ParAdmin: Un Outil d'Assistance à l'Administration et Tuning d'un Entrepôt de Données

Ladjel Bellatreche *, Kamel Boukhalfa *, Sybille Caffiau * **

* LISI/ENSMA - Université de Poitiers - Futuroscope
(bellatreche, boukhalk, caffiaus)@ensma.fr,
** INRIA Domaine de Volucean, Rocquencourt

Résumé. Les entrepôts de données ont rendu les tâches d'administration et de tuning plus complexes que dans les bases de données traditionnelles. Cela est dû aux caractéristiques des entrepôts de données : la volumétrie, les requêtes complexes, les délais de réponse exigés par les décideurs et la gestion de l'évolution. Dans ce contexte, une panoplie de techniques d'optimisation ont été proposées durant la phase de conception physique. Pour chacune d'entre elles un nombre important d'algorithmes de sélection existe, chacun ayant ses propres paramètres. Un autre choix déterminant est à faire entre les techniques d'optimisation similaires - pour optimiser une requête donnée, deux techniques peuvent être candidates. Durant la tâche de conception physique, l'administrateur doit donc effectuer de multiple choix. Dans ce papier, nous montrons d'abord les difficultés qu'un administrateur peut rencontrer durant la phase de conception physique. Deuxièmement, nous présentons une méthode de tuning basée sur la fragmentation horizontale et les index de jointure binaires. Finalement, nous proposons un outil d'administration, appelé *ParAdmin*, qui permet d'assister l'administrateur en proposant la réalisation interactive des tâches de conception physique et de tuning. Elles sont définies à l'aide des choix: de la ou des structures d'optimisation, des algorithmes de sélection, des paramètres, de tuner ou non et de visualiser les recommandations de performance.

1 Introduction

Les entrepôts de données ont largement contribué à l'évolution de la conception physique traditionnelle dédiée aux applications de type OLTP, où la tâche d'un administrateur était concentrée sur la gestion des utilisateurs et d'un nombre restreint de techniques d'optimisation (les index, les différentes implémentations de jointure - boucles imbriquées, sort merge, hash, etc.). L'apparition des entrepôts de données a rendu la tâche d'administration de plus en plus complexe. Cette complexité est due à leurs caractéristiques. L'une d'entre elles est le volume de données. Par exemple, le schéma en étoile de l'entrepôt de données relationnel de la compagnie *General Motors Corporation* dépasse 1.5 téraoctets et la table des faits a plus de deux milliard d'instances (Gill, 2000). De plus, les requêtes décisionnelles sont complexes car elles contiennent des jointures, des sélections et des agrégations. De même, il est impératif d'exécuter ces requêtes en un temps raisonnable. Enfin, les entrepôts de données évoluent

constamment. Cette évolution concerne à la fois les requêtes décisionnelles, le schéma et les instances de l'entrepôt. Pour garantir une meilleure utilisation de l'entrepôt de données, l'administrateur doit réaliser deux étapes principales : *la conception physique* et *le tuning*.

Durant la phase de conception physique, l'administrateur doit sélectionner des techniques d'optimisation pour satisfaire les requêtes décisionnelles (figure 1). Plusieurs techniques ont été proposées, chacune a ses propres avantages, ses inconvénients et certaines ont de fortes similarités. Pour chaque technique, plusieurs algorithmes de sélection sont possibles. En conséquence, l'administrateur doit faire des choix au niveau (1) des techniques d'optimisation, (2) de la nature de sélection et (3) des algorithmes de sélection.

Le choix des techniques d'optimisation : Si nous explorons la littérature et les éditeurs de gestion de bases de données commerciaux, nous trouvons une large panoplie de techniques que nous pouvons classer en deux catégories : (i) les *techniques redondantes* qui nécessitent un espace de stockage et un coût de maintenance (les vues matérialisées, les index avancées, la fragmentation verticale, etc.) et (ii) les *techniques non redondantes* ne nécessitant ni un espace de stockage ni un coût de maintenance (la fragmentation horizontale, le traitement parallèle, etc.). Pour optimiser ses requêtes, l'administrateur peut choisir une ou plusieurs techniques parmi ces deux catégories.

Le choix de la nature de la sélection : Deux types de sélection de techniques d'optimisation existent : la *sélection isolée* et la *sélection multiple*. Dans la première sélection, une seule technique (redondante ou non redondante) est sélectionnée. Cette sélection a été largement étudiée (Aouiche et al., 2005; Bellatreche et al., 2006, 2008; Chaudhuri, 2004; Johnson, 1999; Chee-Yong, 1999; Golfarelli et al., 2002; O'Neil et Quass, 1997). La sélection multiple permet de sélectionner plusieurs techniques à la fois. Elle est principalement motivée par les fortes similarités entre les techniques d'optimisation. Les travaux majeurs dans cette catégorie sont principalement concentrés sur la sélection des vues matérialisées et les index (Labio et al., 1997; Sanjay et al., 2000; Talebi et al., 2008).

Le choix de l'algorithme de sélection : Dans chaque classe de sélection, nous trouvons un large choix d'algorithmes de sélection simples (glouton par exemple) et complexes (algorithmes basés sur la programmation linéaire, génétique, recuit simulé, etc.). Chaque algorithme a ses propres paramètres que l'administrateur doit configurer.

Cet état de l'art montre les difficultés d'administration d'un entrepôt de données. Elles sont liées à la fois aux différents choix cités au dessus et également au choix des tables et des attributs participant à la définition de ses techniques. Pour mieux comprendre ces difficultés, considérons le scénario suivant dans lequel un administrateur a un ensemble de requêtes à optimiser. Pour ce faire, il doit répondre aux questions suivantes :

1. Quelles techniques d'optimisation dois-je choisir ?
   Dans ce scénario, nous supposons qu'il choisit une seule technique concernant les index de jointure en étoile (Aouiche et al., 2005).
2. Quels tables et attributs dois-je indexer ?
3. Quel algorithme de sélection dois-je utiliser ?
   Deux types majeurs d'algorithmes ont été proposés pour sélectionner des index de join-

ture binaires : les algorithmes gloutons (Bellatreche et al., 2007) et les algorithmes basés sur les techniques de fouille de données (Aouiche et al., 2005; Bellatreche et al., 2008).

4. Quels paramètres d'algorithmes de sélection dois-je configurer ?

La conception physique obtenue après avoir répondu aux différentes questions génère une configuration Δ contenant différentes techniques d'optimisation et leurs paramètres. L'entrepôt de données évolue au cours de sa durée de vie, en conséquence, il doit pouvoir être tuné par l'administrateur. Cette évolution concerne plusieurs aspects : (i) le contenu des tables, (ii) la taille des techniques d'optimisation sélectionnées lors de la conception physique (les vues matérialisées, les index et les partitions), (iii) les fréquences des requêtes/mises à jour, (iv) l'ajout et la suppression de requêtes, etc. Ces changements nécessitent une phase de réglage de manière à maintenir une meilleure performance de l'entrepôt et surtout éviter sa dégradation. Le rôle principal de la phase de tuning est de surveiller et de diagnostiquer l'utilisation de la configuration Δ et des différentes ressources qui lui sont affectées (comme le tampon, l'espace de stockage, etc.).
Récemment, plusieurs travaux sur le tuning ont été développés, spécialement des laboratoires de grands éditeurs de SGBDs : SQL Server, BD2, ORACLE. (Chaudhuri et Narasayya) du groupe de Microsoft ont montré l'intérêt du tuning et ses effets sur la performance de requêtes [1].

Malheureusement, peu d'outils existent pour assister et accompagner un administrateur dans sa tâche de conception physique et de tuning. Certains éditeurs de SGBDs ont mis en place des outils pour gérer les techniques d'optimisation, comme *Index Advisor Tool* (développé par DB2 (Valentin et al., 2000)) et *Data Partitioning Advisor* (développé par ORACLE11g) qui offre des recommandations de partitionnement des tables (Sheet, 2007). Les principales limitations de ces outils sont : l'administration d'une et une seule technique d'optimisation, l'absence du choix de l'algorithmes de sélection (algorithme de sélection est imposé), et la non prise en compte des similarités entre les différentes techniques. (Kraft et al., 2007) ont proposé un outil, appelé Statistics API qui permet " d'interroger " une base de données afin de collecter les différentes statistiques. Celles-ci concernent : la méta donnée des tables, méta donnée des colonnes, méta donnée des index, les statistiques sur les tables, les facteurs de sélectivité, etc.

Dans ce papier, nous présentons l'élaboration d'un outil d'administration et de tuning permettant d'assister l'administrateur durant la conception physique et le tuning. Il doit lui permettre de choisir la ou les techniques d'optimisation qu'il souhaite utiliser, les algorithmes de sélection, les attributs et les tables participant à la définition des techniques d'optimisation. Vu le nombre important de techniques d'optimisation, nous avons concentré notre travail sur trois d'entre elles, à savoir la *fragmentation horizontale primaire, dérivée* et les *index de jointure binaires*. Ce choix est dû à la similarité entre ces deux techniques.

Ce papier est divisé en 6 sections. Dans la section 2, nous décrivons brièvement les deux types de fragmentation et leurs problèmes de sélection. La section 3 présente les index de jointure en étoile en décrivant leurs principes et leurs différents algorithmes de sélection. La section 4 montre la forte similarité entre les index et la fragmentation horizontale dérivée et l'exploitation de cette similarité comme un moyen de tuning dans les entrepôts de données. La section 5 détaille la conception et le développement de notre outil en montrant ses principales fonctionnalités. La section 6 conclut le papier en résumant le travail effectué et en citant quelques perspectives.

[1] Ce papier a été consacré le meilleur papier de VLDB'2007

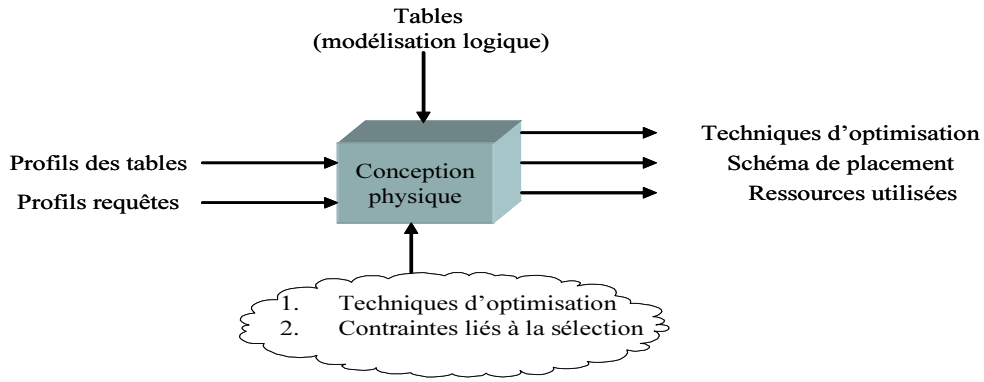


FIG. 1 – *La conception physique*

2 Fragmentation Horizontale

La fragmentation horizontale a été largement adoptée par la communauté des entrepôts de données relationnels. Elle permet de décomposer une table/vue matérialisée en plusieurs sous tables, appelées fragments horizontaux, où chacun contient un sous ensemble d'instances de la table globale. Deux types de fragmentation sont possibles : (i) la fragmentation primaire et (2) la fragmentation dérivée. Dans le premier type, une table est décomposée en fonction de ses attributs. La fragmentation dérivée consiste quant à elle à fragmenter une table en utilisant des attributs d'une autre table. Cette fragmentation est possible uniquement s'il y a un lien de jointure entre les deux tables. Les fragmentations primaire et dérivée peuvent être appliquées respectivement aux tables de dimension et à la table des faits (Bellatreche et Boukhalfa, 2005). Les deux types de fragmentation optimisent les requêtes de jointure en étoile caractérisées par des opérations de sélection définies sur les tables de dimension et des jointures entre les tables de dimension et la table des faits (Bellatreche et Boukhalfa, 2005). Appliquer la fragmentation dérivée sur la table des faits peut générer un nombre important de fragments (noté N) $N = \prod_{i=1}^{g} m_i$ où m_i et g sont respectivement le nombre de fragments de la table de dimension D_i et le nombre de tables de dimension qui ont participé dans le processus de fragmentation.

Le problème de sélection de schéma de fragmentation d'un entrepôt de données est formalisé de la manière suivante : étant donnée une charge de requêtes $Q = \{Q_1, ..., Q_m\}$ définie sur un entrepôt de données composé d'une table de faits F et d tables de dimension $\{D_1, D_2, ..., D_d\}$, et une contrainte W représentant le nombre de fragments de faits que l'administrateur souhaite avoir. Le problème de sélection de schéma de fragmentation horizontale consiste à fragmenter la table des faits en fonction des schémas de fragmentation des tables de dimension en N fragments, tel que ($\sum_{Q_j \in Q} f_j \times Cost(Q_j)$ soit minimisé et ($N \leq W$) (f_j représente la fréquence d'accès de la requête Q_j).

Pour résoudre ce problème, plusieurs algorithmes de sélection ont été proposés (Bellatreche et al., 2006; Boukhalfa et al., 2008) : un algorithme génétique, un algorithme de recuit simulé, un algorithme glouton et un algorithme tabou. Chaque algorithme génère un schéma de fragmentation de l'entrepôt de données. Étant donnés deux types de tables à fragmenter : les

tables de dimension et la table des faits, le schéma de fragmentation de l'entrepôt contient les schémas de fragmentation des tables de dimension et la table des faits. Pour réaliser cette fragmentation, l'administrateur doit identifier les attributs de dimension participant à ce processus, appelés les *attributs de fragmentation*. Un algorithme de fragmentation permet de partitionner le domaine de chaque attribut de fragmentation.

Exemple 1 *Supposons que nous ayons un entrepôt de données avec trois tables de dimension (TEMPS, CLIENT et PRODUIT) et une table des faits VENTES (voir Figure 2). Les attributs de fragmentation sont Ville de la table CLIENT, Catégorie de la table PRODUIT et Mois de la table TEMPS. Les domaines de chaque attributs sont :*
Dom(Ville) = {"Poitiers", "Paris", "Nantes"}, Dom(Catégorie) = {"Beauté", "Multimédia", "Jouet", "Jardinage", "Fitness"} et Dom(Mois) = {"Janvier", "Février", "Mars", "Avril","Mai", "Juin"}. Le processus de fragmentation établi un partitionnement de chaque domaine en sous domaines. Par exemple, si nous décidons de décomposer le domaine de Ville en deux sous domaines SD_1 et SD_2 tels que SD_1 = {"Poitiers", "Nantes"} et SD_2 = {"Paris"}, la table CLIENT est alors décomposée en deux fragments, où chacun correspond à un sous domaine.

Après cette présentation du problème de sélection d'un schéma de fragmentation, nous pouvons lister plusieurs problèmes que l'administrateur doit régler : (1) le choix des tables de dimension à utiliser pour fragmenter la table des faits, (2) la décomposition des domaines en sous domaines que les algorithmes de fragmentation exploitent, (3) le nombre de fragments finaux que l'administrateur souhaite avoir, (3) l'algorithme de fragmentation, et (4) les paramètres utilisés par chaque algorithme. Le tableau suivant récapitule les différents paramètres utilisés par les quatre algorithmes.

Algorithme	Paramètres
Algorithme génétique	Nombre d'individus Nombre de générations Taux de croisement Taux de mutation
Algorithme tabou	La taille de la liste tabou Nombre d'itérations
Recuit Simulé	Température initiale Gel du système Équilibre

3 Index de Jointure Binaires

L'index de jointure binaire est une variante de l'index de jointure en étoile. Étant donné que les requêtes de jointure en étoile possèdent des opérations de jointure suivies par des opérations de sélection, un bitmap représentant les n-uplets de la table de faits est créé pour chaque valeur distincte de l'attribut de la table dimension sur lequel l'index est construit. Le i^{me} bit du bitmap est égal à 1 si le n-uplet correspondant à la valeur de l'attribut indexé peut être joint avec le n-uplet de rang i de la table de faits. Dans le cas contraire, le i^{me} bit est à zéro. Les index de jointure binaires sont efficaces pour les requêtes de type COUNT, AND, OR, NOT, d'où leur implémentation dans les SGBDs commerciaux, comme Oracle, SQL server, et DB2. Notons

que les index binaires sont recommandés pour des attributs de faible cardinalité comme sexe. Notons que dans un contexte décisionnel, les requêtes d'analyse se font généralement sur des indicateurs (attributs) dont le domaine est restreint. Une autre caractéristique offerte par les index de jointure binaires est la compression (Johnson, 1999), ce qui permet une réduction de leur espace de stockage et la possibilité de les stocker en mémoire centrale.

Notons qu'un index de jointure binaire dans les entrepôts de données relationnels peut être défini sur un ou plusieurs attributs non clés de faible cardinalité des tables de dimension. Ces attributs sont appelés les *attributs indexables*. Sur un attribut indexable A_j d'une table de dimension D_i une condition de la forme suivante $D_i.A_j \ \theta \ Valeur$ est définie dans la clause WHERE où $\theta = \{<, >, \leq, \geq, =\}$ et Valeur $\in$ domaine de l'attribut A_j.

Soit $A = \{A_1, A_2, ..., A_K\}$ l'ensemble des attributs indexables candidats pour les index de jointure binaires. Le nombre d'index de jointure possible ayant un seul attribut, est donné par :

$$\binom{K}{1} + \binom{K}{2} + ... + \binom{K}{K} = 2^K - 1 \quad (1)$$

Pour $K = 4$, ce nombre est égale à 15. Le nombre possible de tous les index est donné par :

$$\binom{2^K-1}{1} + \binom{2^K-1}{2} + ... + \binom{2^K-1}{2^K-1} = 2^{2^K-1} \quad (2)$$

Pour $K = 4$, ce nombre est (2^{15}). En conséquence, sélectionner un ensemble d'index de jointure binaires minimisant le coût total d'exécution de requêtes et satisfaisant une contrainte de stockage est formalisé de la manière suivante. Étant donné un entrepôt de données ayant un ensemble de tables de dimension $D = \{D_1, D_2, ..., D_d\}$ et une table des faits F, une charge de requêtes $Q = \{Q_1, Q_2, ..., Q_m\}$, où chaque requête Q_j a une fréquence d'accès f_j, et une capacité de stockage S, le problème de sélection des index de jointure consiste à trouver un ensemble d'index minimisant le coût d'exécution de requêtes et satisfaisant la contrainte de stockage S.

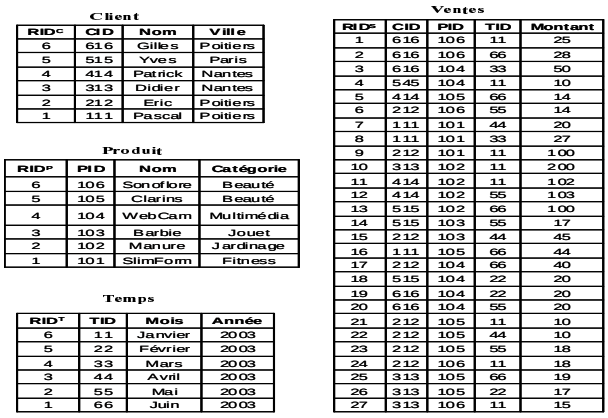
Deux types d'algorithmes ont été utilisés pour sélectionner des index de jointure en étoile : algorithmes gloutons (Bellatreche et al., 2007) et algorithmes dirigés par les techniques de fouille de données (Aouiche et al., 2005; Bellatreche et al., 2008).

4 Utilisation des Index de Jointure Binaires et la Fragmentation Horizontale pour le Tuning des Entrepôts de données

Dans cette section, nous montrons comment les index de jointure binaires et la fragmentation horizontale peuvent être utilisés pour tuner un entrepôt de données. Cela est possible grâce leur similarité. Pour illustrer cette similarité, nous considérons l'exemple suivant : Considérons l'entrepôt décrit dans l'exemple 1. La population de ce schéma est décrite dans la Figure 2. Sur ce schéma, considérons la requête suivante :

```
SELECT Count(*)
FROM CLIENT C, PRODUIT P, TEMPS T, VENTES V
WHEERE C.Ville='Poitiers'
AND P.Catégorie='Beauté'
AND T.Mois='Juin'
AND P.PID = V.PID AND C.CID=C.CID AND T.TID=V.TID
```

Cette requête contient trois prédicats de sélection définis sur les attributs suivants : *Ville, Catégorie* et *Mois* et trois jointures. Ces attributs sont candidats pour la définition de schéma de fragmentation et la définition des index de jointure binaires. Pour optimiser cette requête, trois modes d'exécution sont possibles pour l'administrateur : (i) l'utilisation de la fragmentation horizontale seule (FHSEULE), (ii) l'utilisation des index de jointure seuls (IJBSEUL) et (iii) l'utilisation mixte (FH&IJB) qui combine les deux modes.



Client

RID^C	CID	Nom	Ville
6	616	Gilles	Poitiers
5	515	Yves	Paris
4	414	Patrick	Nantes
3	313	Didier	Nantes
2	212	Eric	Poitiers
1	111	Pascal	Poitiers

Produit

RID^P	PID	Nom	Catégorie
6	106	Sonoflore	Beauté
5	105	Clarins	Beauté
4	104	WebCam	Multimédia
3	103	Barbie	Jouet
2	102	Manure	Jardinage
1	101	SlimForm	Fitness

Temps

RID^T	TID	Mois	Année
6	11	Janvier	2003
5	22	Février	2003
4	33	Mars	2003
3	44	Avril	2003
2	55	Mai	2003
1	66	Juin	2003

Ventes

RID^S	CID	PID	TID	Montant
1	616	106	11	25
2	616	106	66	28
3	616	104	33	50
4	545	104	11	10
5	414	105	66	14
6	212	106	55	14
7	111	101	44	20
8	111	101	33	27
9	212	101	11	100
10	313	102	11	200
11	414	102	11	102
12	414	102	55	103
13	515	102	66	100
14	515	103	55	17
15	212	103	44	45
16	111	105	66	44
17	212	104	66	40
18	515	104	22	20
19	616	104	22	20
20	616	104	55	20
21	212	105	11	10
22	212	105	44	10
23	212	105	55	18
24	212	106	11	18
25	313	105	66	19
26	313	105	22	17
27	313	106	11	15

FIG. 2 – *Un exemple de Population de l'Entrepôt*

FHSEULE : L'administrateur peut décomposer la table des faits VENTES en se basant sur les schémas de fragmentation des trois tables de dimension : CLIENT, TEMPS et PRODUIT fragmentées en utilisant *Ville*, *Mois*, *Catégorie*, respectivement. En conséquence, la table des faits est partitionnée en 90 fragments [2], où chaque fragment des faits $Ventes_i$ $(1 \leq i \leq 90)$ est défini comme suit :

$$Ventes\_i = VENTES \ltimes CLIENT_j \ltimes TEMPS_k \ltimes PRODUIT_m \quad ,$$

avec $(1 \leq j \leq 3)$, $(1 \leq k \leq 6)$, $(1 \leq m \leq 5)$, et $\ltimes$ représente la semi-jointure.
La Figure 3 présente le sous schéma en étoile (VENTES_BJP) correspondant aux achats de produits de beauté réalisées par les clients vivant à Poitiers durant le mois de Juin. En conséquence, la requête ci-dessus est réécrite comme suit :
SELECT Count(*) FROM VENTES ***PARTITION****(VENTES_BJP)* [3]
Pour exécuter cette requête, l'optimiseur ne charge en mémoire que le fragment des faits Ventes_BJP (Figure 3). Ce mode permet d'éviter les trois opérations de jointure ce qui représente un gain considérable.

IJBSEUL : L'administrateur peut sélectionner un index de jointure binaire entre la table des faits et les trois tables de dimension sur les attributs *Ville, Mois* et *Catégorie* comme suit :

[2] Supposons que nous avons 3 villes, 6 mois et 5 catégories.
[3] Nous utilisons une syntaxe d'Oracle.

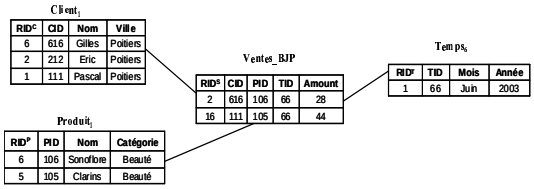


FIG. 3 – *Sous Schéma en Étoile VENTES_BJP*

```
CREATE BITMAP INDEX vente_client_ville_prod_cat_time_mois_bjix
ON VENTES(CLIENT.Ville, PRODUIT.Catégorie, TEMPS.Mois)
FROM VENTES V, CLIENT C, TEMPS T, PRODUIT P
WHERE V.CID = C.CID AND V.PID = P.PID AND V.TID = T.TID
```

Pour exécuter la requête, l'optimiseur doit accéder aux bitmaps (colorés en gris dans la Figure 4a) qui correspondent aux valeurs de trois attributs d'indexation puis il réalise une opération logique "AND" entre ces bitmaps (Figure 4b). De ce fait, aucune opération de jointure n'est effectuée pour exécuter la requête.

	Ville			Mois						Catégorie					
RID	P	Pr	N	Ja	Fe	Ma	Av	Mai	Ju	B	M	Jo	Ja	F	AND
1	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0
2	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1
3	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0
4	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0
5	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0
6	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0
7	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0
8	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0
9	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0
10	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0
11	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0
12	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0
13	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0
14	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0
15	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0
16	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1
17	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0
18	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0
19	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0
20	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0
21	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0
22	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0
23	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0
24	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0
25	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0
26	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0
27	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0

(a) (b)

FIG. 4 – *(a) Index de jointure généré (b) Opération AND*

FH&IJB : Au lieu d'utiliser chaque technique séparément, nous proposons de les combiner en prenant en compte leur similitude. Cette opération se déroule en quatre étapes :

1. Déterminer l'ensemble des attributs de dimension utilisés dans les prédicats de sélection des requêtes (cet ensemble est noté AD).
2. Fragmenter l'entrepôt de données en utilisant AD et l'ensemble de requêtes de départ : $Q = \{Q_1, Q_2, \cdots, Q_m\}$. Cette fragmentation peut être réalisée avec n'importe quel algorithme de partitionnement (glouton, recuit simulé, génétique, tabou, etc.). Le fait que nous sommes contraints par le seuil de maintenance W, il est fort possible que

le schéma de fragmentation obtenu ne couvre pas tous les attributs de AD. On note par $AFSET$ ($AFSET \subseteq AD$) l'ensemble des attributs de dimension participant à la définition des fragments.

3. Identifier les requêtes bénéficiaires de la fragmentation horizontale : le schéma de fragmentation pourrait être bénéfique pour certaines requêtes et non bénéfique pour d'autres. Pour déterminer les requêtes bénéficiaires, nous définissons une métrique (appelée, métrique de tuning) pour chaque requête Q_j :

$$taux(Q_j) = \frac{C[Q_j, SF]}{C[Q_j, \phi]} \quad (3)$$

où $C[Q_j, SF]$ et $C[Q_j, \phi]$ représentent le coût d'exécution de la requête Q_j respectivement sur un schéma partitionné SF et non partitionné. L'administrateur a le droit de paramétrer cette métrique en utilisant un seuil λ défini comme suit :
Si $taux(Q_j) \leq \lambda$ alors la requête Q_j est bénéficiaire, sinon elle est non bénéficiaire.

4. Identifier les attributs indexables candidats : ce sont des attributs de faible cardinalité utilisés par les requêtes non bénéficiaires $Q^{nobenefit} = \{Q'_1, Q'_2, \cdots, Q'_l\}$ et n'appartiennent pas à $AFSET$.

Exemple 2 *Pour mieux comprendre HF&IJB, nous considérons le scénario suivant (basé sur l'exemple 1), où la FHSEULE a généré 90 fragments de la table des faits. Supposons que l'administrateur souhaite n'avoir que 18 fragments au lieu de 90 générés ($W = 18$). Dans ce cas, le schéma de fragmentation obtenu est défini sur deux attributs de dimension, à savoir Ville et Mois ($AFSET = \{Mois, Ville\}$). Si la requête initiale est bénéficiaire alors aucun index de jointure n'est défini. Dans le cas contraire, un index de jointure sera défini sur l'attribut Catégorie de la table de dimension PRODUIT, car il n'est pas pris en compte dans le processus de fragmentation. Cet index de jointure sera défini sur chaque sous schéma en étoile (Figure 5).*

Ventes_PJ

RIDs	CID	PID	TID	Montant
2	616	106	66	28
16	111	105	66	44
17	212	104	66	40

BJI_CAT

	Catégorie				
RID	B	M	J	Jr	F
2	1	0	0	0	0
16	1	0	0	0	0
17	0	1	0	0	0

FIG. 5 – *Index de Jointure défini sur un Sous Schéma en étoile*

5 Mise en Oeuvre de l'Outil d'Assistance

Dans cette section, nous proposons un outil d'assistance à l'administration et de tuning, appelé *ParAdmin*, qui permet à l'administrateur d'établir ses choix et de paramétrer les différents algorithmes. *ParAdmin* gère principalement trois techniques d'optimisation[4] ; la fragmentation

[4] *ParAdmin* est actuellement en extension pour inclure les vues matérialisées

horizontale primaire ; la fragmentation horizontale dérivée et les index de jointure binaires. Les objectifs principaux de *ParAdmin* sont :

- Permettre la visualisation de l'état courant de l'entrepôt de données : la structure de l'entrepôt (schéma, attributs, taille de chaque table, définition de chaque attribut, etc.) et la charge définie sur l'entrepôt (le nombre de sélections, le nombre de jointures, attributs de sélection, les facteurs de sélectivité de chaque prédicats, etc.)
- Offrir deux modes de sélection de techniques d'optimisation : *personnalisée* et *non personnalisée*. Dans la sélection non personnalisée, nous supposons que l'administrateur n'a pas assez de connaissance sur la technique sélectionnée. L'outil fait alors le choix (par défaut) de l'algorithme de sélection et propose " un meilleur schéma ". Tandis que la sélection personnalisée donne plus de liberté à l'administrateur pour le choix de l'algorithme, des paramètres, des attributs, et des tables sur lesquels il souhaite sélectionner la technique d'optimisation.
- Offrir une sélection mono-structure de son choix.
- Offrir une sélection multi-structure.
- Permettre à l'administrateur de revoir ses choix s'il n'est pas satisfait.
- Proposer la génération des scripts implémentant les techniques d'optimisation.
- Permettre la visualisation de la qualité de chaque technique d'optimisation proposée et de ses critères de performance : le coût d'entrées/sorties de requêtes, le taux d'amélioration apporté par la technique, etc.

5.1 La Conception de ParAdmin

La principale difficulté pour concevoir notre outil est l'identification des différents besoins et leurs représentations. Pour ce faire, nous avons utilisé un formalisme de modèle de tâches (Balbo et al., 2004). Ceux-ci permettent d'une part d'analyser les systèmes interactifs en mettant leur utilisation au premier plan et d'autre part d'exprimer les différentes activités qu'un utilisateur souhaite pouvoir réaliser. Les outils développés pour les supporter intègrent le plus souvent un outil de simulation qui produit les scénarii. Ces derniers sont utilisés pour permettre la validation par l'utilisateur de l'ordonnancement des tâches du logiciel en conception. Parmi les différents formalismes de modèle de tâches qui ont été développés et qui disposent d'un outil logiciel support citons CTT (CTTE) (Paterno et al., 2001) et K-MAD (K-MADe) [5]. Nous avons utilisé ce dernier principalement parce qu'il est développé en partenariat avec l'équipe IHM de notre laboratoire et l'INRIA. De plus, il permet la définition de conditions logiques (pré, post, itération) formelles prises en compte lors de la simulation augmentant de ce fait le niveau de vérification.

La modélisation des tâches nous a permis d'identifier les différents besoins d'utilisation de *ParAdmin*. Nous avons validé l'ordonnancement des tâches à l'aide des scénarii afin de produire une interface permettant de les accomplir selon les besoins de l'administrateur. Ces modèles nous ont permis d'obtenir l'architecture globale de *ParAdmin* (voir la Figure 6).

[5] http ://kmade.sourceforge.net

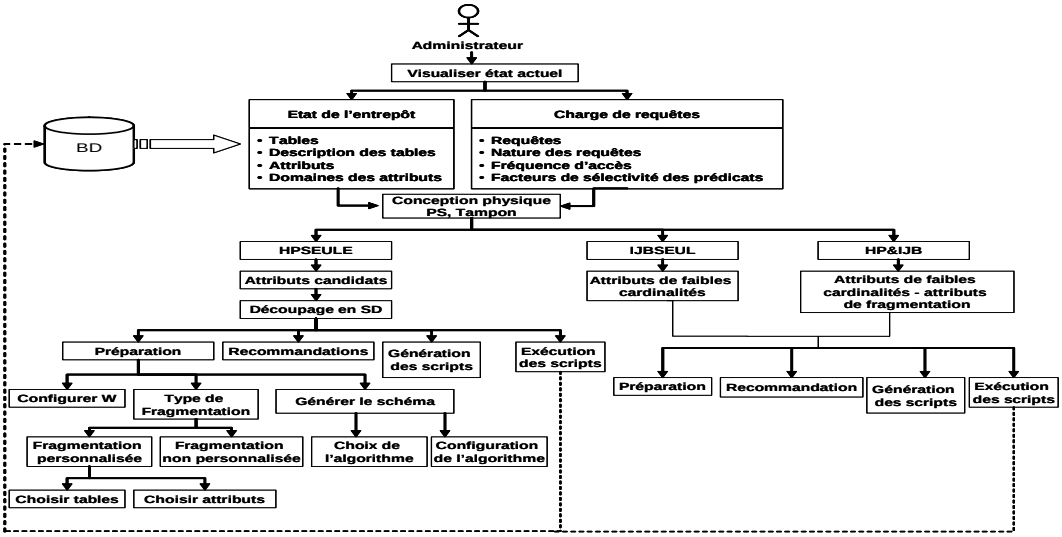


FIG. 6 – *Architecture générale du fonctionnement de ParAdmin*

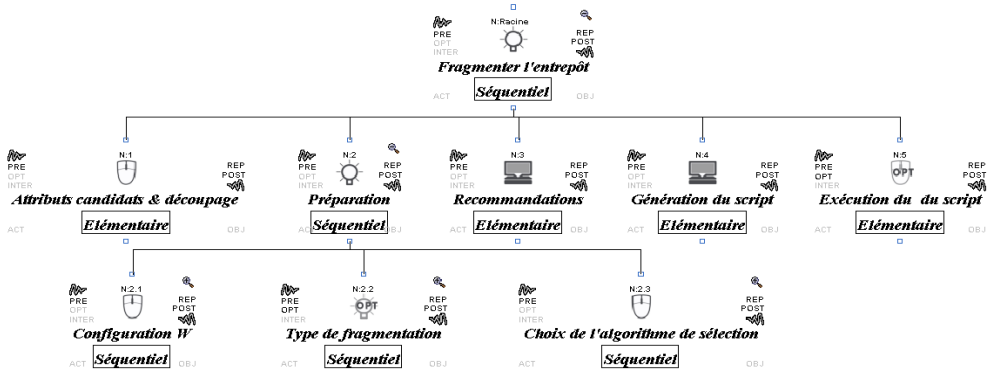


FIG. 7 – *Modèle de tâche de la fragmentation de l'entrepôt*

5.2 Analyse des Tâches de ParAdmin

Pour concevoir notre outil, nous avons identifié trois principales tâches non élémentaires que l'administrateur doit pouvoir effectuer : (1) visualiser l'état courant de l'entrepôt de données, (2) partitionner l'entrepôt de données, et (3) l'indexer.

5.2.1 La visualisation de l'état de l'entrepôt

Cette tâche permet à l'administrateur de visualiser deux sortes d'informations relatives à l'entrepôt de données (les tables, leurs descriptions, la définition de chaque attribut de dimension ou de faits, le domaine de chaque attribut) et à la charge des requêtes (la nature de la requête : recherche ou mise à jour, la fréquence d'accès de chaque requête, les différents facteurs de sélectivité des prédicats de sélection, etc.). La Figure 8 montre l'interface correspondante à la réalisation de cette tâche.

5.2.2 Le partitionnement de l'entrepôt

La fragmentation de l'entrepôt est une étape du processus de conception physique. La Figure 7 en présente la modélisation sous forme d'un arbre de tâches K-MAD. Deux types de fragmentation sont supportés dans les entrepôts de données, la fragmentation primaire (appliquée aux tables de dimension) et la fragmentation dérivée (appliquée à la table des faits), c'est pourquoi l'administrateur doit préalablement identifier les attributs non clés des tables de dimension candidats au processus de fragmentation. Une fois cette identification établie, il propose un découpage de leurs domaines en sous domaines. Un paramètre important que l'administrateur doit configurer est la contrainte de maintenance. Celle-ci représente le nombre de fragments de la table des faits généré (W). Pour rendre cette fragmentation opérationnelle,

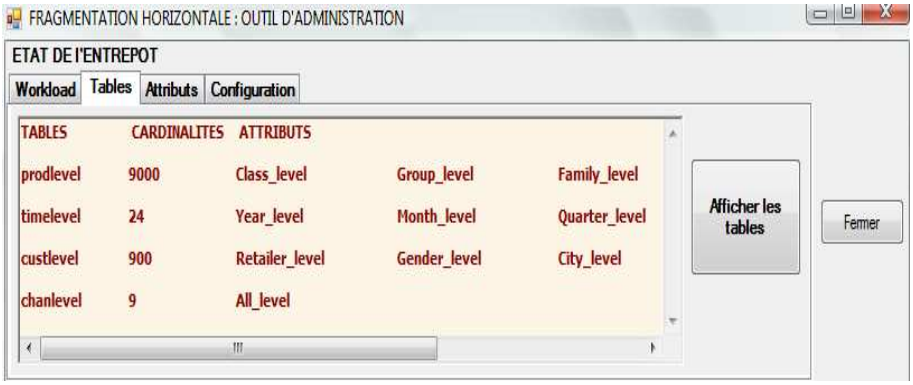


FIG. 8 – *Visualisation de l'état de l'entrepôt*

deux types de partitionnement sont disponibles : (a) la fragmentation non personnalisée et (b) la fragmentation personnalisée. Dans le premier type, *ParAdmin* partitionne l'entrepôt de données en utilisant tous les attributs candidats et un algorithme de fragmentation par défaut. Le deuxième type *offre plus de liberté* à l'administrateur dans le processus de sélection. Il peut

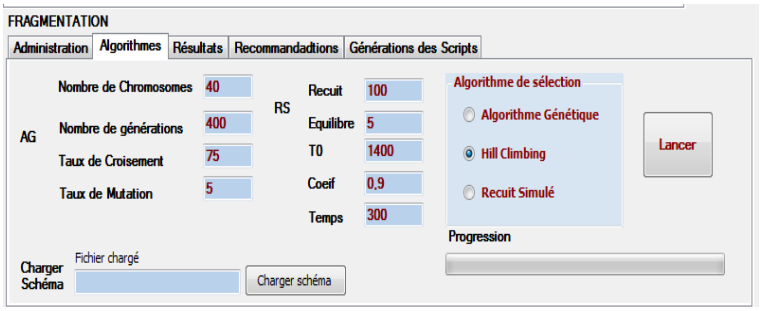


FIG. 9 – *Choix et configuration des algorithmes*

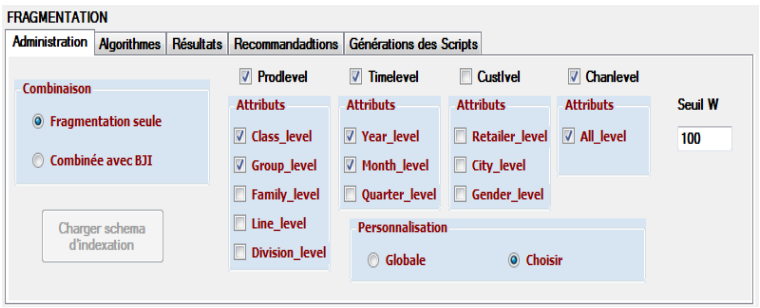


FIG. 10 – *Personnalisation de la fragmentation*

choisir les tables et les attributs de dimension participant au processus de fragmentation. Il doit choisir l'algorithme de partitionnement et établir ses paramètres. Cette obligation de choix de l'algorithme a été formalisée à l'aide des conditions formelles proposées dans K-MADe par une pré-condition logique. Trois types d'algorithme sont disponibles : un algorithme glouton, un algorithme génétique et un algorithme de recuit simulée (Figure 9). Une fois le schéma de l'entrepôt fragmenté, l'administrateur peut visualiser une recommandation proposée par *ParAdmin*. Elle contient le nombre de sous schémas en étoile générés, le découpage final du domaine de chaque attribut de sélection, une estimation du nombre d'entrées et sorties nécessaires pour exécuter la charge de requêtes, le nombre de fragments de chaque table de dimension, le gain de performance obtenu par cette fragmentation (par rapport à un schéma non fragmenté), les attributs de fragmentation, etc. Si l'administrateur n'est pas satisfait de cette recommandation, il peut revenir à l'étape précédente et changer les différents paramètres (rechoisir les attributs et les tables ou l'algorithme, ses paramètres, etc.). Ce retour en arrière est primordial dans la phase de conception physique. Une fois satisfait, l'administrateur demande à l'outil de générer les scripts de fragmentation. Ceux-ci pourront par la suite être appliqués sur l'entrepôt de données. La Figure 10 montre l'interface d'une fragmentation personnalisée.

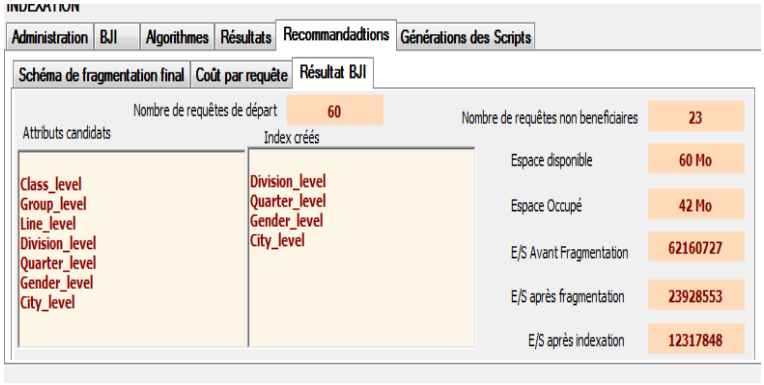


FIG. 11 – *Recommandation d'indexation*

5.2.3 La tâche d'indexation

Elle permet de sélectionner des index de jointure en étoile. Cette tâche peut être exécutée d'une manière isolée (IJBSEUL) ou combinée (FH&IJB). Dans le cas d'une IJBSEUL, l'administrateur doit d'abord choisir les attributs indexables candidats. Le paramètre important pour l'indexation est la contrainte de stockage. Comme pour la fragmentation horizontale, deux types d'indexation sont possibles : *l'indexation non personnalisée* et *l'indexation personnalisée*. Cette tâche est quasi similaire à la précédente. Dans la recommandation, on trouve les informations concernant le gain des index, les attributs indexés, le coût de stockage exigé par les index sélectionnés, etc. La Figure 11 présente l'interface dédiée à la tâche de recommandation sur les index (l'espace occupé, l'espace disponible, le nombre de requêtes non bénéficiaires, etc.).

FH&IJB a le même principe que IJBSEUL sauf pour le choix des attributs candidats à l'indexation (au lieu de considérer les attributs candidats à partir de la configuration initiale de l'entrepôt de données, l'administrateur doit identifier ces attributs parmi les attributs d'indexation non utilisés par HFSEULE) et le choix du paramètre du tuning (λ). Ce paramètre peut jouer un rôle important dans le tuning, par exemple, si l'évolution de l'entrepôt de données impose à l'administrateur de réduire le coût de stockage de structures d'optimisation redondantes. Il est donc nécessaire de pouvoir paramétrer λ pour que la plupart des requêtes soient bénéficiaires.

6 Conclusion

Les entrepôts de données ont fait naître un nouveau besoin d'administration et de tuning. Cela est dû à leurs caractéristiques : la volumétrie, la complexité des requêtes, les exigences de temps de réponse raisonnable et la gestion de l'évolution de l'entrepôt. Dans cet environnent, nous avons mis en évidence les difficultés qu'un administrateur pourrait rencontrer durant les phases de conception physique et de tuning. Ces difficultés sont multiples car elles concernent plusieurs niveaux de conception : le choix des techniques d'optimisation pertinentes pour l'en-

semble de requêtes à optimiser, le choix de la nature de sélection des techniques d'optimisation et le choix des algorithmes et leur paramètres. Vu ces difficultés, nous avons identifié le besoin de développer un outil d'assistance à l'administrateur qui permet de répondre aux besoins en terme de choix possibles. Nous avons proposé un outil, appelé *ParAdmin* offrant trois techniques d'optimisation : la fragmentation horizontale primaire, dérivée et les index de jointure binaires. Il permet à l'administrateur de choisir les différents algorithmes et leurs paramètres. Il peut alors utiliser ces techniques d'une manière isolée ou combinée. Une autre particularité de *ParAdmin* est le fait de proposer des sélections personnalisées et non personnalisées des structures d'optimisation. Après chaque sélection, *ParAdmin* propose des recommandations à l'administrateur permettant de visualiser la qualité de la ou les techniques qu'il a choisie(s). Il peut alors soit valider ses choix en générant et exécutant les scripts ou les revoir.

De plus, notre outil peut être pluggé sur n'importe quelle base de données ou entrepôt de données et peut facilement évoluer. Cette évolution peut concerner : l'ajout ou la suppression d'un algorithme de sélection, l'ajout ou la suppression d'une technique d'optimisation comme les vues matérialisées.

Références

Aouiche, K., O. Boussaid, et F. Bentayeb (2005). Automatic Selection of Bitmap Join Indexes in Data Warehouses. *7th International Conference on Data Warehousing and Knowledge Discovery (DAWAK 05)*.

Balbo, S. O., N., et C. Paris (2004). Choosing the right task-modeling notation : A taxonomy. *In The Handbook of Task Analysis for Human-Computer Interaction, D. Diaper and N. Stanton (Eds.), Lawrence Erlbaum Associates (LEA)*.

Bellatreche, L. et K. Boukhalfa (2005). An evolutionary approach to schema partitioning selection in a data warehouse environment. *Proceeding of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK'2005)*, 115–125.

Bellatreche, L., K. Boukhalfa, et H. I. Abdalla (2006). Saga : A combination of genetic and simulated annealing algorithms for physical data warehouse design. *in 23rd British National Conference on Databases* (212-219).

Bellatreche, L., K. Boukhalfa, et M. K. Mohania (2007). Pruning search space of physical database design.

Bellatreche, L., R. Missaoui, H. Necir, et H. Drias (2008). A data mining approach for selecting bitmap join indices. *Journal of Computing Science and Engineering 2*(1), 206–223.

Boukhalfa, K., L. Bellatreche, et P. Richard (2008). ragmentation primaire et dérivée : Étude de complexité, algorithmes de sélection et validation sous oracle10g. Techreport http ://www.lisi.ensma.fr/members/bellatreche, LISI/ENSMA.

Chaudhuri, S. (2004). Index selection for databases : A hardness study and a principled heuristic solution. *IEEE Transactions on Knowledge and Data Engineering 16*(11), 1313–1323.

Chaudhuri, S. et V. Narasayya. Self-tuning database systems : A decade of progress. *Proceedings of the International Conference on Very Large Databases*.

Chee-Yong, C. (1999). Indexing techniques in decision support systems. Phd. thesis, University of Wisconsin - Madison.

Gill, P. J. (2000). Breaking the warehouse barrier. *Oracle Magazine 1*(IX), 38–44.

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

Johnson, T. (1999). Performance measurements of compressed bitmap indices. *Proceedings of the International Conference on Very Large Databases*, 278–289.

Kraft, T., H. Schwarz, et B. Mitschang (2007). A statistics propagation approach to enable cost-based optimization of statement sequences. pp. 267–282.

Labio, W., D. Quass, et B. Adelberg (1997). Physical database design for data warehouses. *Proceedings of the International Conference on Data Engineering (ICDE)*.

O'Neil, P. et D. Quass (1997). Improved query performance with variant indexes. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 38–49.

Paterno, F., G. Mori, et R. Galimberti (2001). *CTTE : An Environment for Analysis and Development of Task Models of Cooperative Applications*. ACM Press.

Sanjay, A., C. Surajit, et V. R. Narasayya (2000). Automated selection of materialized views and indexes in microsoft sql server. *Proceedings of the International Conference on Very Large Databases*, 496–505.

Sheet, O. D. (2007). Oracle partitioning. *White Paper : http ://www.oracle.com/technology/products/bi/db/11g/*.

Talebi, Z. A., R. Chirkova, Y. Fathi, et M. Stallmann (2008). Exact and inexact methods for selecting views and indexes for olap performance improvement. *To appear in 11th International Conference on Extending Database Technology (EDBT'08)*.

Valentin, G., M. Zuliani, D. C. Zilio, G. M. Lohman, et A. Skelley (2000). Db2 advisor : An optimizer smart enough to recommend its own indexes. *ICDEŠ00*, 101–110.

Summary

The task of administrating data warehouses becomes a crucial issue compare to traditional databases. This is due to the characteristics of data warehouses: large amount of data, complex queries, satisfaction of decision maker requirement and management of data evolution. In the context of data warehouse, a large spectrum of optimisation techniques was proposed during the physical database design. Each technique has a number of selection algorithms, where each one has its own parameters. Another interesting point is the strong similarity between optimisation techniques like for materialized views and indexes. During the physical design, the data warehouse administrator shall establish several choices. In this paper, we firstly, show difficulties that an administrator has during its task of administrating and tuning a warehouse. Secondly, we propose a tuning method based on horizontal partitioning and bitmap join indexes. Finally, we propose a tool advising the administrator in an interactive way to realise its tasks. They concern the choice of optimisation techniques, selection algorithms, their parameters and visualisation of performance recommendation.