24)3, pp 8-11.

O'neil P. et Quass D. (1997), Improved Query Performance with Variant Indexes, SIGMOD Record, (26)2, pp 38-49.

Pasquier N., Bastide Y., Taouil R. et Lakhal L. (1999), Discovering Frequent Closed Itemsets for Association Rules, Proceedings of the 7th International Conference on Database Theory, Jerusalem, Israel, LNCS, Vol. 1540, pp 398-416.

Pasquier N., Bastide Y., Taouil R. et Lakhal L. (1999), Efficient mining of association rules using closed itemset lattices, Information Systems, (24)1, pp 25-46.

TPC Transaction Processing Council (1999), TPC Benchmark R Standard Specification.

Vanichayobon S. et Gruenwald L. (1999), Indexing Techniques for Data Warehouses's Queries, Technical repport, University of Oklahoma, School of Computer Science.

Weikum G., Monkeberg A., Hasse C. et Zabback P. (2002), Self-tuning Database Technology and Information Services : from Wishful Thinking to Viable Engineering, Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, China.

Summary

With the wide development of databases in general and data warehouses in particular, it is important to reduce the tasks that a database administrator must perform manually. The idea of using data mining techniques to extract useful knowledge for administration from the data themselves has existed for some years. However, little research has been achieved. The aim of this study is to search for a way of extracting useful knowledge from stored data to automatically apply performance optimization techniques, and more particularly indexing techniques. We have designed a tool that extracts frequent itemsets from a given workload to compute an index configuration that helps optimizing data access time. The experiments we performed showed that the index configurations generated by our tool allowed performance gains of 15% to 25% on a test database and a test data warehouse.