Sélection automatique d'index dans les entrepôts de données

Kamel Aouiche, Jérôme Darmont, Omar Boussaid

Laboratoire ERIC, Université Lumière – Lyon 2
5 avenue Pierre Mendès-France
69676 Bron Cedex
{kaouiche, jdarmont, boussaid}@eric.univ-lyon2.fr

Résumé. L'efficacité de l'interrogation d'un entrepôt de données est liée à sa conception physique. Cette conception repose sur la sélection d'index pertinents et leur combinaison avec les vues matérialisées. La sélection d'index est un problème NP–complet car le nombre d'index est exponentiel en nombre total d'attributs dans la base. Il faut donc concevoir et mettre en œuvre des méthodes permettant de réduire cette complexité afin de recommander un ensemble d'index (configuration). Dans cet article, nous proposons une méthode pour la sélection d'index basée sur la fouille de données (recherche des motifs fréquents, classification). Le contexte d'extraction de connaissances est construit après l'analyse des requêtes du journal des transactions exécutées. Les index de la configuration obtenue sont ensuite créés après une étape d'optimisation liée aux spécificités du SGBD utilisé.

1 Introduction

L'administrateur d'un Système de Gestion de Base de Données (SGBD) prend plusieurs décisions concernant des tâches d'administration, telles que la conception logique ou physique des bases de données, la gestion de l'espace de stockage, le réglage de performance (*performance tuning*), etc. L'une des plus importantes est la conception physique des bases de données, qui inclut l'organisation des données et l'amélioration de l'accès à ces données. Pour améliorer les temps d'accès, l'administrateur emploie en général des index pour rechercher rapidement les informations nécessaires à une requête sans parcourir toutes les données. Cependant, la mise à jour des données doit être propagée sur les index. Cela engendre un coût de maintenance supplémentaire.

La sélection d'index est difficile car leur nombre est exponentiel en nombre total d'attributs dans la base. C'est pourquoi nous nous intéressons au problème de sélection automatique d'un ensemble d'index (configuration) minimisant le coût d'exécution des requêtes. Dans ce sens, nous avons proposé une méthode de sélection d'index dans les bases de données basée sur des techniques de fouilles de données [ADG03a, ADG03b].

Par ailleurs, le problème de sélection d'index est aussi posé dans les entrepôts de données, mais certaines adaptations sont nécessaires. En effet, les entrepôts contiennent en général de très gros volumes de données. L'indexation de ces données donne donc lieu à des index volumineux. De plus, la sélection d'index se fait en conjonction avec la matérialisation des vues. L'espace nécessaire aux vues et aux index est de ce fait très grand. La sélection d'index doit donc se faire en prenant en compte l'espace de stockage disponible. D'autre part, les rafraîchissements de l'entrepôt sont réalisés périodiquement quand le système est hors ligne. Le coût de maintenance des index doit donc être pris en compte de manière différente que dans le cas des bases de données.

Dans cet article, nous étendons dans un premier temps notre approche grâce à une analyse plus fine des requêtes de la charge, à la détermination des techniques d'indexation les plus appropriées aux index, à l'établissement d'un ordre des attributs constituant les index multi–attributs et à la prise en compte des spécificités du SGBD utilisé lors de la création des index. Par la suite, nous proposons des adaptations dans le contexte des entrepôts de données afin de prendre en compte la contrainte sur l'espace disponible pour stocker les index et de sélectionner des techniques d'indexation propres aux entrepôts.

Cet article est organisé comme suit. Après un état de l'art sur les solutions proposées pour le problème de sélection d'index (Section 2), nous détaillons l'approche que nous proposons (Section 3). Nous terminons par une conclusion et des perspectives de recherche (Section 4).

2 État de l'art

2.1 Sélection d'index dans les bases de données

Le problème de sélection d'un ensemble d'index pour une base de données a été étudié depuis les années 70. Il est NP–Complet [Com78]. En effet, ce problème réduit à une seule table de n attributs candidats peut théoriquement être résolu en évaluant le coût de 2^n ensembles d'index possibles. Cela induit un coût de calcul exponentiel. Il s'agit donc de trouver un ensemble d'index proche de l'optimal sans évaluer le coût de tous ces ensembles.

On peut distinguer deux grandes familles de travaux proposant des solutions au problème de la sélection d'index. Dans la première famille, une fonction de calcul de coût basée sur un modèle mathématique est élaborée pour estimer le coût d'un ensemble d'index [ISR83, BMT85, BP90, CBC93a, CBC93b, KLT03, FR03]. Par exemple, Kratica *et al.* utilisent une optimisation par un algorithme génétique [KLT03]. Le système DINNER [FR03] exploite une base de connaissances et représente les requêtes d'une charge sous forme de graphes de solutions. Une solution décrit les chemins d'accès (avec index ou non) parcourus pour exécuter une requête. Il utilise ensuite plusieurs heuristiques pour élaguer les mauvaises solutions.

Dans la deuxième famille, l'optimiseur de requêtes du SGBD est utilisé pour évaluer le coût de chaque index [FST88, FON92, CN97, CN98, VZZ$^+$00, ACN00]. Frank *et al.* proposent un outil d'aide à la décision pour l'administrateur qui, à partir d'une charge et d'un ensemble d'index donnés, fournit une estimation du coût global de chaque index grâce à une communication constante avec l'optimiseur de requêtes [FON92]. L'outil de sélection d'index IST (*Index Selection Tool*) [CN97, CN98, ACN00] développé par Microsoft au sein du SGBD SQL Server, exploite une charge et en extrait une configuration d'index candidats mono–attributs. Un algorithme glouton permet de sélectionner les meilleurs index de cette configuration grâce à des estimations de coût effectuées par l'optimiseur de requêtes. Le processus est ensuite réitéré pour générer des index sur deux attributs à partir des index mono-attributs, et ainsi de suite pour les index multi-attributs de taille supérieure. Pour DB2, un algorithme de sélection d'index génère des index candidats pour chaque requête séparément, en injectant dans la base des index dits virtuels [VZZ$^+$00]. Ces index virtuels sont des sous–ensembles de tous les index possibles choisis par un algorithme dont les détails ne sont pas mentionnés. Les index proposés sur une table donnée sont combinés en les sélectionnant dans un ordre décroissant du ratio gain–espace de stockage.

2.2 Sélection d'index dans les entrepôt de données

Dans les entrepôts de données, deux familles d'algorithmes de sélection d'index existent : des algorithmes pour optimiser le temps de maintenance [LQA97] et d'autres pour optimiser le temps d'exécution des requêtes [GHRU97, ACN01, GRS02]. Dans les deux cas, l'optimisation est réalisée sous la contrainte de l'espace de stockage.

Gupta *et al.* proposent un algorithme glouton utilisant un modèle de coût [GHRU97]. Cet algorithme parcourt un multi–graphe bipartie pour recommander un ensemble de vues et d'index minimisant le coût d'exécution des requêtes. Ce multi–graphe relie les index et les vues aux requêtes où ils sont présents. Agrawal *et al.* proposent un outil implanté sous SQL Server [ACN01]. Ce dernier commence par énumérer tous les index et vues pouvant contribuer à la réduction du temps d'exécution d'un ensemble de requêtes donné. Une réduction du nombre de candidats (index et vues) est ensuite effectuée par l'optimiseur de requêtes. Un ensemble final d'index et de vues matérialisées est alors proposé. Une approche heuristique utilisant un algorithme glouton a également été proposée [GRS02]. Cet algorithme admet en entrée un ensemble de vues et d'index pour choisir progressivement les index les plus avantageux. Le choix est réalisé en comparant le coût des plans d'exécution possibles, générés par l'optimiseur, de chaque requête de la charge.

2.3 Motivation et discussion

La sélection d'index soit utilise une fonction mathématique, soit fait appel à l'optimiseur de requêtes du SGBD. Une fonction de coût est basée sur des hypothèses théoriques, par exemple, “la fréquence d'insertion et de suppression des n–uplets d'une table est telle que le nombre total des n–uplets dans cette table reste constant pour deux choix consécutifs d'un ensemble d'index”, ou “le nombre d'index n'est pas limité par table”, qui ne sont pas toujours réalisables dans la pratique. Par contre, l'optimiseur de requêtes utilise des statistiques et un modèle de coût inhérent à un SGBD donné. La sélection d'index est donc dépendante du SGBD et doit être adaptée pour un autre SGBD. De plus, le coût de communication avec l'optimiseur peut être important si le nombre d'index candidats à évaluer est grand.

L'approche de sélection d'index que nous proposons dans cet article est similaire à celle de Microsoft. Cependant, l'ensemble d'index candidats n'est pas obtenu par une heuristique qui fait constamment appel à l'optimiseur de requêtes mais en appliquant des techniques de fouille de données à la charge. Le résultat est directement une configuration d'index candidats dont le coût peut être évalué soit grâce à une fonction de coût soit par l'optimiseur de requêtes. Dans le second cas, le coût de communication avec l'optimiseur est réduit car le nombre d'index candidats à évaluer est moins important.

Dans les entrepôts, le problème devient plus crucial du fait de la grande volumétrie des données. Dans ce contexte, notre approche contribue efficacement à la sélection d'index en tenant compte de l'espace de stockage disponible.

3 Fouille de données pour la sélection d'index

L'approche que nous proposons (Figure 1) exploite le journal des transactions (ensemble de requêtes résolues par le SGBD) pour recommander une configuration d'index (ensemble d'index) améliorant le temps d'accès aux données.

La méthode proposée analyse la charge pour chercher les attributs pouvant être utiles lors de l'exécution d'une requête s'ils sont indexés. Cette analyse génère également un ensemble d'attributs sur lesquels il n'est pas souhaitable de construire des index. Les attributs trouvés précédemment sont stockés dans les matrices M_{Select} et M_{update}, respectivement. La combinaison de ces matrices permet de construire une matrice "requêtes–attributs". Dans les entrepôts de données, la matrice M_{update} n'existe pas car le rafraîchissement des entrepôts se fait hors ligne (*off line*). La matrice "requêtes–attributs" est donc construite directement. Parallèlement à l'analyse de la charge, nous déterminons la technique d'indexation appropriée à chaque index candidat. Les techniques d'indexation diffèrent suivant l'environnement utilisé : base ou entrepôt de données. La matrice "requêtes–attributs" correspond au contexte d'extraction utilisé pour générer les index de la configuration en utilisant les techniques de fouilles de données telles que la recherche des motifs fréquents ou la classification. Enfin, l'ordre des attributs dans les index multi–attributs est établi et un processus d'élagage est proposé avant de créer les index de la configuration. Nous détaillons chacune des ces étapes dans les sections suivantes.

3.1 Analyse de la charge

Les requêtes présentes dans la charge (journal des transactions) sont traitées par un analyseur de requêtes SQL afin d'extraire tous les attributs susceptibles d'être des supports d'index. Dans un environnement de base de données, la charge extraite est divisée en deux parties : une partie ne contenant que les requêtes d'interrogation (SELECT), notée $Part_{select}$ et une deuxième partie ne contenant que les requêtes de mises à jour (UPDATE, INSERT, DELETE), notée $Part_{update}$. Dans les requêtes des deux parties, le choix d'un attribut à indexer n'est pas dû au hasard. En effet, une requête d'interrogation comportant des jointures ou des sélections peut être très coûteuse si elle opère de manière séquentielle (sans utilisation d'index). Il est alors judicieux de construire des index sur les attributs utilisés pendant la recherche. Pour ces mêmes raisons, il est recommandé de créer des index sur les attributs servant à la mise à jour. Par contre, il n'est pas souhaitable de créer des index sur les attributs mis à jour (par exemple, les attributs d'une clause SET) car ils engendrent un coût de maintenance supplémentaire. Partant de ce constat, nous avons établi des règles pour recommander des attributs à indexer suivant chaque type de requête.

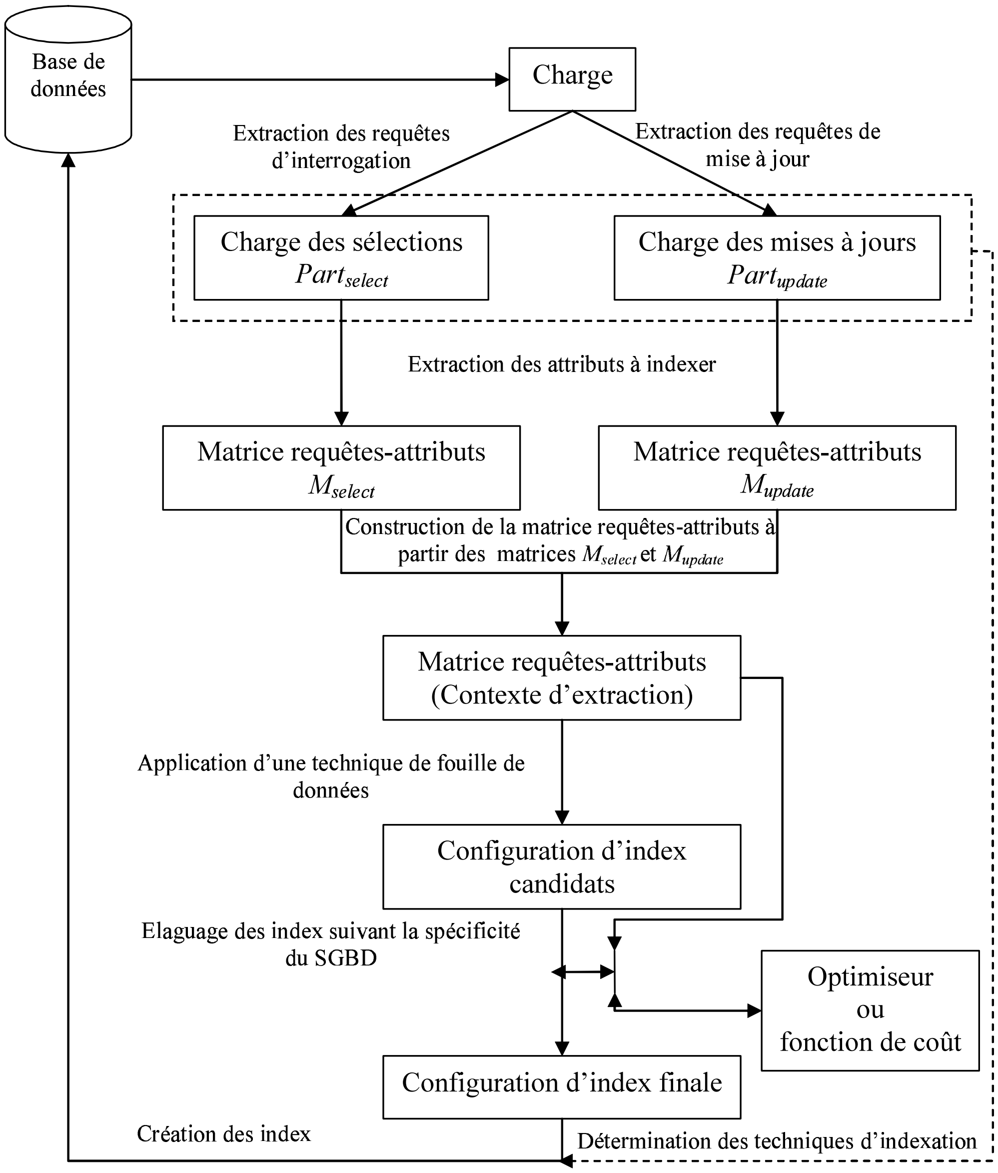

Fig. 1 – Démarche proposée pour la sélection d'index

Dans les entrepôts de données, les requêtes sont de type SELECT en lecture seule (*read only*) et les rafraîchissements (*batch update*) sont réalisées périodiquement [VG99]. Dans ce cas, nous construisons directement la matrice "requêtes–attributs".

Illustrons quelques règles à travers les exemples suivants. Soit un prédicat de restriction qui a l'une des formes suivantes : $t.a$ between v_1 and v_2, $t.a\,\theta\,v_2$, $t.a$ like v_1 ou $t.a$ in $(v_1, v_2, ...)$

où :

- t est un nom ou un alias d'une table,
- a est un attribut de cette table,
- θ est l'un des opérateurs $=$, $\neq$, $<$, $>$, $\leq$ ou $\geq$,
- v_1, v_2 sont des valeurs constantes ne contenant pas une spécification d'une table (nom ou alias d'une table).

Plusieurs cas peuvent conduire à proposer des index "inutiles", c'est–à–dire qui ne seront pas utilisés pour répondre aux requêtes précédentes.

1. Un prédicat de la forme $E(t.a) = v_1$ où $E(t.a)$ est une expression en $t.a$, par exemple $t.a^2+3t.a$. Dans un tel cas, $t.a$ n'est pas l'opérande immédiat de l'opérateur de comparaison. Le SGBD n'utilise pas d'index sur l'attribut a pour chercher les tuples vérifiant ce prédicat.
2. Un prédicat de la forme $t.a \neq v_1$ n'utilise pas un index construit sur a car toutes les données sont parcourues sauf v_1 si elle existe dans la table.
3. Un prédicat de comparaison de chaînes de caractères utilisant like n'exploite pas l'existence d'un index pour chercher les n–uplets vérifiant ce prédicat si la recherche s'effectue sur une sous–chaîne (par exemple, dans le cas d'utilisation d'un joker au début like '%chaîne') plutôt que sur la totalité de la chaîne.

3.2 Détermination des techniques d'indexation

Dans une base de données, les attributs des clauses <attribut>=<constante> | <sous-requête>, <attribut> in <liste_de_valeurs> | <sous--requête> donnent lieu à la création d'index de hachage. Les attributs des clauses restantes et les attributs de jointure donnent lieu à la création d'index structurés en b–arbres ou leurs variantes (b^+–arbre, b^*–arbre). La raison est qu'un index de hachage est plus rapide qu'un index en b–arbre pour la recherche d'un enregistrement. Cependant, l'utilisation d'un index de hachage statique sur une table qui ne l'est pas (la fréquence des insertions et des suppressions est importante) peut dégrader substantiellement les performances. Pour pallier ce problème, l'analyse des différentes requêtes du contexte d'extraction peut nous indiquer les tables qui ne sont pas statiques.

Par ailleurs, plusieurs techniques d'indexation sont utilisées dans les entrepôts : index en b–arbre, index *bitmap*, index de jointure et index de projection [VG99]. Dans ce cas, plusieurs facteurs servent à la détermination de la technique d'indexation la mieux adaptée. Ces facteurs sont liés aux caractéristiques des attributs indexés et à leur usage dans une requête SQL.

Les facteurs liés aux caractéristiques des attributs indexés sont :

- la cardinalité d'un attribut (le nombre de valeurs distinctes de cet attribut) ;

– la distribution d'un attribut, qui détermine la fréquence des occurrences des valeurs distinctes de cet attribut ;
– la portée (*value range*) d'un attribut, qui est la différence entre les valeurs maximum et minimum d'un attribut.

Par exemple, pour un attribut de grande cardinalité et de petite portée, une technique d'indexation basée sur un *bitmap* est souhaitable (un b–arbre dégraderait les performance).

De plus, l'emploi d'un index candidat dans une clause d'une requête (jointures, prédicats de restriction, GROUP BY, etc.) peut aider à déterminer le choix de la technique d'indexation à adopter. Par exemple, il est intéressant de créer un index de jointure sur deux attributs fréquemment utilisés ensemble dans une condition de jointure.

3.3 Construction du contexte d'extraction

A partir des attributs extraits dans l'étape précédente, nous construisons une matrice "requêtes-attributs" qui a pour lignes les requêtes de la charge et pour colonnes les attributs à indexer. Cette matrice est le résultat de la combinaison de deux matrices : la matrice M_{select} (construite à partir de $Part_{select}$), dont les lignes représentent seulement les requêtes d'interrogation se trouvant dans la charge, et la matrice M_{update} (construite à partir de $Part_{update}$), dont les lignes représentent les requêtes de mise à jour. La partie $Part_{update}$ est divisée à son tour en deux groupes : G_1 et G_2. G_1 contient les attributs mis à jour et G_2 contient les attributs servant à la mise à jour (par exemple, les attributs de la clause WHERE d'un UPDATE). Par conséquence, la matrice M_{update} est partitionnée verticalement en deux parties : un groupe de colonnes correspondant aux attributs de G_1 et un deuxième groupe correspondant aux attributs de G_2. Ces matrices sont construites de telle façon à garder un lien entre chaque attribut candidat et la requête de la charge où cet attribut est présent. Nous illustrons la construction de ces matrices à travers cet exemple.

Soit une base de données composée de trois tables : `Item (catalog-number`, `name`, `price`, `supplier-code`, `delivery-time`), `Warehouse (warehouse-code`, `warehouse-name`, `warehouse-manager)`, `Item-Warehouse` (`catalog-number`, `warehouse-code`, `quantity`, `emergency-quantity`). La Figure 2 représente un extrait de la charge composé de sept requêtes sur ces tables. Les Figures 3 et 4 montrent les matrices M_{select} et M_{update} respectivement obtenues.

M_{select} peut être diminuée des attributs de $Part_{update}$ qui sont très fréquemment mis à jour et servent rarement à la recherche (par exemple, dans une clause WHERE d'un SELECT ou d'un UPDATE). Un ratio entre la fréquence d'utilisation d'un attribut dans la partie "attributs mis à jour" de la matrice M_{update} et la fréquence du même attribut dans les matrices M_{update} (la partie "attributs de recherche") et M_{select} permet d'élaguer cet attribut. Ce ratio est comparé à un seuil défini par l'administrateur. Il est également possible de prévoir une série de tests pour estimer empiriquement ce seuil.

La matrice "requête–attributs" obtenue par la combinaison des matrices M_{select} et M_{update} représente le contexte d'extraction des connaissances pour recommander un ensemble d'index. Nous pensons que l'utilité d'un index est fortement corrélée avec

```
(1)     Select name, price from Item
        where catalog-number like '999%'
        and price > 1000 order by name
(2)     Select * from Item
        where delivery-time > 100 or price > 100
(3)     Select Item.name from Item, Item-Warehouse
        where Item-Warehouse.catalog-number = Item.catalog-number
        and Item-Warehouse.quantity > 10000
        and Item.supplier-code = 4 and Item.price > 1000
(4)     Select Item.name, Warehouse.name, Item-Warehouse.quantity,
        Item.prace from Item, Warehouse, Item-Warehouse
        where Item.catalog-number = Item-Warehouse.catalog-number
        and Item-Warehouse.warehouse-code =
        Warehouse.warehouse-code
        and Item.price>1000 and Warehouse.Warehouse-code like '%12'
        and Item-Warehouse.quantity<1000
(5)     Update Item  set delivery-time = delivery-time - 1
(6)     Update
        Item-Warehouse set quantity = 50
        where catalog-number = 10
(7)     Delete from Item where delivery-time = 0
```

FIG. 2 – Exemple de requêtes extraites d'une charge

la fréquence de son utilisation dans l'ensemble des requêtes de la matrice "requête–attributs". A priori, la recherche des motifs fréquents ou la classification sont des techniques appropriées pour mettre en évidence cette corrélation et faciliter le choix des index. Nos premiers travaux [ADG03a, ADG03b] se basent sur la recherche des motifs fréquents à l'aide de l'algorithme CLOSE [PBTL99] et apportent des premiers résultats encourageants. La classification peut également être intéressante [Roc03]. L'idée est de regrouper des requêtes similaires. Le but est de ne garder que les index candidats appartenant à un groupe de requêtes ayant une fréquence totale supérieure ou égale à un seuil donné. Ce seuil peut être défini par l'administrateur de la base ou empiriquement.

3.4 Ordre des attributs dans les index multi-attributs

Les index de la configuration peuvent être mono–attributs tels les index de projection ou multi–attributs tels les index de jointure. Le contexte d'extraction peut être exploité pour établir un ordre dans les attributs constituant les index multi–attributs. Un choix judicieux de cet ordre peut réduire le nombre d'index de la configuration. En effet, soit un index i_1 sur les attributs $a_1...a_n$ recommandé pour les requêtes de l'ensemble Q_1. L'index i_1 peut être éliminé s'il existe un autre index i_2 sur les attributs $a_1..a_n a'_1...a'_m$ recommandé pour les requêtes de l'ensemble $Q_2 \supset Q_1$. Si $Q_1 = Q_2$, l'index i_1 est conservé et i_2 est éliminé car il occupe plus d'espace.

Par ailleurs, soient trois index i_1 sur les attributs $a_1...a_n$, i_2 sur $a'_1...a'_m$ et i sur

Tables	Item				
Requêtes	price	name	delevery-time	catalog-number	supplier-code
(1)	1	1	0	0	0
(2)	1	0	1	0	0
(3)	1	0	0	1	1
(4)	1	0	0	1	0

Tables	Item-Warehouse			Warehouse
Requêtes	catalog-number	quantity	warehouse-code	warehouse-code
(1)	0	0	0	0
(2)	0	0	0	0
(3)	1	1	0	0
(4)	1	1	1	1

FIG. 3 – Exemple de matrice M_{select}

$a_1...a_na'_1...a'_m$ (ou i sur les attributs $a'_1...a'_ma_1...a_n$) recommandés pour les requêtes des ensembles Q_1, Q_2 et Q, respectivement. Si $Q \subseteq Q_1 \cup Q_2$, l'index i peut être éliminé.

Pour illustrer ces idées, prenons l'exemple suivant. Soient deux requêtes : `select * from t where a=4 and b=5` et `select * from t where b=3`. Un index `(b,a)` peut être utilisé par ces deux requêtes, alors qu'un index `(a,b)` peut seulement être utilisé par la première requête. Nous pouvons donc nous contenter de créer l'index `(b,a)`. Ajouter une troisième requête `select * from t where a=3` rendrait l'index `(b,a)` inutilisé par cette requête. Dans ce cas, les index `a` et `b` peuvent être utilisés par l'ensemble des trois requêtes.

L'application de ces règles a deux conséquences intéressantes : (1) garantir qu'un index est utilisé par un plus grand nombre de requêtes ; (2) gérer plus efficacement l'espace de stockage. Le deuxième point est particulièrement intéressant dans les entrepôts de données.

3.5 Élagage d'index de la configuration

Le nombre d'index candidats obtenu est d'autant plus important que la charge en entrée est volumineuse. En pratique, la création de tous les index de la configuration peut ne pas être réalisable car le système n'autorise qu'un nombre d'index prédéfini par table. Ce nombre est différent d'un SGBD à l'autre. Par exemple, l'optimiseur d'Oracle autorise jusqu'à seize index par table depuis la version 6. Il faut donc réduire le nombre d'index à générer.

Nous pensons utiliser une des méthodes suivantes. La première consiste à mettre en œuvre un modèle de coût permettant d'estimer le coût d'utilisation d'un index. La deuxième consiste à faire appel à l'optimiseur de requêtes pour estimer ce coût. Dans les deux cas, les index les moins avantageux (ayant un coût plus élevé) dans la configuration sont éliminés. Pour cela, le coût moyen d'utilisation d'un même index dans les différentes requêtes de la charge est calculé. Les index sont ensuite groupés

	Attributs mis à jour				
Tables	Item				
Requêtes	catalog-number	name	price	supplier-code	delivery-time
(5)	0	0	0	0	1
(6)	0	0	0	0	0
(7)	1	1	1	1	1

	Attributs mis à jour	Attributs de recherche	
Tables	Item-Warehouse	Item	Item-Warehouse
Requêtes	quantity	delivery-time	catalog-number
(5)	0	0	0
(6)	1	0	1
(7)	0	1	0

FIG. 4 – Exemple de matrice M_{update}

selon leur table d'appartenance et triés suivant leur coût moyen décroissant. Les k (k est le nombre maximum d'index autorisé) meilleurs index sont conservés et les autres sont éliminés.

Dans les entrepôts de données, la sélection est réalisée sous la contrainte de l'espace de stockage disponible. Si l'utilisation de deux index différents a le même coût, l'index occupant le moins d'espace est privilégié.

4 Conclusion et perspectives

Nous avons proposé dans cet article une méthode basée sur la fouille de données pour la sélection automatique d'index dans les entrepôts de données. Nous complétons en premier lieu nos travaux antérieurs par une analyse plus fine de la charge, une méthode pour la détermination des techniques d'indexation, l'établissement d'un ordre dans les attributs des index multi–attributs et la prise en compte des spécifités du SGBD utilisé. Nous avons également apporté des adaptations pour la sélection d'index dans le contexte des entrepôts de données. Ces adaptations consistent dans le choix des attributs indexables, les facteurs déterminant les techniques d'indexation typiques des entrepôts et la contrainte sur l'espace de stockage disponible.

Actuellement, nous sommes en train de mettre en œuvre ces idées au sein d'un SGBD. D'autre part, nous envisageons dans nos travaux futurs de traiter le problème de la sélection des vues à matérialiser. En effet, la classification effectuée sur le contexte d'extraction permettrait de construire des groupes de requêtes ayant de fortes similarités. Chaque groupe de requêtes peut alors être un point de départ pour la génération des vues matérialisées.

Par ailleurs, l'exploitation de l'ensemble des motifs fréquents pourrait nous permettre de mieux cibler les index de jointure en étoile partiels à construire en fonction de la charge à traiter. Nous éviterions ainsi la constitution d'un seul index de jointure en étoile complet qui serait très coûteux en terme d'espace.

Enfin, il sera indispensable de valider cette approche. Dans un premier temps, nous vérifierons que les performances de notre première approche sont bien améliorées par nos adaptations. Nous comparerons aussi les index que nous proposons avec ceux construits par un expert. Finalement, nous mènerons une étude comparative par rapport aux autres méthodes de sélection d'index, dans la mesure du possible.

Références

[ACN00] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In *26th International Conference on Very Large Data Bases (VLDB 2000), Cairo, Egypt*, pages 496–505, 2000.

[ACN01] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Materialized view and index selection tool for Microsoft SQL Server 2000. *ACM SIGMOD International Conference on Management of Data*, 2001.

[ADG03a] K. Aouiche, J. Darmont, and L. Gruenwald. Frequent itemsets mining for database auto-administration. In *7th International Database Engineering and Application Symposium (IDEAS 2003), Hong Kong, China*, pages 98–103, 2003.

[ADG03b] K. Aouiche, J. Darmont, and L. Gruenwald. Vers l'auto–administration des entrepôts de données. *Revue des Nouvelles Technologies de l'Information*, (1) :1–12, 2003.

[BMT85] R. Bonano, D. Maio, and P. Tiberio. An approximation algorithm for secondary index selection in realtional database physical design. *The Computer Journal*, 28(4) :398–405, 1985.

[BP90] E. Barcucci and O. Pinzani. Optimal selection of secondary indexes. *IEEE Transactions on Software Engineering*, 16(1) :32–38, 1990.

[CBC93a] S. Choenni, H. M. Blanken, and T. Chang. Index selection in relational databases. In *5th International Conference on Computing and Information (ICCI 1993), Ontario, Canada*, pages 491–496, 1993.

[CBC93b] S. Choenni, H. M. Blanken, and T. Chang. On the selection of secondary indices in relational databases. *Data and Knowledge Engineering*, 11(3) :207–238, 1993.

[CN97] S. Chaudhuri and V. R. Narasayya. An efficient cost-driven index selection tool for microsoft SQL server. In *23rd international Conference on Very Large Data Bases (VLDB 1994), Athens, Greece*, pages 146–155, 1997.

[CN98] S. Chaudhuri and V. R. Narasayya. Autoadmin 'what-if' index analysis utility. In *ACM SIGMOD International Conference on Management of Data, Seattle, USA*, pages 367–378, 1998.

[Com78] D. Comer. The difficulty of optimum index selection. *ACM Transactions on Database Systems*, 3(4) :440–445, 1978.

[FON92] M. R. Frank, E. Omiecinski, and S. B. Navathe. Adaptive and automated index selection in RDBMS. In *3rd International Conference on Extending Database Technology (EDBT 1992), Vienna, Austria*, volume 580 of *Lecture Notes in Computer Science*, pages 277–292, 1992.

[FR03] Y. A. Feldman and J. Reouven. A knowledge–based approach for index selection in relational databases. *Expert Systems with Applications*, 25(1) :15–37, 2003.

[FST88] S. J. Finkelstein, M. Schkolnick, and P. Tiberio. Physical database design for relational databases. *ACM Transactions on Database Systems*, 13(1) :91–128, 1988.

[GHRU97] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for OLAP. In *13th International Conference on Data Engineering (ICDE 1997), Birmingham, U.K.*, pages 208–219, 1997.

[GRS02] M. Golfarelli, S. Rizzi, and E. Saltarelli. Index selection for data warehousing. In *4th International Workshop on Design and Management of Data Warehouses (DMDW 2002), Toronto, Canada*, pages 33–42, 2002.

[ISR83] M. Y. L. Ip, L. V. Saxton, and V. V. Raghavan. On the selection of an optimal set of indexes. *IEEE Transactions on Software Engineering*, 9(2) :135–143, 1983.

[KLT03] J. Kratica, I. Ljubić, and D. Tošić. A genetic algorithm for the index selection problem. In *Applications of Evolutionary Computing, Essex, England*, volume 2611 of *LNCS*, pages 281–291, 2003.

[LQA97] W. Labio, D. Quass, and B. Adelberg. Physical database design for data warehouses. In *13th International Conference on Data Engineering (ICDE 1997), Birmingham, U.K.*, pages 277–288, 1997.

[PBTL99] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *7th International Conference on Database Theory (ICDT 1999), Jerusalem, Israel*, volume 1540 of *LNCS*, pages 398–416, 1999.

[Roc03] C. Rochas. A clustering method for database auto–indexing. Mémoire de maîtrise, IUP ISEA, Université Lumière Lyon 2, 2003.

[VG99] S. Vanachayobon and L. Gruenwald. Indexing techniques for data warehouses' queries. Technical report, The University of Oklahoma, School of Computer Science, 1999.

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