La République Algérienne Démocratique Et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Amar Telidji Laghouat
Faculté De Science
Département d'informatique

Mémoire de fin de cycle

En vue de l'obtention du diplôme
Master 2

Option :

Système d'information et de décision

Thème

Adaptation des mesures d'agrégation textuelle au problème de sélection d'index

Présenté par :
M^r BABAOUSMAIL Hassen
M^r TELLAI Mohammed

Soutenu le : 30/06/2015

Devant le jury composé de :

Président :	M^r *XXX* Xxxxxx	M.A.A, Université X. XXXXXXXX
Examinateurs :	D^r *XXXXX* Xxxxxxx	M.C.B, Université X XXXXX Xxxxxxx
Examinateurs :	M^r *XXXXXXX* Xxxxxxxx	M.A.B, Université X XXXXX Xxxxxxx
	M^{elle} *XXXX* Xxxx	XXXXXX, Université XXXXXXXXXX

Année universitaire 2014/2015

Remercienments

Nous tenons à remercier sincèrement Mr. Benamar Ziani, qui, en tant que Dirigeant de mémoire, s'est toujours montré à l'écoute et très disponible tout au long de la réalisation de ce mémoire, ainsi pour l'inspiration, l'aide et le temps qu'il a bien voulu nous consacrer et sans qui ce mémoire n'aurait jamais vu le jour.

Nous souhaitons adresser nos remerciements les plus sincères à Mr. Youcef Ouinten et Mr TAIBAOUI Abd el kader pour leur aide précieuse qui a contribuée à l'élaboration de ce travail.

Nous tenons à remercier aussi les membres de jury d'avoir accepté de juger notre travail.

Nos remerciements s'adressent également à tous les enseignants qui nous ont encadré et conseillé tout le long de nos études, et tout l'encadrement du Département d'informatique.

Dédicaces

Ce présent mémoire est dédié à :
Mes chers parents pour leurs encouragements et leurs prières.
Mes frères et mes soeurs.
Ma fiancée.
A toute ma famille.
Tous Mes amis.
Et à ceux qui m'ont aidé moralement et concrètement, de près ou de loin, pour l'aboutissement de ce modeste travail.

Hassen Babaousmail

Dédicaces

Je dédie ce modeste travail :
Tout d'abord et spécialement à mes chers parents qui m'ont toujours apporté leur soutien, dévouement et leurs encouragements, pour ma réussite.
À Mon frère et mes soeurs.
À toute ma famille.
À tous mes amies.
À tous ceux qui me sont chères.
À tous ceux qui m'aiment.
À tous ceux que j'aime.

Mohammed Tellai

Résumé

La sélection des index pertinents permettant d'optimiser la performance du système en réduisant le temps d'accès aux données est l'une des taches primordiales de l'administrateur. La sélection d'index est un problème NP-Complet car le nombre d'index possible est exponentiel par rapport au nombre d'attributs candidats. Dans ce mémoire nous proposons une méthode de résolution du problème de sélection d'index mono-attribut basée sur la fonction d'agrégation textuel.Nous avons choisi pour valider notre approche de manière expérimentale d'élaborer un outil AFIS qui prend en entrée un ensemble d'attributs indexable issu de la charge de requête du benchmark APB1 (Council, 1998) et en sortie il rend une configuration d'index finale. Les résultats expérimentaux obtenus montrent que notre approche n'est pas intéressante dans le contexte des entrepôts de données.

Mots-cles : Entrepôt de données , base de données , problème NP-Complet , index , Agrégation textuelle , recherche d'information , TF-IDF.

Abstract

The selection of the relevant index in order to optimize system performance by reducing the time of data access is one of the fundamental tasks of the administrator. The index selection is an NP-complete problem because the number of possible index is exponential to the number of candidates attributes. In this memory we propose a method for solving the problem of selection of single- attribute index based on the textual aggregation function .In order to validate our approach experimentally, we have chosen to develop a toolAFIS that takes as input a set of indexable attributes resulting from the query load of the benchmark APB1 (Council, 1998) and as an output, it returns a final index configuration . The experimentally obtained results show thatour approach is not interesting in the context of data warehouses.

Keywords : Data Warehouse , database , NP-Complete problem, index, textual aggregation, information retrieval, TF-IDF.

Table des matières

Liste des tableaux

Table des figures

Introduction générale

L'administrateur a comme rôle principale, la sélection de structures physiques (index, vues matérialisées, etc.) et de politiques (de gestion du cache, de regroupement, de partitionnement, etc.) convenable susceptibles d'améliorer les performances du système en réduisant le temps d'accès aux données [FS88].

Plusieurs structures ont été proposées afin d'optimiser les performances et réduire le temps de réponse des requêtes d'interrogation. Ces structures 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 la fragmentation. Dans la seconde catégorie, nous pouvons citer les index et les vues matérialisées. Comme leur nom l'indique, ces structures exigent un coût de maintenance et un espace de stockage.

Dans le cadre de notre travail nous nous intéressons aux structures redondantes, à savoir les index. Un index est une structure physique permettant d'accélérer l'interrogation de la base de données en créant des chemins d'accès direct.

La sélection d'index est un problème NP-complet [Com78] car le nombre d'index possibles est exponentiel par rapport au nombre total d'attributs candidats à la procédure d'indexation. De ce fait, il n'existe pas d'algorithme qui donne une solution optimale en un temps raisonnable.

D'autre part, les techniques récentes du domaine de recherche d'information (RI) permettent de rechercher les termes les plus significatifs pour un ensemble de document. En s'inspirant de cette remarque, nous voulons adapter la fonction de pondération TF-IDF() du domaine de recherche d'information afin de résoudre le problème de sélection d'index.

Le mémoire est organisé comme suit :
Le premier chapitre est consacré a l'indexation dans les bases et entrepôts de données . On définit le problème de sélection d'index ainsi on presente l'état de l'art sur les principaux traveaux realisée dans ce domaine.
Le deuxième chapitre présente les notions de base de domaine de recherche d'information (RI), ainsi, la description de notre approche et sa relation avec le domaine RI.
Dans le troisième chapitre, une étude expérimentale qui montre les résultats obtenus ainsi qu'une comparaison avec l'approche des motifs fréquents maximaux [ZB10] enutilisant un modèle de coût,Les tests sont réalisés sur un banc d'essais (APB-1) [Cou98].
Nous concluons ce mémoire en rappelant la problématique étudiée ainsi que les résultats obtenus et en proposant quelques perspectives.

Chapitre 1

Techniques d'indexation dans les bases et entrepôts de données relationnels

1.1 Sélection d'index

Dans les systèmes de gestion de bases de données (SGBD), lorsque les bases de données sont volumineuses l'accès aux données devient lent. L'opération qui consiste à parcourir séquentiellement les données est une opération lente et pénalisante pour l'exécution des requêtes, particulièrement dans le cas des requêtes de jointure où ce parcours doit souvent être effectué de façon répétitive [K05].

La création d'un index permet de réduire le temps d'accès aux données en créant des chemins d'accès directs [K05].
Il existe deux types d'index :

1. les index primaires, aussi appelés index groupants.
2. les index secondaires, aussi appelés index non-groupants.

Les adresses contenues dans un index primaire sont triées suivant le placement physique sur disque des n-uplets composant la table indexée. En revanche, les adresses d'un index secondaire ne suivent pas cette organisation. Lorsqu'un index primaire est utilisé, peu de blocs disques sont parcourus et les requêtes de recherche sont ainsi résolues de manière efficace. Toutefois, ce type d'index souffre d'un coût de maintenance très élevé car il faut maintenir l'ordre du tri. Les index secondaires sont moins efficaces que les index primaires, mais moins coûteux au niveau de la maintenance. Il peut y avoir au plus un index primaire, mais plusieurs index secondaires sur une table donnée sont possibles.

Indexable Attributes : c'est l'ensemble de tous les attributs qui peuvent réduire le temps de réponses des requêtes, ils se trouvent dans la clause $WHERE$ des requêtes SQL.

Nous présentons dans les sections suivantes les principales techniques d'indexation utilisées dans les SGBD relationnels et les entrepôts de données.

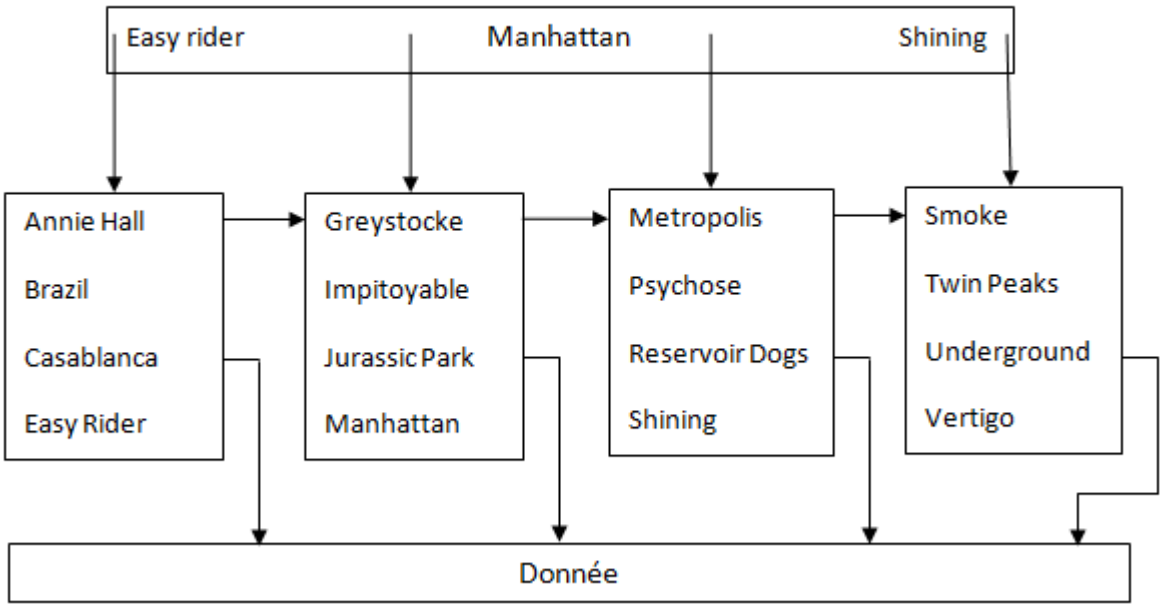

FIGURE 1.1 – Index en B-arbre construit sur l'attribut Film_Titre

1.1.1 index B-arbre

Un B-arbre est une liste chaînée de nœuds dont la valeur est celle de l'index. Si cet index est construit sur un attribut clé, les feuilles de l'arbre font référence à une seule valeur. Dans le cas où l'index est construit sur un attribut non-clé des n-uplets de la table indexée, les feuilles de l'arbre font référence à plusieurs valeurs. Cette référence spécifie l'emplacement physique du n-uplet sur le disque [NEC13].

Le B-arbre est une structure d'index qui offre un excellent compromis pour les opérations de recherche par clé et par intervalle car sa mise à jour est dynamique. Ces qualités expliquent qu'il soit ainsi que ces variantes systématiquement intégrées dans la plupart des SGBD relationnelles [NEC13] [K05].

La Figure 1.1 montre un exemple de B-arbre construit sur la table Films définie par le schéma $Films(Film\_ID, Film\_Titre, Film\_Année, ...)$.

1.1.2 Les index de jointure binaires

Dans les entrepôts de données modélisés par un schéma en étoile, l'IJB sert à précalculer les jointures entre une ou plusieurs tables de dimension et la table des faits [OP95] ; [OP97].

Un index de jointure binaire (IJB) peut être défini sur une table des faits en utilisant un seul attribut (dans ce cas, appelé un IJB mono-attribut) ou plusieurs attributs appartenant

à une ou plusieurs tables de dimension (appelé, IJB multi-attributs).Ainsi le nombre d'IJB candidat peut être très important ce qui rend la tâche de sélection très compliquée.

La nature binaire des IJB permet d'améliorer les performances des requêtes en permettant d'appliquer des opérations logiques AND, OR, NOT,... etc. Ces opérations permettent de rechercher des n-uplets vérifiant des conjonctions ou des disjonctions de prédicats. Les IJB sont très bénéfiques pour les requêtes de type $Count(*)$ où l'accès à l'index binaire seulement permet de répondre à ces requêtes [Bk10].
Une autre caractéristique offerte par les index de jointure binaires est la compression [J99], d'où une réduction de leur espace de stockage et la possibilité de les stocker en mémoire centrale.

Supposons un attribut A ayant n valeurs distinctes $v_1, v_2, ..., v_n$ appartenant à une table de dimension D. Supposons que la table des faits F est composée de m instances. Pour construire un index de jointure binaire définis sur l'attribut A, on suit les étapes suivantes [J99] :

1. Créer n vecteurs composés chacun de m entrées ;
2. Le i^{eme} bit du vecteur correspondant à une valeur v_k est mis à 1 si le n-uplet de rang i de la table des faits est joint avec un n-uplet de la table de dimension D tel que la valeur de A de ce n-uplet est égale à v_k. Il est mis à 0 dans le cas contraire.

1.2 Problème de sélection d'index

Le problème de sélection d'un ensemble d'index optimal pour une base de données a été étudié depuis les années 70 [NEC13].Il consiste à construire une configuration d'index optimisant le coût d'exécution d'une charge donnée. Cette optimisation peut être réalisée sous certaines contraintes, comme l'espace de stockage alloué aux index à sélectionner.

Formellement : Etant donnée :

1. $W = \{q_1^{f_1}, ..., q_n^{f_n}\}$ un ensemble de requêtes les plus frÃĺquentes avec leur frÃĺquences respectivement.
2. S la taille de l'espace maximum du disque alloué par l'administrateur pour stocker les index à sélectionner.

alors il faut trouver une configuration d'index CI tel que [ZB10] :

1. le coût d'exécution des requêtes de la charge soit minimal ;

$$C = armin_i \sum Cout(q_i) \times f_i$$

2. l'espace de stockage des index de CI ne dépasse pas S ;

$$(Taille(CI) \leq S)$$

Client

ID	CID	Nom	Age	Ville
1	700	Ahmed	20	Alger
2	515	Omar	42	Laghouat
3	400	Med	21	Ghardaia
4	313	Hassen	52	Tebessa
5	200	Mounir	18	Blida
6	111	Khaled	17	Oran

Temps

ID	TID	Mois	Année
1	11	Janvier	2003
2	12	Février	2003
3	33	Mars	2003
4	44	Avril	2003
5	55	Mai	2003
6	66	Juin	2003

Produit

ID	PID	Nom	Gamme
1	101	Sonoflore	Beauté
2	102	Clarin	Beauté
3	103	WebCam	Multimedia
4	104	Barbie	Jouets
5	105	Manure	Gardering
6	106	SlimForme	Sport

Vent

ID	CID	PID	TID	MONTANT
1	700	106	11	25
2	700	106	66	28
3	700	104	33	50
4	515	104	11	10
5	400	105	66	14
6	200	106	55	14
7	111	101	44	20
8	111	101	33	27
9	200	101	11	100
10	313	102	11	200
11	400	102	11	102
12	400	102	55	103
13	515	102	66	100
14	515	103	55	17
15	212	103	44	45

IJB sur Ville

ID	Alger	Laghouat	Ghardaia
1	1	0	0
2	1	0	0
3	1	0	0
4	0	1	0
5	0	0	1
6	1	0	0
7	1	0	0
8	1	0	0
9	1	0	0
10	0	0	1
11	0	0	1
12	0	0	1
13	0	1	0
14	0	1	0
15	1	0	0

FIGURE 1.2 – Exemple d'index de jointure binaire

Complexité La sélection d'index est un problème NP-Complet [Com78] étant donné la grande volumétrie des bases et entrepôts de données relationnels et la complexité des requêtes d'interrogation.
Par conséquent, l'espace de recherche qui se rapporte à une charge de requêtes peut être très grand étant donné le nombre important des attributs candidats participant à la construction des index [GVA00].

Soit $A = \{A_1, A_2, ..., A_K\}$ un ensemble d'attributs indexables candidats pour une configuration d'index. Chaque index dans cette configuration est constitué d'un sous ensemble d'attributs de A, Ainsi cette configuration constitue une partition de A en un ensemble de groupes. Chaque groupe d'attributs représente un index potentiel [BK08]. Nous pouvons considérer deux cas :

- index unique (un seul index) : le nombre de possibilité est donné par :

$$N = 2^K - 1$$

  Si le nombre d'attributs indexables est égal à 5 ($K = 5$), alors le nombre d'index possible est $N = 31$.
- Une configuration d'index (un ou plusieurs index) : le nombre de possibilités est donné par :

$$N = 2^{2^K - 1} - 1$$

Si le nombre d'attributs indexables est égal à 5 ($K = 5$), alors le nombre d'index possible est $N = 2^{31} - 1 > 1.2 \times 10^9$.

1.3 Approches de sélection d'index

Plusieurs approches ont été proposées dans la littérature pour approcher une solution optimale, ils se distinguent principalement selon les caractéristiques suivantes [Dar06] :

1. la méthode de sélection de l'ensemble d'index candidats ,
2. la méthode de sélection d'une configuration finale d'index ,
3. les modéles de coûts utilisés pour cette seconde sélection.

La plupart de ces approches utilisent deux phases [NEC13] :

1. la sélection des index candidats.
2. la sélection d'une configuration finale d'index.

1.3.1 Phase de sélection d'index candidats

Cette phase permet d'identifier les attributs indexable et de réduire l'espace de recherche du problème de sélection d'index en éliminant les attributs non pertinents : après une analyse de la charge de requêtes, la détermination des attributs indexables peut se faire selon deux méthodes : manuelle ou automatique.

Les premières approches de sélection d'index qui ont été proposées dans la littérature font appel à l'expertise humaine pour constituer manuellement l'ensemble d'index candidats [KW85], [RF92], [SC93], [Gun99], [JK03], mais la tendance dans les travaux plus récents est de recourir à une approche automatique, plus adéquate pour être déployée à grande échelle [Dar06] . Pour cela, une analyse syntaxique de la charge de requêtes est réalisée pour la génération d'un ensemble d'attributs candidats. [SC97], [GV00], [MG02], [YF03b].

1.3.2 Phase de sélection d'index finaux

Dans cette phase , le but est de construire la configuration finale d'index qui va reduire le coût d'execution des requêtes en respectant l'espace de stockage disponible pour ce faire un modèle de coût est utilisé.

1.4 Travaux sur la sélection d'index

Le problème de sélection d'index est NP-Complet [Com78]. De ce fait, il n'existe pas d'algorithme qui propose 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.

Nous présentons dans ce qui suit un état d'art des travaux proposés pour résoudre le problème de sélection d'index :

1.4.1 Travaux de sélection d'index dans les bases de données

Travaux de Chaudhuri et al [A13] : Dans le but d'auto administration des bases de données ; Microsoft a lancé un projet de recherche pour la mise en œuvre d'un outil d'aide à la résolution du problème de la conception physique afin d'améliorer les performances des bases de données notamment la sélection d'index qui est gérée manuellement par l'administrateur dans ce contexte Chaudhiri et al ont développé l'outil de sélection d'index IST (Index Selection Tool) sous Microsoft SQL Server 7.0 qui se déroule en trois étapes pour la sélection d'une configuration comme suit : (Figure 1.3) [CS08]

1. Sélection des index candidats : Dans cette étape ; une analyse de la charge de requête pour détecter les d'attributs existant dans les clauses WHERE,GROUP BY et ORDER BY afin de construire l'ensemble des attributs indexables.
2. Réduction de l'ensemble d'index : L'élimination d'un certain nombre d'index qui ont un gain de performance faible .à l'aide d'un algorithme glouton et a partir de n index candidats construits les k meilleurs sont sélectionnés. Ce module est en échange permanent avec l'optimiseur de requêtes.
3. Génération d'index multi-attributs : dans cette étape l'IST utilise les index obtenus dans l'étape précédente et qui sont mono-attribut et Pour construire des index multi-attributs, deux fonctions sont utilisées : MCLEAD et MC-ALL. La fonction MC-LEAD permet de générer un index multi-attribut en combinant un index mono-attribut avec un attribut indexable (n'est encore indexé). La fonction MC-ALL permet de générer un index multi-attributs en combinant deux attributs mono-attributs. La génération des index multi-attributs de taille supérieure se fait selon le même principe.

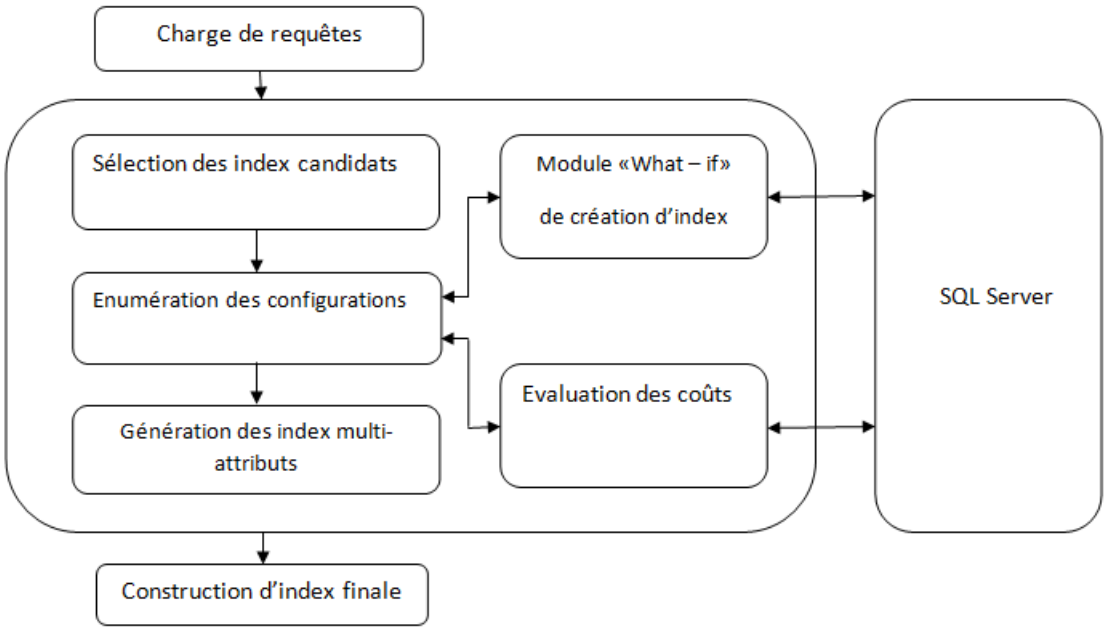

FIGURE 1.3 – Architecture de l'outil de sélection d'index

Travaux de Valentin et al [A13] : Dans [ZG00] les auteurs proposent une solution pour la sélection d'index sous le système DB2 Advisor d'IBM ; la sélection d'index est établie par un algorithme qui est une extension de l'optimiseur qui génère pour chaque requête le plan d'exécution optimal. Les index utilisés dans ces plans seront recommandés et une fois l'ensemble d'index définit, le problème est modélise comme le problème du sac à dos pour être résolu par heuristiques.

Travaux de Feldman et al [A13] : Les auteurs proposent un outil d'assistance à base de connaissance pour la sélection d'index nommé DINNER [YF03a]. A partir des statistiques sur l'ensemble des tables et les différentes requêtes sur ces tables l'outil constitue un modèle de coût, ensuite pour l'ensemble des requêtes, DINNER construit un graphe représentant l'ensemble des solutions possibles qui peut utiliser chaque requête ; Les requêtes de jointures sont décomposées en plusieurs requêtes définies sur une seule table. à partir des meilleurs graphes, on obtient une configuration d'index, avec leurs coûts et leurs espaces de stockage. l'outil utilise des heuristiques pour trouver, dans cette configuration d'index candidats, les meilleurs index.

Travaux de kratica et al [A13] : Dans cette approche Kratica et al. proposent un algorithme génétique pour résoudre le problème de sélection d'index [TD03] ; ils appliquent l'algorithme sur une population d'individus N_{pop}. N_{elit} est le nombre d'individus élus pour survivre à la génération qui suit .La fonction objective à évaluer représente le temps de traitement des requêtes. Elle est stockée dans une table cache de taille Ncache pour accélérer le temps des calculs .

Travaux de Frank et al [K05] [A13] : Les travaux de Frank et al. , menés à l'Université de Georgia Tech, ont pour but de proposer un outil d'aide à la décision pour le choix d'index dans une base de données [RFN92].

Cet outil utilise une configuration initiale d'index et une charge de requêtes. Un dialogue est établi entre l'outil de sélection d'index et l'optimiseur de requêtes du SGBD afin de calculer le gain en performance qu'apporte l'utilisation d'un index pour la charge. Le gain est défini comme la différence entre le coût d'exécution des requêtes sans index et celui de leur exécution avec index.

L'estimation du coût d'un index est réalisée en faisant appel à l'optimiseur de requêtes. Nous résumons cette méthode par les cinq points suivants :

1. Une requête de la charge est soumise à l'optimiseur de requêtes avec un ensemble d'index initial.
2. L'ensemble des index utilisés pour la requête courante est stocké avec le gain de performance pour la requête.
3. De nouveaux ensembles d'index sont générés et l'étape 2 est réitérée jusqu'à effectuer un parcours séquentiel.
4. Les gains en performance de chaque index sont additionnés.
5. Les index présentant un gain total positif sont enfin proposés à l'utilisateur.

Travaux de Whang - Algorithmes ADD and DROP [NEC13] : Whang et al, [Wha87]. ont proposé une approche ascendante et une autre descendante pour la sélection d'index. L'approche ascendante qui est implémentée par l'algorithme de sélection DROP, commence d'abord en considérant tous les index possibles.

Durant chaque itération, l'index engendrant la plus grande décroissance du coût des requêtes est éliminé. Quand cette réduction du coût n'est plus possible en éliminant un seul index, l'algorithme DROP élimine deux index à la fois, trois index à la fois et ainsi de suite, jusqu'à ce que ceci ne soit plus possible.

L'approche ascendante qui est implémentée par l'algorithme de sélection ADD, initialise le processus d'optimisation par l'ensemble vide (aucun index n'est encore sélectionné). A chaque itération, l'algorithme ADD ajoute un index réduisant le coût d'exécution des requêtes. Le processus s'arrête lorsqu'aucune réduction de coût n'est possible.

Travaux de Brunel, Rollin et al [NEC13] : Ces travaux ont été effectués dans le cadre d'une collaboration entre l'université d'Oklahoma et l'université Lyon 2. Deux outils IUS (Index Usage Statistics) et ISCA (Index Selection and Creation Algorithm) ont été réalisés pour la sélection automatique des index [PB01] :

1. L'outil IUS (Index Usage Statistics) :
   IUS qui est inspiré des travaux de Frank et al., [RFN92]. dialogue avec l'optimiseur de requêtes mais évite la construction du graphe d'index et le calcul des gains de performances.

   Cet outil facilite le choix des index à construire en établissant un ensemble de statistiques représentant l'ordre d'importance de chaque index de l'ensemble initial pour chacune des requêtes de la charge. Ces statistiques sont obtenues, à partir d'un ensemble d'index, par l'optimiseur qui effectue un travail itératif couvrant toutes les requêtes de la charge.

   A chaque itération et tant qu'il y a des index utiles, l'optimiseur propose pour la requête courante le meilleur index (moins coûteux) à utiliser et l'outil lui affecte un rang. Par la suite, on enregistre les index et leurs rangs trouvés pour la requête traitée et on réinitialise l'ensemble d'index à l'ensemble initial.

   A la n, on obtient un ensemble de statistiques représentant l'ordre d'importance de chaque index de l'ensemble initial dans chacune des requêtes de la charge. Cela facilite à l'administrateur le choix des index à construire.
2. L'outil ISCA (Index Selection and Creation Algorithm) :
   A partir d'une charge représentant uniquement l'ensemble des requêtes de sélection obtenues à partir du chier log. ISCA extrait les attributs présents dans chaque requête et leur affecte à chacun un identificateur qui est un numéro binaire établi dans sa table d'appartenance.

   ISCA calcule pour chaque attribut de la table, le nombre de fois (fréquence) où il a été utilisé dans l'ensemble des requêtes de la charge et aussi le nombre de fois où il a été utilisé avec un ou plusieurs autres attributs extraits. Ensuite, l'administrateur de la base de données fixe un seuil et ne conserve que les attributs dépassant ce seuil.

Travaux de Dogac et al [NEC13] : MAESTRO7 (Metu Automated indEx Selection Tool foR Oracle7) [AD94]. est un outil de sélection automatique d'index spécifique pour ORACLE 7. Il a été proposé par le centre de recherche et développement de logiciel TUBITAK de l'université technique à Ankara, Turquie. Il permet de proposer, parmi un ensemble complet d'index, des index primaires (clustering index) et secondaires (non-clustering index) en tenant compte du coût de maintenance.

MAESTRO7 commence par extraire automatiquement, à partir du chier Log, les requêtes SQL et leurs statistiques pour construire la charge de requête. Ensuite, il classier les requêtes produisant le même plan d'exécution et met à jour leurs poids en les cumulant. Il détermine ensuite les colonnes susceptibles d'être des index dans le but d'améliorer les performances. Pour cela, il utilise la logique du SGBD ORACLE7 dans l'utilisation ou non des index. Ainsi aucune colonne présentant ces caractéristiques n'est sujette à être candidate :

- Si une colonne est modifiée dans une clause WHERE
- Si une colonne est utilisée avec : NOT IN, NOT LIKE, IS NOT NULL, IS NULL
- Si elle est utilisée avec la clause LIKE pour comparer une chaine de caractère ayant le '%' au début.
- Les colonnes d'une même table apparaissant dans les deux cotes d'un opérateur de comparaison, car un parcours séquentiel doit être effectué dans cette situation.
- Si une requête contient une sous-requête, il est nécessaire d'exécuter la sous-requête pour chaque ligne de la requête. Donc l'index sur la colonne de la requête qui est compare au résultat de la sous-requête n'est pas utile.

Par la suite, il considère les colonnes utilisées avec l'une des clauses suivantes comme candidates pour les index de hachage :
$< nom\_colonne >=< constante >< sous - requête >$
$< nom\_colonne > LIKE < constante >$
$< nom\_colonne > IN < liste\_de\_valeurs >< sous - requête >$.
Cependant, étant donné qu'ORACLE7 utilise un index de hachage statique qui n'est performant que s'il est utilisé sur une table statique. L'outil supprime, grâce aux informations déjà collectées, toutes les colonnes candidates pour l'index de hachage dont la table correspondante est considérée comme non statique. Une table est considérée comme non statique si la différence entre le nombre de lignes insérées et les lignes supprimées est supérieure au nombre de ligne total par un certain pourcentage. Le reste des colonnes indexables sont candidates pour l'index B +Tree.

Travaux de Choenni et al [NEC13] : Dans le contexte des bases de données relationnelles, Choenni et al., [SCC93]. utilisent un modèle mathématique pour réduire l'ensemble des candidats pour le problème de sélection d'index.

Il se base sur le fait que l'ajout d'un index à une configuration peut augmenter son coût, le réduire, comme il peut aussi ne pas le changer. Les propriétés suivantes permettent de minimiser l'espace de recherche à explorer an d'éviter d'effectuer une recherche exhaustive.

Les deux premières propriétés sont spécifiques à une fonction mathématiques supermodulaire et les deux dernières sont spécifiques à une fonction sous-modulaire. Il est à noter que les propriétés 1 et 4 sont équivalentes et les propriétés 2 et 3 sont équivalentes [K05].

Proposition 1 : supposant un index i à ajouter à un ensemble d'index I. Si la valeur de

C ne décroit pas suite à cet ajout, alors elle ne décroit pas non plus en ajoutant l'index i à tout ensemble I' contenant I. Cela signifie que l'index i n'appartient pas à la configuration d'index optimale.

Proposition 2 : supposant que l'index i soit à éliminer de l'ensemble I. Si la valeur de C n'est pas réduite suite à cette élimination, la valeur de C ne peut pas l'être pour tout index i éliminé de l'ensemble I' contenu dans I. Cela signifie que l'index i appartient à la configuration d'index optimale.

Proposition 3 : supposant que l'index i est avantageux pour un ensemble d'index I, il l'est aussi pour tout ensemble d'index I' contenant I. Cela signifie que i fait partie de la configuration optimale.

Proposition 4 : si un index i réduit le coût d'un ensemble d'index I. Suite à son élimination de cet ensemble, ce coût est aussi réduit pour tout ensemble I' contenu dans I. Cela signifie que i ne fait pas partie de la configuration optimale.

L'algorithme de sélection d'index prend en entrée une fonction de coût C, un ensemble d'attributs à indexer selon l'administrateur du système et une configuration initiale d'index pertinente pour chaque requête de la charge. Il considère aussi un sous-ensemble de Q notés Q_red. Q_red étant l'ensemble de requêtes de la charge Q pour lesquelles aucun index parmi la configuration initiale n'est avantageux.

L'algorithme renvoie, pour chaque sous-ensemble Q_red, un ensemble d'index avantageux constituant la configuration d'index optimale et un autre ensemble d'index désavantageux n'appartenant pas à la configuration d'index optimale.

L'évaluation des coûts des requêtes en présence de ces index mène généralement à décomposer chaque sous-ensemble de Q_red en deux groupes. Un groupe G1 contenant les requêtes rendant la fonction de coût C super-modulaire et un groupe G2 contenant celles rendant la fonction de coût C sous-modulaire. A chaque groupe sont appliqués les propositions adéquates an d'optimiser l'espace de recherche à explorer.

Enfin, une fusion des ensembles d'index avantageux et des ensembles d'index désavantageux est nécessaire an de n'avoir qu'un seul ensemble d'index avantageux et un seul ensemble d'index désavantageux pour pouvoir résoudre le problème de sélection d'index avec une recherche exhaustive. Le temps et la faisabilité de cette recherche dépend du nombre d'index à engendrer.

Travaux de Gundem [NEC13] : La méthode présentée par Gundem [Gun93]., utilise un ensemble d'index candidats fourni par l'administrateur pour y sélectionner un sous-ensemble qui minimise, suivant une erreur tolérée, le coût de traitement des opérations de mise à jour et de sélection sans violer la contrainte d'espace de stockage.

Cette méthode prend en compte le fait qu'un attribut donné peut être indexé suivant plusieurs techniques et que, par conséquent, les index construits sur cet attribut suivant différentes techniques apportent des gains et occupent des espaces de stockage différents. En effet, un index candidat peut être un B-arbre, un index d'ordre, un ensemble d'index partiels etc. Ainsi, pour les attributs correspondants aux tables de la base considérée, l'ensemble des

index associés à un attribut donné est appelé une classe d'équivalence et l'ensemble de ces classes constitue une partition. Le processus de sélection est subdivisé en deux étapes.

La première étape consiste à effectuer une optimisation locale pour calculer un ensemble disjoint d'index candidats qui maximisent le gain.

En premier, on déni un seul index candidat pour chaque attribut (un index par classe d'équivalence). Par la suite, on calcule, grâce à une fonction de coût. Le gain pour chaque index candidat est calculé en évaluant son coût de construction et l'apport obtenu par son utilisation pour les opérations de sélection et de mise à jour. Le résultat de cette étape est donc un ensemble disjoint d'index possédant un gain non négatif.

La deuxième étape consiste à réaliser une optimisation globale. Ceci consiste à sélectionner parmi l'ensemble des candidats obtenus à l'étape précédente, ceux qui minimisent une fonction de coût total tous en respectant la contrainte d'espace de stockage requis pour les index. La fonction de coût total est calculée comme la différence entre le coût de traitement des opérations de la charge sans optimisation et l'apport obtenu par l'utilisation de tous les index candidats.

Travaux de Finkelstein et al [NEC13] : DBDSGN est un outil expérimental de conception physique des bases de données relationnelles développé par Finkelstein et al. [SF82] dans le laboratoire de recherche d'IBM à San Jose aux états unis.

Compte tenu d'une charge de requête pour le système R [DC81]., constitué d'un ensemble d'instructions SQL et leurs fréquences d'exécution, DBDSGN suggère des configurations pour des performances optimales.

Chaque configuration se compose d'un ensemble d'index et un ordre pour chaque table. Les états de la charge de travail sont évalués uniquement pour les configurations qui ont un seul index par table.

DBDSGN utilise les informations fournies par l'optimiseur du système R [DC81]. à la fois pour déterminer quelle colonne pourrait être indexable et obtenir aussi des estimations du coût concernant les états d'exécution des différentes configurations.

Finkelstein et al., [SF82]. considèrent que si les estimations des coûts fournies par l'optimiseur sont les coûts réels d'exécution, l'outil trouvera la solution optimale.

Les principes de DBDSGN ont été utilisés dans l'outil RDT (relationnel Design Tool), un produit commercial d'IBM, qui effectue la conception pour SQL/DS, un système relationnel basé sur le système R.

Dans ce contexte, un dialogue entre l'outil et le DBMS est établi en utilisant une nouvelle instruction SQL appelée SQL EXPLAIN pour les estimations des coûts et autres renseignements utiles peuvent être obtenues à partir du système R an qu'un autre système les utilises.

Elle permet aussi de sauvegarder le plan d'exécution (incluant le chemin d'accès), pour une requête donnée, choisi par l'optimiseur de requête dans des tables dite d'explication.

1.4.2 Travaux de sélection d'index dans les entrepôt de données

Travaux de Golfarelli et al [A13] : L'algorithme de Golfarelli et al de sélection [SS02] d'index est basé sur un modèle de coût mathématique dont les étapes sont comme suit :

- Initialisation de l'ensemble des index candidats C et l'ensemble des index optimaux O ainsi que l'espace de stockage S nécessaire pour stocker les index à sélectionner.
- On utilise algorithme glouton pour la sélection, à partir de l'ensemble des index candidats C , les index apportant un meilleur coût en les ajoutant dans l'ensemble O, Si après un ajout il arrive que tous les attributs composant la clé primaire d'une table de faits soient indexés, ces index sont transformés en un seul index multi-attributs construit sur la clé primaire de la table de fait . multi-attributs construit sur la clé primaire de la table de fait [SS02].

Travaux de Aouiche et al [A13] : La recherche des motifs fréquents fermés où chaque motif est un index de jointure binaire candidat est la base de L'approche d'Aouiche et Al [K05], les auteurs ont choisi d'utiliser l'algorithme CLOSE pour la recherche des motifs fréquents fermésà partir d'une charge de requêtes donnée afind'élaguer l'espace de recherche des index de jointure.

Pour arriver à une configuration d'index finale l'approche d'Aouiche et al passe par les étapes suivantes :

- Extraction des requêtes : Extraire la charge de requêtes a partir du journal des transaction de l'entrepôt de données.
- Extraction des attributs indexable : extraction des attributs susceptible d'être des supports d'index par un analyseur syntaxique.
- Construction d'un contexte d'extraction : Une matrice 'Requêtes-attributs' de taille $n*m$ est construite où M est le nombre de requêtes et N le nombre des attributs d'index candidats T, l'existence d'un attribut indexable dans une requête est symbolisée par 1 et son absence par 0.

Exemple :

Q1 : T1 T2 T3 T4
Q2 : T2 T5
Q3 : T1 T2 T4T5
Q4 : T2 T5
Q5 : T1T2 T4
Q6 : T2 T5

La matrice dans ce cas est :

	T1	T2	T3	T4	T5
Q1	1	1	1	1	0
Q2	0	1	0	0	1
Q3	1	1	0	1	1
Q4	0	1	0	0	1
Q5	1	1	0	1	0
Q6	0	1	0	0	1

TABLE 1.1: Matrice Requêtes-attributs

- Génération de l'ensemble d'index candidats : L'algorithme CLOSE est appliqué sur le contexte d'extraction pour obtenir les motifsfréquents. Chaque motif extrait est composé d'un ensemble d'attributs de l'entrepôt de données.
  Exemple :
  A partir de table1 le motif fréquent est : T2.T5
- Construction de la configuration finale d'index : A partir de l'ensemble d'index généré dans l'étape précédente, un algorithme glouton est appliqué pour sélectionner une configuration d'index finale.

Travaux de Bellatrache et al [A13] : Bellatreche et al. proposent d'ajouter d'autres paramètres comme la taille des tables de dimension,la taille de la page système, etc. et leur approche se base sur la technique de motifs fréquent fermé.vu que 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. les auteurs ont montré à travers un exemple que la fréquence d'accès seule ne permet pas de sélectionner un ensemble d'index efficace. Ils proposent donc les algorithmes DynaClose et DynaCharm qui prennent en compte ces paramètres [MR07]. Ces algorithmes utilisent une nouvelle métrique appelée Fitness qui utilise, outre la fréquence, les tailles des tables ; pour générer les attributs candidats ; cette approche utilise une méthode de Datamining pour réduire l'espace de recherche après avoir récupérer les attributs indexable et pour sélectionner une configuration d'index finale un algorithme glouton est utilisé , il se base sur un modèle de coût pour définir la fonction objective afin d'évaluer le temps d'exécution de la charge de

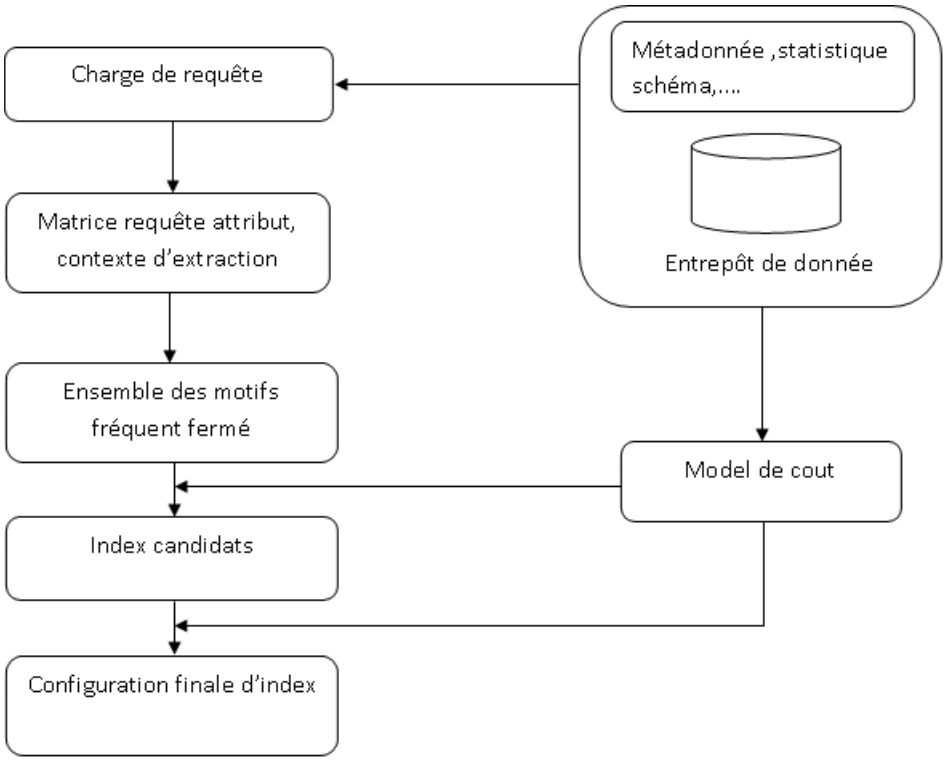

FIGURE 1.4 – Architecture de fonctionnement de l'approche de Aouiche et al

requête et le coût de stockage de l'index.afin de pénaliser les attributs fréquents définis sur des tables de petites tailles.

L'approche de Ziani et al [A13] : L'approche de Ziani et al. consiste a trouver une configuration d'index CI pour minimiser le coût d'exécution de requête Q on respectant la contrainte de stockage S, L'objectif de cette approche est basé sur la recherche des motifs fréquent maximum.Ces derniers à la particularité que tous ses sur-ensembles sont non fréquents. Cette particularité va permettre la génération des index moins nombreux et plus pertinents.

L'approche de Ziani et al est constituée des étapes suivantes :

- Extraction de la charge de requête : sélection d'une charge de requêtes, supposée représentative, de l'activité du système .
- Construction du contexte d'extraction : structuration des attributs indéxables contenus dans la charge sous forme d'une base transactionnelle où les requêtes représentent les transactions et les attributs représentent les motifs.
- Construction de l'ensemble d'index candidats : génération des index candidats par l'algorithme $fpmax$ implémenté en java .

Chapitre 2

Adaptation de la fonction TF_IDF à la resolusion du problème de sélection d'index

2.1 Introduction

La recherche d'information (RI) est apparue comme une discipline de recherche dans le but de résoudre les problèmes liée à l'accès aux informations contenues dans des grandes masses de documents. Depuis que internet a fait son entrée, on a assistée a une croissance importante en terme de nombre de ressources d'informations facilement accessible. Pour bien organiser et faciliter l'accès aux informations contenues dans ces documents dont le nombre ne cesse de croître, le RI avait proposée des techniques dont les premiers mécanismes proposées étaient : la classification des documents par catégories et la création des indexes.
Dans ce chapitre, nous allons voir quelques définition et concepts de base sur le domaine de recherche d'information, puis on va présenter notre démarche dans laquelle on utilise une technique de RI afin de résoudre le problème de sélection d'index mono-attribut.

2.2 Notion de base de la recherche d'information

2.2.1 Concepts clés

Définition de la recherche d'information

D'après l'Association Française de Normalisation (AFNOR) [Che11] : La RI est définis comme : Action, méthodes et procédures ayant pour objet d'extraire d'un ensemble de document les informations voulues. Dans un sens plus large, toute opération (ou ensemble d'opérations) ayant pour objet la recherche, la collecte et l'exploitation des informations en réponse a un sujet précis.

Document

Selon l'Organisation Internationale de Normalisation (ISO) : Le document est un ensemble formé par un support et une information, généralement enregistré de façon permanente et tel qu'il puisse être lu par l'homme et la machine.

Collection de documents

Collection de documents (ou fond documentaire, corpus) est un ensemble d'informations exploitable et accessible par un utilisateur, ou simplement, c'est l'ensemble de documents dans lequel un utilisateur cherche une information [N14].

Le système de recherche d'information (SRI)

Selon [N14] : C'est un système informatique qui sert d'interface entre une collection de document et des utilisateurs.il leur permet de retrouver les documents dont le contenu correspond le mieux à leur besoin d'information.

Indexation

L'indexation est une étape primordiale dans la recherche d'information. C'est un processus qui permet d'extraire d'un document une représentation parametré qui couvre au mieux son contenu sémantique. Le résultat de l'indexation est une liste de mots clés appelée descripteurs [N14]. Mathématiquement, un index est une fonction qui relie chaque documentD_i a l'ensemble des mots clés du corpus T. [Baa05] :

$$\text{Index} : D_i \overset{traite}{\longmapsto} t_j \text{ ou } tj \in 2^T$$

Pondération des termes

La pondération est une fonction fondamentale en RI, elle permet de définir l'importance qu'a un terme dans un document donné, elle est également utilisée pour filtrer l'index résultant du processus de l'indexation [DAH06]. parmi les méthodes de pondération les plus utilisées dans la fouille de textes on a TF_IDF.

2.2.2 Définition de TF_IDF

Le TF_IDF (de l'anglais TermFrequency-Inverse Document Frequency) est une méthode de pondération souvent utilisée dans la fouille de textes. Par tf, on désigne une mesure qui a un rapport avec l'importance d'un terme pour un document. En général, cette valeur est déterminée par la fréquence du terme dans le document. Par idf, on mesure si le terme est discriminant (ou non uniformément distribué) [JS03]. Cette mesure statistique permet d'évaluer l'importance d'un terme contenu dans un document, relativement à une collection ou un corpus. Le poids augmente proportionnellement au nombre d'occurrences du mot dans le document. Il varie également en fonction de la fréquence du mot dans le corpus.

$TF - IDF$ est calculé à l'aide des formules suivantes :

D'abord :

$$TF(t) = \frac{\text{Nombre du terme t dans le document}}{\text{Nombre Total des termes dans le document}}$$

Et :

$$IDF(t) = \log \frac{\text{Nombre total des documents}}{\text{Nombre des document contenant t}}$$

Et enfin :

$$TF\_IDF = TF \times IDF$$

2.2.3 Exemple

Considérons un document qui contient 100 mots ou le mot Algérie apparait 3 fois.

$$TF(Algérie) = \frac{3}{100} = 0.03$$

Maintenant, supposons qu'on a 10 million documents et le mot Algérie apparait dans 1000 de ces documents. Alors :

$$IDF(Algérie) = \log \frac{10{,}000{,}000}{1{,}000} = 4$$

Donc :

$$TF\_IDF(Algérie) = 0.03 \times 4 = 0.12$$

2.3 TF_IDF et le problème de sélection d'index mono-attribut

Puisque les attributs fréquemment utilisées dans la clause $WHERE$ sont candidats pour être choisi dans la configuration d'index. Donc on utilisera la fonction $TF()$ pour calculer la fréquence de chaque attribut de la manière suivante :

1. A partir de la charge de requêtes, les attribut indexable sont tous mis dans un seul fichier texte .
2. Ensuite, la fonction $TF()$ est appliqué sur chaque attribut pour calculer sa fréquence dans le texte.

Note : On utilise $TF()$ seulement sans $IDF()$ car on a un seul fichier texte ou tout les termes ont une même valeur(pas de termes vides de sens).

2.4 Démarche générale de notre approche

La figure 2.1 représente l'architecture générale de notre solution.A partir de la base de données on utilise une charge de requête afin de produire une configuration d'index mono-attributs. Le schéma décrit les principales étapes de notre solution qui sont :

1. Préparation des données.
2. Calcul des couts.
3. Configuration finale des index.

Dans ce qui suit nous montrons en détail chacune de ces étapes.

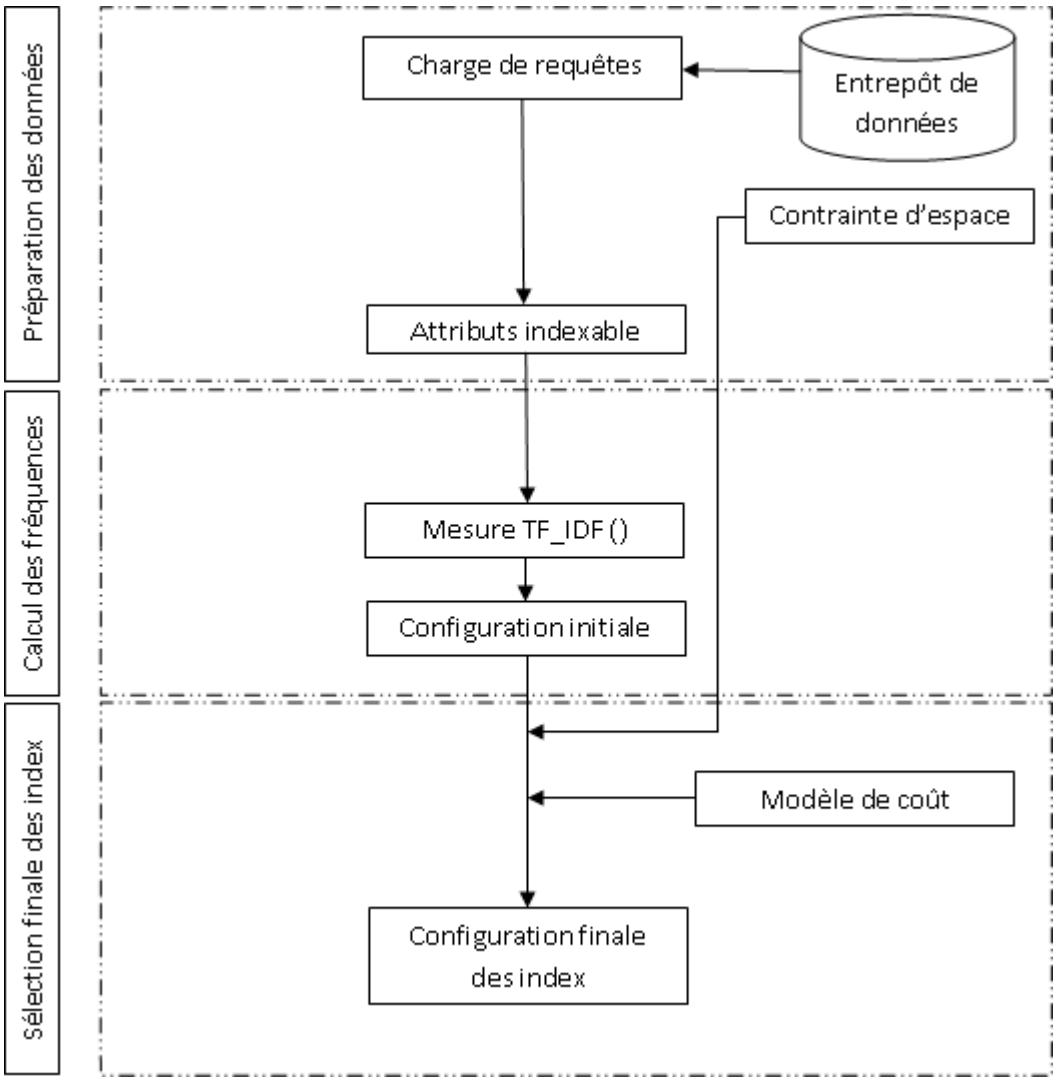

FIGURE 2.1 – Architecture de la solution

a) Préparation des données : On procède comme suit :

1. On extrait la charge de requêtes à partir du journal de transaction de l'entrepôt de données.
2. On analyse séquentiellement toutes les requêtes SQL de cette charge afin d'extraire les attributs dit indexable.Pour cela on ne considère que les attributs présents dans la clause $WHERE$.
3. En suite les attributs résultants seront mis dans un fichier texte en respectant leurs fréquences.

b) Calcul des fréquences On procède comme suit :

1. On parcourt le fichier texte pour extraire les attributs indexables.
2. On applique sur chaque attribut la fonction $TF()$ afin de calculer sa fréquence dans le fichier texte.
3. Ensuite, les attributs avec leurs fréquences seront mis dans un tableau.
4. On met le tableau en ordre descendant.

c) Configuration finale des index

1. On calcule les coûts de chaque attribut du tableau en allant du plus fréquent vers le moins fréquent. On ne calcule le coût d'un index seulement si son prédécesseur est déjà ajouté à la configuration d'index CI(.sauf pour l'index premier dans la liste).
2. Un attribut est ajouté à la configuration d'index CI seulement si après son ajout la contrainte de stockage S reste respectée. (c.à.d. ($Taille(CI) \leq S$)).
3. La configuration d'index finale est obtenu à la fin du processus de calcule de coût.

Chapitre 3

Implémentation et expérimentation

3.1 Introduction

Les travaux de recherches concernant le problème de sélection d'une configuration d'index dans les entrepôts de données relationnels vise à réduire le cout en temps de repense et d'avoir une meilleur taille d'index qui respect la contraint d'espace de stockage. Dans ce contexte, nous nous somme intéressé à l'utilisation des techniques d'agrégation textuelles. Dans la première partie de ce chapitre, nous allons presenter l'outil AFIS(Attribut Frequency for Index Selection) que nous avons elaboré pour validé notre approche.Ensuite, on le compare avec l'outil $fpmax$ de la méthode de recherche de motifs fréquents maximaux.

3.2 Outil de sélection d'index $AFIS$

$AFIS$ est notre outil de sélection des index mono-attribut dans le cadre de validation de notre approche présenté dans le chapitre précédent, notre outil graphique $AFIS$ réalisé en java dont l'architecture et l'interface principale sont présentées respectivement dans les figures 3.1 et 3.2 .

Importation des données Dans ce module, l'utilisateur introduit le contexte d'extraction sous forme d'un fichier texte (.txt), ce dernier est obtenu grâce a un parcourt séquentiel de

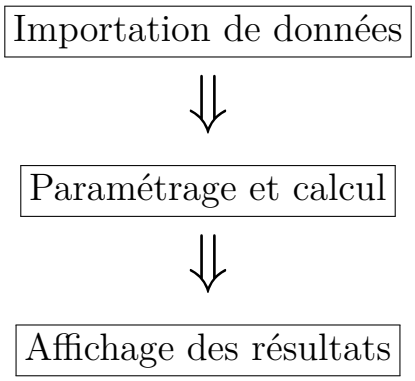

FIGURE 3.1 – Architecture de AFIS

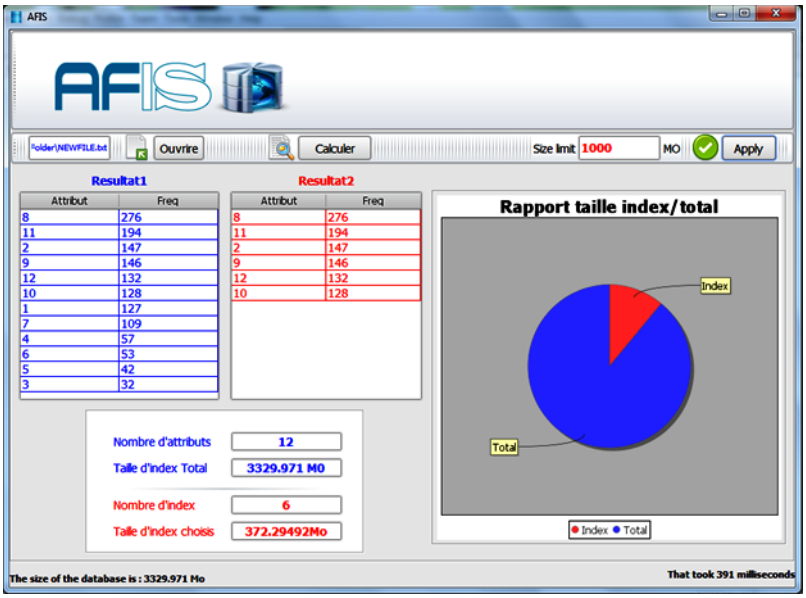

FIGURE 3.2 – Interface de AFIS

toutes les requêtes SQL notamment dans les clauses WHERE dans la charge pour extraire tous les attributs utiles qui peuvent être des support d'index dans la configuration d'index.

Paramétrage et calcul Dans ce module l'utilisateur donne la valeur de la taille de stockage d'index, puis l'outil procède au calcul.

Affichage des résultats Le module affichage des résultats montre la configuration d'index finale avec le coût de stockage. Ainsi un graphique a secteur qui montre le pourcentage de la taille de la configuration finale par rapport a la taille de l'ensemble de tous les index candidats.

3.3 Expérimentation

Environnement expérimental

Entrepôt de données L'entrepôt de données utilisées dans notre étude est issu du benchmark APB1 [Cou98]. Sur cet entrepôt de données. Le schéma en étoile est donné par la figure C. Il est constitué d'une table de faits Actvars (24 786 000 tuples) et de quatre dimensions, Prodlevel (9 000 tuples), Custlevel (900 tuples), Timelevel (24 tuples) et Chanlevel (9 tuples).

Charge de requêtes La Charge de requêtes que nous avons considéré contient 60 requêtes indexables sur 12 attributs présentés dans le tableau suivant 3.2 :

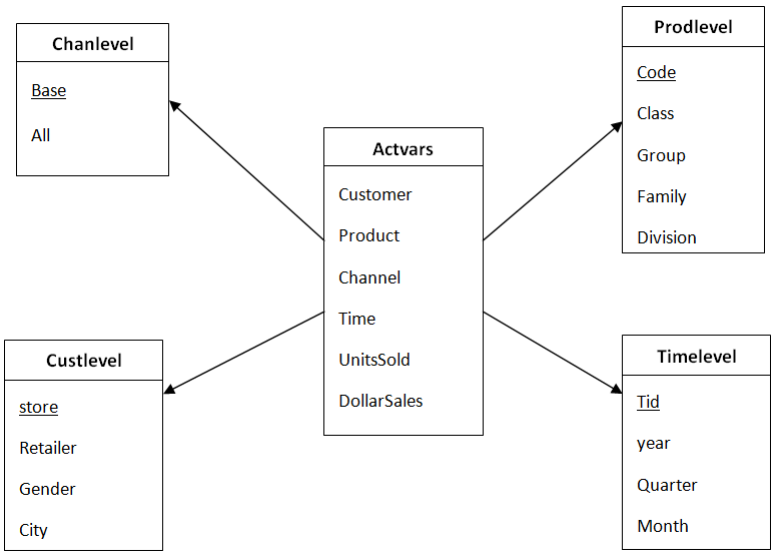

FIGURE 3.3 – Schéma de l'entrepôt expérimentale

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

TABLE 3.1: Caractéristiques des tables de l'entrepôt expérimental

3.4 Environnement de travail

Nous avons implémenté la fonction d'agrégation *AFIS* en JAVA ,il s'agit d'un langage de programmation orienté objet, gratuit et portable ,ce qui lui a permis d'être parmi les langage les plus utilisé .Il a été développé par la firme Sun Micro-système en 1995, cette dernière a été rachetée en 2009 par Oracle.le JDK (JAVA Development Kit) et le JRE (JAVA Runtime Environnent) peuvent être gratuitement télécharges sur le site officiel.*http* : *//www.oracle.com/*.

Nous avons utilisé un ordinateur portable doté d'un processeur Intel (R)CORETM I3 @ 2,10 GHz avec une mémoire vive RAM de 4 GO qui fonctionne sur le système d'exploitation Microsoft Windows 7 intégral 64 bits.

Pour l'éditeur, nous avons choisi L'IDE(Integrated Development Environment) NetBeans IDE 6,8 . Ce dernier est parmi les éditeur java les plus appréciés ,cette forte appréciation est due à de multiple avantage notamment les simplification d'édition et la facilité de la création des interface graphique par l'option≪ *drag and drop* ≫.Ils intègrent les fonctionnalité suivantes :

1. Editeur de textes avec coloriage syntaxique.
2. L'option auto complète (menus contextuels suggère de multiples choix).
3. Génération automatique des conteneurs et dossiers nécessaires à l'organisation d'un

Attribut	Code	Type	Taille	Cardinalité
CLASS_LEVEL	1	Char	12	605
QUARTER_LEVEL	2	Char	6	4
GROUP_LEVEL	3	Char	12	300
FAMILY_LEVEL	4	Char	12	75
LINE_LEVEL	5	Char	12	15
DIVISION_LEVEL	6	Char	12	4
YEAR_LEVEL	7	Char	4	2
MONTH_LEVEL	8	Char	6	12
RETAILER_LEVEL	9	Char	12	99
GENDER_LEVEL	10	char	12	2
ALL_LEVEL	11	char	12	5
CITY_LEVEL	12	char	12	4

TABLE 3.2: Table des attributs

projet et des paquetages des classes.

4. Intégration des commandes Java et de leur option dans menus et des boîtes de dialogue appropriés.
5. Débogueur pour corriger les erreurs.
6. Proposition pour la résolution automatique de quelques erreurs de programmation.

3.5 Coût de stockage d'un index

Le coût de stockage d'un index dépend étroitement de la cardinalité, La taille de l'espace de stockage requis pour un index construit sur un attribut A appartenant à une table T, est donnée par la formule suivante [Inm02], [Wu99] :

$$S(I(A)) = \frac{Card(A) \times |T|}{8}$$

Dans notre travail, on a utilisée un outil pour le calcul du coût d'E/S, cet outil a était développe par l'équipe qui travail sur les motifs fréquents maximaux [ZB10].

3.6 Description de l'expérimentation

Pour évaluer notre approche nous avons procédé à deux expériences qui nous permettent de mesurer l'intérêt de notre solution qui sont :

1. Etude de leffet de la variation de l'espace de stockage d'index.

2. Comparaison avec l'approche des motifs fréquents maximaux.

Effet de la variation de l'espace de stockage d'index Dans le but de connaitre l'effet de la variance de l'espace de stockage sur le nombre d'index et par la suite l'effet de la variance du nombre d'index sur le coût d'entrée sortie. nous avons mené une expérience dans laquelle nous calculons le nombre d'index pour les valeurs du coût de stockage de 0,50 Go à 6.5 Go avec un pas de 0 , 50 Go.

La figure 3.4 montre que l'augmentation de la taille de l'espace de stockage permet l'augmentation du nombre d'index, tandis que La figure 3.5 montre que l'augmentation du nombre d'attribut n'implique pas une diminution du coût d'entrée sortie et cela est dû à la nature des index mono-attribut qui n'apporte pas un effet remarquable sur les requêtes qui utilisent plusieurs attributs dans la clause $WHERE$.

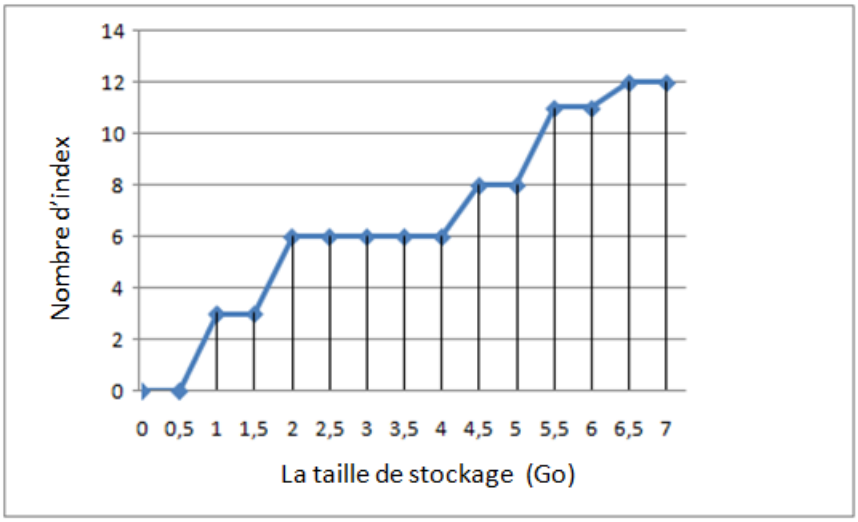

FIGURE 3.4 – Effet de la variation de la taille de stockage sur le nombre d'index

Comparaison avec les motifs fréquents maximaux nous avons effectué une expérimentation dans laquelle nous avons comparé nos résultats avec ceux de l'approche des motifs fréquents maximaux. Nous avons calculé le coût d'exécution et l'espace de stockage. La figure 3.6 montre que le coût de stockage de notre approche à diminuer de 32.92% par rapport à l'approche des motifs maximaux qui représente un gain en espace de stockage, par contre le coût d'exécution de notre approche est largement supérieur ; cela est dû à l'utilisation des index mono-attribut ce qui pénalise les requêtes qui utilisent plusieurs attributs dans la clause $WHERE$ figure 3.7 .

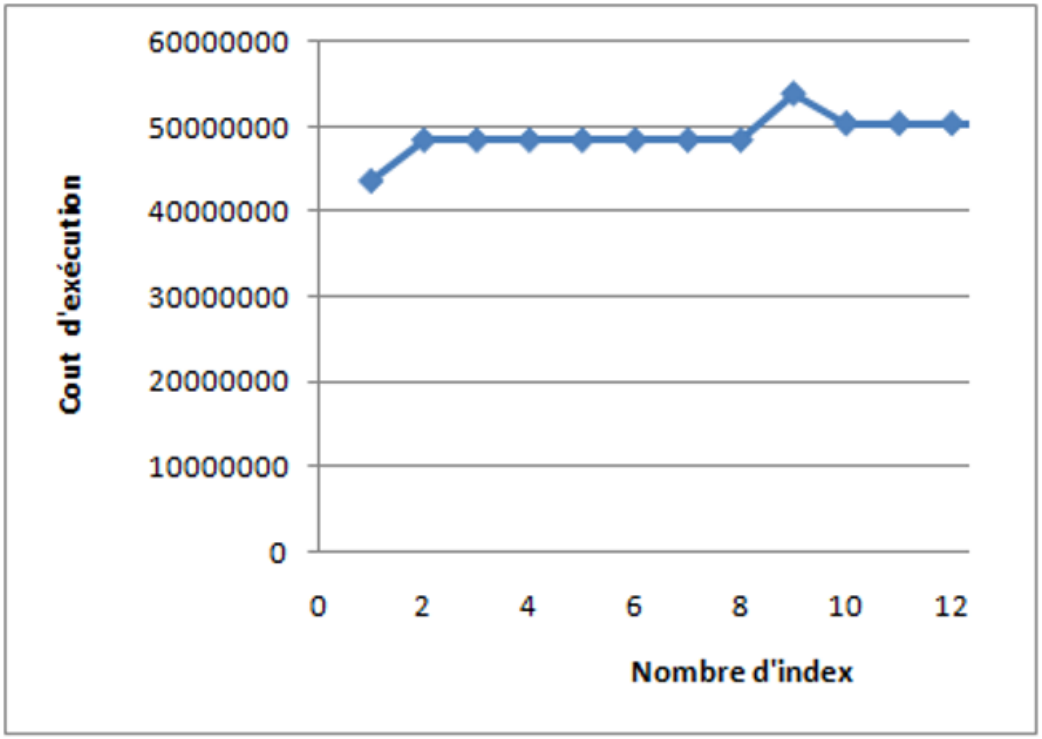

FIGURE 3.5 – Effet de la variation du nombre d'index sur le coût d'execution

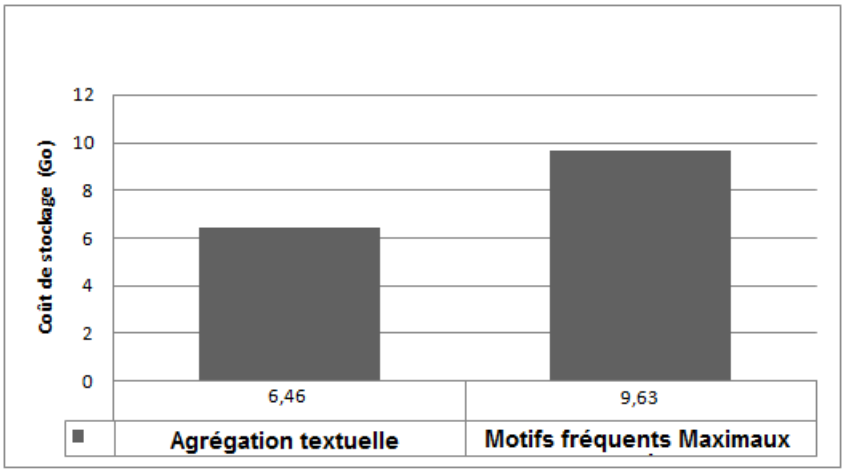

FIGURE 3.6 – Comparaison du coût de stockage

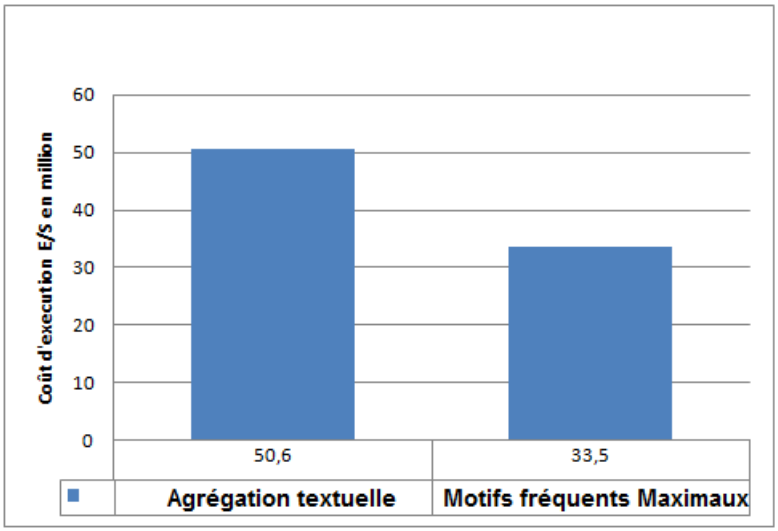

FIGURE 3.7 – Comparaison du coût d' E/S

Conclusion générale

La sélection d'index est un problème NP-complet [Com78], il n'existe pas d'algorithme qui donne une solution optimale en un temps fini.

Le problème de sélection d'un ensemble d'index optimal pour une base de données consiste à construire une configuration d'index optimisant le coût d'exécution d'une charge donnée. Cette optimisation peut être réalisée sous certaines contraintes, comme l'espace de stockage alloué aux index à sélectionner.

Notre but a travers ce travail est d'optimiser la performance du système et de réduire le temps d'accès aux données .

Nous avons adapté la fonction de pondération textuelle TF-IDF() du domaine de recherche d'information (RI) a la résolusion du problème de sélection d'index .

On a effectuée l'implémentation de notre approche en langage JAVA, ensuite on a réalisée une étude expérimentale ainsi qu'une comparaison avec l'approche des motifs fréquents maximaux [ZB10].

Les résultats obtenus nous on montré que l'approche des motifs fréquents maximaux est largement meilleur, et ça revient au fait que notre approche utilise seulement des index mono-attributs .

la fonction de recherche d'information malgré qu'elle nous donne des informations intéressantes sur les termes, mais elle n'exploite pas les compositions des mots, alors qu'avec les motifs fréquents on peut exploiter toutes les combinaisons .

Enfin, nous concluons que les index mono-attributs ne sont pas intéressent dans le contexte des entrepôts de données .

Annexe

Requête	Fréquence
Q1	3
Q2	7
Q3	10
Q4	10
Q5	12
Q6	3
Q7	3
Q8	16
Q9	4
Q10	12
Q11	4
Q12	21
Q13	21
Q14	21
Q15	3
Q16	7
Q17	10
Q18	10
Q19	20
Q20	4
Q21	12
Q22	4
Q23	21
Q24	21
Q25	21
Q26	3
Q27	7
Q28	10
Q29	10
Q30	20

Requête	Fréquence
Q31	12
Q32	4
Q33	12
Q34	4
Q35	21
Q36	3
Q37	15
Q38	10
Q39	3
Q40	12
Q41	12
Q42	10
Q43	18
Q44	11
Q45	12
Q46	5
Q47	21
Q48	12
Q49	12
Q50	10
Q51	15
Q52	30
Q53	5
Q54	20
Q55	18
Q56	3
Q57	4
Q58	10
Q59	2
Q60	5

TABLE 3.3: Les fréquence des requêtes de la charge

Requête 1 select Time_level,count(*) from ACTVARS A,PRODLEVEL P where A.PRODUCT_LEVEL=P.CODE_LEVEL and P.Class_LEVEL='PO0HV1RICH5W' group by Time_level	Requête 2 select line_level,sum(Dollarcost) from ACTVARS A,PRODLEVEL P where A.PRODUCT_LEVEL=P.CODE_LEVEL and P.Class_LEVEL='CI493YZ9KZUJ' group by line_level
Requête 3 SELECT count(*) FROM ACTVARS A,PRODLEVEL P WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND P.Class_LEVEL='FDXAQ1N5U026'	Requête 4 SELECT Time_level,Avg(UNITSSOLD) FROM ACTVARS A,Timelevel T WHERE A.TIME_LEVEL=T.TID AND T.Quarter_level='Q1' group by Time_level
Requête 5 SELECT division_level,count(*) FROM ACTVARS A,PRODLEVEL P WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND P.group_LEVEL='E4NJTW0ZR9FN' group by division_level	Requête 6 SELECT Max(UNITSSOLD) FROM ACTVARS A,Timelevel T WHERE A.TIME_LEVEL=T.TID AND T.Quarter_level='Q2'
Requête 7 SELECT Time_level,count(*) FROM ACTVARS A,PRODLEVEL P WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND P.family_LEVEL='BEMFVK0N8125' group by Time_level	Requête 8 SELECT year_level,sum(Dollarcost) FROM ACTVARS A,PRODLEVEL P, Timelevel T WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.time_level=T.TID AND P.family_LEVEL='UJHZ4TZMJT6V' group by year_level
Requête 9 SELECT month_level,count(*) FROM ACTVARS A,Timelevel T WHERE A.time_level=T.TID AND T.Quarter_level='Q3' group by month_level	Requête 10 SELECT Sum(Dollarcost) FROM ACTVARS A,PRODLEVEL P WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND P.LINE_LEVEL = 'MJ1F1U1EG009'
Requête 11 SELECT Product_level,count(*) FROM ACTVARS A,Timelevel T WHERE A.TIME_LEVEL=T.TID AND T.Quarter_level='Q4' group by Product_level	Requête 12 SELECT retailer_level,Avg(unitssold) FROM ACTVARS A,PRODLEVEL P ,Custlevel C WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.customer_level=C.Store_level AND P.DIVISION_LEVEL = 'BCR2T4K2K9D3' group by retailer_level

TABLE 3.4: La liste des requêtes

Requête 13 SELECT count(*) FROM ACTVARS A,PRODLEVEL P WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND P.DIVISION_LEVEL = 'XRLXY6H61SLC'	Requête 14 SELECT Customer_level,Sum(Dollarcost) FROM ACTVARS A,PRODLEVEL P WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND P.DIVISION_LEVEL = 'RC5406URP1IE group by Customer_level
Requête 15 SELECT all_level,count(*) FROM ACTVARS A,PRODLEVEL P ,Chanlevel H WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.channel_level=H.Base_level AND P.DIVISION_LEVEL = 'G4HA5YITG3H7' group by Channel_level	Requête 16 SELECT count(*) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.year_LEVEL = '1995'
Requête 17 SELECT Product_level,Sum(Dollarcost) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.year_LEVEL = '1996' group by Product_level	Requête 18 SELECT division_level,Avg(Unitssold) FROM ACTVARS A,TIMELEVEL T Prodlevel P WHERE A.TIME_LEVEL=T.TID AND A.product_level=P.Code_level AND T.month_LEVEL = '1' group by division_level
Requête 19 SELECT count(*) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '2'	Requête 20 SELECT Product_level,count(*) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '3' group by Product_level
Requête 21 SELECT division_level,Sum(Dollarcost) FROM ACTVARS A,TIMELEVEL T ,Prodlevel P WHERE A.TIME_LEVEL=T.TID AND A.product_level=P.Code_level AND T.month_LEVEL = '4' group by division_level	Requête 22 SELECT Avg(Unitssold) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '5'

Requête 23 SELECT Product_level,count(*) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '6' group by Product_level	Requête 24 SELECT division_level,Sum(Dollarcost) FROM ACTVARS A,TIMELEVEL T ,Prodlevel P WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '7' AND A.product_level=P.Code_level group by division_level
Requête 25 SELECT Sum(Dollarcost) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '8'	Requête 26 SELECT Customer_level,count(*) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '9' group by Customer_level
Requête 27 SELECT year_level,Sum(Dollarcost) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '10' group by year_level	Requête 28 SELECT count(*) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '11'
Requête 29 SELECT Product_level,Time_level,Avg(unitssold) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '12' group by Product_level,Time_level	Requête 30 SELECT year_level,month_level, Max(unitssold) FROM ACTVARS A,CUSTLEVEL C ,Timelevel T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND C.RETAILER_LEVEL = 'Z6OFS4YAAD4J' AND A.time_level=T.TID group by year_level,month_level
Requête 31 SELECT Product_level,Time_level,Avg(unitssold) FROM ACTVARS A,TIMELEVEL T WHERE A.TIME_LEVEL=T.TID AND T.month_LEVEL = '12' group by Product_level,Time_level	Requête 32 SELECT year_level,month_level, Max(unitssold) FROM ACTVARS A,CUSTLEVEL C Timelevel T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND C.RETAILER_LEVEL = 'Z6OFS4YAAD4J' AND A.time_level=T.TID group by year_level,month_level

Requête 33 SELECT count(*) FROM ACTVARS A,CUSTLEVEL C WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND C.RETAILER_LEVEL = 'RQJNEN0UPKMQ'	Requête 34 SELECT Product_level,Time_level, Min(unitssold) FROM ACTVARS A,CUSTLEVEL C WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND C.RETAILER_LEVEL = 'NXEYFSIQE3JM' group by Product_level,Time_level
Requête 35 SELECT year_level,Sum(Dollarcost) FROM ACTVARS A,CUSTLEVEL C ,Timelevel T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.time_level=T.TID AND C.GENDER_LEVEL='M' group by year_level	Requête 36 SELECT count(*) FROM ACTVARS A,CHANLEVEL CH WHERE A.CHANNEL_LEVEL=CH.BASE_LEVEL AND CH.ALL_LEVEL ='ABCDEFGHIJKL'
Requête 37 SELECT Channel_level,Time_level, Sum(Dollarcost) FROM ACTVARS A,CHANLEVEL CH WHERE A.CHANNEL_LEVEL=CH.BASE_LEVEL AND CH.ALL_LEVEL ='BCDEFGHIJKLM' group by Channel_level, Time_level	Requête 38 SELECT class_level,year_level, Time_level, Avg(Unitssold) FROM ACTVARS A,CHANLEVEL CH ,Prodlevel P ,Timelevel T WHERE A.CHANNEL_LEVEL=CH.BASE_LEVEL AND A.product_level=P.Code_level AND A.time_level=T.TID AND CH.ALL_LEVEL ='CDEFGHIJKLMN' group by class_level,year_level, Time_level
Requête 39 SELECT count(*) FROM ACTVARS A,CHANLEVEL CH WHERE A.CHANNEL_LEVEL=CH.BASE_LEVEL AND CH.ALL_LEVEL ='DEFGHIJKLMNO	Requête 40 SELECT Time_level,count(*) FROM ACTVARS A,CHANLEVEL CH WHERE A.CHANNEL_LEVEL=CH.BASE_LEVEL AND CH.ALL_LEVEL ='EFGHIJKLMNOP' group by Time_level

Requête 41 SELECT year_Level,month_level, sum(dollarcost) FROM ACTVARS A,CUSTLEVEL C,PRODLEVEL P ,Timelevel T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.time_level=T.TID AND A.PRODUCT_LEVEL=P.CODE_LEVEL AND P.CLASS_LEVEL='CI493YZ9KZUJ' AND C.RETAILER_LEVEL='RQJNEN0UPKMQ' AND C.CITY_LEVEL='Bordeaux' group by year_level,month_level	Requête 42 SELECT sum(dollarcost) FROM ACTVARS A, PRODLEVEL P,TIMELEVEL T WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.TIME_LEVEL=T.TID AND T.QUARTER_LEVL='Q2' AND T.YEAR_LEVEL='1996' AND P.CLASS_LEVEL='FDXAQ1N5U026'
Requête 43 SELECT Customer_Level, Time_level, Avg(Unitssold) FROM ACTVARS A, CHANLEVEL H,PRODLEVEL P, TIMELEVEL T,CUSTLEVEL C WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.TIME_LEVEL=T.TID AND A.CUSTOMER_LEVEL=C.STORE_LEVEL A.CHANNEL_LEVEL=H.BASE_LEVEL and P.CLASS_LEVEL='FDXAQ1N5U026' AND H.ALL_LEVEL ='ABCDEFGHIJKL' AND T.QUERTER_LEVEL='Q1' AND C.GENDER_LEVEL='F' group by Customer_level,Time_level	Requête 44 SELECT year_Level, division_level, Max(Unitssold) FROM ACTVARS A, CUSTLEVEL C,TIMELEVEL T ,Prodlevel P WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.product_level=P.Code_level AND A.TIME_LEVEL=T.TID and T.MONTH_LEVEL='1' AND C.RETAILER_LEVEL ='RQJNEN0UPKMQ' AND C.GENDER_LEVEL='F' AND C.CITY_LEVEL='Poitiers' group by year_level, division_level

Requête 45 SELECT sum(dollarcost), Avg(Unitssold) FROM ACTVARS A, CUSTLEVEL C,TIMELEVEL T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.TIME_LEVEL=T.TID and T.MONTH_LEVEL='12' AND C.RETAILER ='RQJNEN0UPKMQ' AND C.CITY_LEVEL='Dijon'	Requête 46 SELECT Customer_Level, Time_level,Min(unitssold) FROM ACTVARS A, CHANLEVEL H,TIMELEVEL T, CUSTLEVEL C WHERE A.CHANNEL_LEVEL=H.BASE_LEVEL AND A.CUTOMER_LEVEL=C.STORE_LEVEL AND A.TIME_LEVEL=T.TID and T.YEAR_LEVEL='1996' AND T.QUARTER_LEVEL='Q3' AND H.ALL_LEVEL='DEFGHIJKLMNO' AND C.GENDER_LEVEL='M' group by Customer_level, Time_level
Requête 47 SELECT month_Level, all_level, Time_level, Sum(Dollarcost) FROM ACTVARS A, CUSTLEVEL C,PRODLEVEL P,TIMELEVEL T ,Chanlevel H WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.channel_level=H.Base_level AND A.TIME_LEVEL=T.TID and T.YEAR_LEVEL='1996' AND C.RETAILER_LEVEL ='NXEYFSIQE3JM' AND C.CITY_LEVEL='Paris' AND P.LINE_LEVEL='MJ1F1U1EG009' group by month_level,all_level, Time_level	Requête 48 SELECT Avg(unitssold) FROM ACTVARS A, CHANLEVEL H,PRODLEVEL P,TIMELEVEL T WHERE A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.CHANNEL_LEVEL=H.BASE_LEVEL and A.TIME_LEVEL=T.TID and T.MONTH_LEVEL='1' AND P.DIVISION_LEVEL='XRLXY6H61SLC' AND H.ALL_LEVEL='BCDEFGHIJKLM'

Requête 49 SELECT Customer_Level, Product_level, Channel_level, sum(dollarcost) FROM ACTVARS A, CHANLEVEL H,CUSTLEVEL C,PRODLEVEL P WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.CHANNEL_LEVEL=H.BASE_LEVEL and P.FAMILY_LEVEl='AGG214DG271Q' AND C.RETAILER_LEVEL='NXEYFSIQE3JM' AND C.GENDER_LEVEL='M' AND H.ALL_LEVEL='DEFGHIJKLMNO' group by Customer_level, Product_level, Channel_level	Requête 50 SELECT division_Level, year_level, Max(Unitssold) FROM ACTVARS A, CHANLEVEL H,CUSTLEVEL C,TIMELEVEL T ,Prodlevel P WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.product_level=P.Code_level AND A.CHANNEL_LEVEL=H.BASE_LEVEL and A.TIME_LEVEL=T.TID and T.MONTH_LEVEL='4' AND C.RETAILER_LEVEL='NXEYFSIQE3JM' AND C.CITY_LEVEL='Poitiers' AND C.GENDER8LEVEL='F' AND H.ALL_LEVEL='BCDEFGHIJKLM' H.ALL_LEVEL='BCDEFGHIJKLM'
Requête 51 SELECT Min(Unitssold) FROM ACTVARS A, CHANLEVEL H,CUSTLEVEL C,PRODLEVEL P,TIMELEVEL T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.CHANNEL_LEVEL=H.BASE_LEVEL and A.TIME_LEVEL=T.TID and T.MONTH_LEVEL='7' AND P.GROUP_LEVEL='E4NJTW0ZR9FN' AND C.RETAILER_LEVEL='Z6OFS4YAAD4J' AND H.ALL_LEVEL='EFGHIJKLMNOP'	Requête 52 SELECT Customer_Level, Channel_level, sum(dollarcost) FROM ACTVARS A, CHANLEVEL H,CUSTLEVEL C,PRODLEVEL P,TIMELEVEL T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.CHANNEL_LEVEL=H.BASE_LEVEL and A.TIME_LEVEL=T.TID and T.MONTH_LEVEL='11' AND P.CLASS_LEVEL ='FDXAQ1N5U026' AND C.RETAILER_LEVEL ='RQJNEN0UPKMQ' AND C.CITY_LEVEL='Poitiers' AND H.ALL_LEVEL='DEFGHIJKLMNO' group by Customer_level, Channel_level

Requête 53 SELECT line_Level, month_level, Avg(Unitssold) FROM ACTVARS A, CHANLEVEL H,CUSTLEVEL C,PRODLEVEL P,TIMELEVEL T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.CHANNEL_LEVEL=H.BASE_LEVEL AND A.TIME_LEVEL=T.TID and T.YEAR_LEVEL='1995' AND T.QUARTER_LEVEL='Q1' AND P.GROUP_LEVEL='E4NJTW0ZR9FN' AND C.RETAILER_LEVEL='RQJNEN0UPKMQ' AND H.ALL_LEVEL='BCDEFGHIJKLM' group by line_level, month_level	Requête 54 SELECT Min(Unitssold) FROM ACTVARS A, CHANLEVEL H,CUSTLEVEL C,PRODLEVEL P,TIMELEVEL T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.CHANNEL_LEVEL=H.BASE_LEVEL and A.TIME_LEVEL=T.TID and T.YEAR_LEVEL='1995' AND T.QUARTER_LEVEL='Q1' AND T.MONTH_LEVEL in('2','3','4') AND P.CLASS_LEVEL='CI493YZ9KZUJ' AND C.RETAILER_LEVEL in ('Z6OFS4YAAD4J','RQJNEN0UPKMQ') AND C.GENDER_LEVEL='F' AND C.CITY_LEVEL='Poitiers' AND H.ALL_LEVEL ='ABCDEFGHIJKL'
Requête 55 SELECT Customer_Level, Product_level, Time_level, Channel_level, Max(Unitssold) FROM ACTVARS A, CHANLEVEL H,CUSTLEVEL C,PRODLEVEL P,TIMELEVEL T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.CHANNEL_LEVEL=H.BASE_LEVEL and A.TIME_LEVEL=T.TID and T.YEAR_LEVEL='1996' AND T.MONTH_LEVEL in('5','6','7') AND P.CLASS_LEVEL='FDXAQ1N5U026' AND C.GENDER_LEVEL='M' AND C.RETAILER_LEVEL='RQJNEN0UPKMQ' AND H.ALL_LEVEL ='DEFGHIJKLMNO' group by Customer_level, Product_level, Time_level, Channel_level	Requête 56 SELECT Sum(Dollarcost) FROM ACTVARS A,Timelevel T WHERE A.time_level=T.TID AND C.GENDER_LEVEL='F'

Requête 57 SELECT month_level,Sum(Dollarcost) FROM ACTVARS A,CUSTLEVEL C ,Timelevel T WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.time_level=T.TID AND C.City_LEVEL='Poitiers' group by month_level	Requête 58 SELECT time_level,avg(Dollarcost) FROM ACTVARS A,CUSTLEVEL C WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND C.City_LEVEL='Bordeaux' group by time_level
Requête 59 SELECT avg(Dollarcost) FROM ACTVARS A,CUSTLEVEL C WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND C.City_LEVEL='Paris	Requête 60 SELECT year_level,max(Dollarcost) FROM ACTVARS A,CUSTLEVEL C WHERE A.CUSTOMER_LEVEL=C.STORE_LEVEL AND C.City_LEVEL='Dijon' group by year_level

Bibliographie

[A13] Benmelouka A. Recherche des règles d'association pour la sélection des index multi-attributs. Master's thesis, 2013.

[AD94] A Ikinci A Dogac, Y Erisik. An automated index selection tool for oracle7 : Maestro 7. In *Software Research and Development Center*, 1994.

[Baa05] N. Baaziz. Adaptive watermarking schemes based on a redundant contourlet transform. In *IEEE International Conference on Image Processing ICIP 2005*, 2005.

[BK08] Bellatreche L. ParAdmin. Boukhalfa K., Caffiau S. Un outil d'assistance à l'administration et tuning d'un entrepôt de données. In *Revue des Nouvelles Technologies de l'Information*, 2008.

[Bk10] Ziani B. Boukhalfa k., Bellatreche l. Index de jointure binaires : Stratégies de sélection etude de performances. In *6èmes journées francophones sur les Entrepôts de Données et l'Analyse en ligne (EDA 2010)*, 2010.

[Che11] Max Chevalier. *Usagers et recherche d'information.* PhD thesis, Université Paul Sabatier, 2011.

[Com78] D Come. *The difficulty of optimum index selection. ACM Transactions on Database Systems.* PhD thesis, 1978.

[Cou98] O. Council. Apb-1 olap benchmark, release ii. In *http ://www.olapcouncil.org/research/resrch.htm*, 1998.

[CS08] Narasayya V Chaudhuri S. Index selection for databases : A hardness study and a principaled heuristic solution. In *In IEEE Transactions on Knowledge and Data Engineering*, 2008.

[DAH06] F DAHAK. Indexation des documents semi-structurÃĬs : Proposition dâĂŹune approche basÃĬe sur le fichier inversÃĬ et le âĂIJtrieâĂİ. Master's thesis, Institut National de formation en Informatique (I.N.I), 2006.

[Dar06] J. Darmont. *Techniques de fouille de données pour l'optimisation automatique des performances des entrepôts de données.* PhD thesis, au laboratoire ERIC, Universite Lumiere Lyon 2, 2006.

[DC81] W Blasgen N Gray F King G Lindsay R Lorie W Mehl G Price F Putzolu G Selinger M Schkolnick R Slutz L Traiger W Wade A Yost D Chamberlin, M Astrahan. A history and evaluation of system r. In *Magazine Communications of the ACM*, 1981.

[FS88] Tiberio P Finkelstein S, Schkolnick M. Physical database design for relational databases. In *ACM Transactions on Database Systems*, 1988.

[Gun93] T.I Gundem. Near optimal multiple choice index selection for relational databases. In *Computers Mathematics with Applications*, 1993.

[Gun99] T.I. Gundem. Near optimal multiple choice index selection for relational databases. In *Computers Mathematics with Applications*, 1999.

[GV00] D. Zilio G. Lohman A. Skelley G. Valentin, M. Zuliani. An optimizer smart enough to recommend its own indexes. In *16th International Conference on Data Engineering (ICDE 00)*, 2000.

[GVA00] Zilio D Lohman G G Valentin, Zuliani M and Skelley A. Db2 advisor : An optimizer smart enough to recommend its own indexes. In *16th international conference on data engineering*, 2000.

[Inm02] H. Inmon. *Building the Data Warehouse*. John Wiley Sons, 2002.

[J99] Theodore J. Performance measurements of compressed bitmap indices. In *25th International Conference on Very Large Data Bases*, 1999.

[JK03] I. Ljubièc J. Kratica. A genetic algorithm for the index selection problem. In *EvoWorkshops , Essex*, 2003.

[JS03] Dray G Jaillet S, Teisseire M. *Adéquation des modèles de représentation aux méthodes de catégorisation*. PhD thesis, Université Montpellier, 2003.

[K05] Aouiche K. *Techniques de fouille de données pour l'optimisation automatique des performances des entrepôts de données*. PhD thesis, 2005.

[KW85] M Schkolnick K.Y. Whang. Index selection in relational databases. In *International Conference on Foundations of Data Organization (FODO 85)*, 1985.

[MG02] E. Saltarelli M. Golfarelli, S. Rizzi. Index selection for data warehousing. In *4th International Workshop on Design and Management of Data Warehouses (DMDW 02)*, 2002.

[MR07] Bellatreche L Missaoui R, et Necir H. Data mining approach for selecting bitmap join indices. In *Journal of Computing Science and Engineering*, 2007.

[N14] ABBAS N. Vers une extension sémantique de l'analyse formelle de concepts : Application à la recherche d'informations. Master's thesis, 2014.

[NEC13] H. NECIR. *IntÃlgration d'Agents Intelligents dans les Approches Data Mining pour la Sélection des Index et des Vues Matérialisées dans les Entrepôts de Données Relationnels*. PhD thesis, Université des Sciences et Technologies Houari Boumediene, 2013.

[OP95] Graefe G O'Neil P. Multi-table joins through bitmapped join indices. 1995.

[OP97] Quass D O'Neil P. Improved query performance with variant indexes. In *titre du proceeding de la conférence*, 1997.

[PB01] J Darmont L Gruenwald P Brunel, V Rollin. Dbms auto-indexing using datamining techniques. In *Université d'oklahoma USA*, 2001.

[RF92] B. Navathe R. Frank, E. Omiecinski. Adaptive and automated index selection in rdbms. In *3rd International Conference on Extending Database Technology (EDBT 92)*, 1992.

[RFN92] E Omiecinski R Frank and B Navathe. Adaptive and automated index selection in rdbms. In *3rd international conference on extending database technology (LNCS EDBT 92)*, 1992.

[SC93] T. Chang S. Choenni, M. Blanken. Index selection in relational databases. In *5th International Conference on Computing and Information (ICCI 93)*, 1993.

[SC97] U. Dayal S. Chaudhuri. Data warehousing and olap for decision support. In *ACM SIGMOD International Conference on Management of Data (SIGMOD 97)*, 1997.

[SCC93] M Blanken S Choenni and T Chang. Index selection in relational databases. In *5th International Conference on Computing and Information (ICCI 93)*, 1993.

[SF82] P Tiberio S.J Finkelstein, M Schkolnick. Dbdsgn : A physical database design tool for system r. In *IEEE database eng*, 1982.

[SS02] E. Rizzi S. Saltarelli, M. Golfarelli. Index selection for data warehousing. In *The 4th International Workshop on Design and Management of Data Warehouses*, 2002.

[TD03] Ljubic I Tosic D, Kratica J. Genetic algorithm for the index selection problem. In *Applications of Evolutionary Computing, EvoWorkshops 2003*, 2003.

[Wha87] K Whang. Index selection in relational databases. In *International conference on foundations of data organization*, 1987.

[Wu99] M. Wu. Query optimization for selections using bitmap. In *ACM SIGMOD international conference on management of data*, 1999.

[YF03a] J. Reouven Y. Feldman. A knowledge based approach for index selection in relational databases. In *Expert System with Application*, 2003.

[YF03b] J. Reouven Y.A. Feldman. A knowledge-based approach for index selection in relational databases. In *Expert System with Applications*, 2003.

[ZB10] OUINTEN Y ZIANI B. Vers l'auto-sélection des index dans les entrepôts de données : une approche basée sur la recherche des motifs fréquents maximaux. In *CARI 2010*, 2010.

[ZG00] Zuliani M Zilio G, Valentin G. Db2 advisor : An optimizer smart enough to recommend its own indexes. In *International Conference on Data Engineering (ICDE 00),San Diego,USA*, 2000.