Maintenance de charge pour l'optimisation des entrepôts de données évolutifs : aide à l'administrateur

Cécile Favre, Fadila Bentayeb, Omar Boussaid

Université de Lyon (Laboratoire ERIC - Lyon 2)
5 av. Pierre Mendès-France, 69676 Bron Cedex
{cfavre|bentayeb}@eric.univ-lyon2.fr, omar.boussaid@univ-lyon2.fr
http://eric.univ-lyon2.fr

Résumé. Dans un contexte où les entrepôt de données sont amenés à subir des évolutions, nous proposons d'aider l'administrateur à la maintenance de la charge (ensemble de requêtes) qui sert à l'évaluation des performances. En répercutant les évolutions de l'entrepôt de données sur la charge, il est alors possible pour l'administrateur d'avoir une gestion pro-active des performances, évitant d'attendre que l'entrepôt mis à jour soit interrogé par les utilisateurs pour construire une charge valide pour optimiser les performances.

Mots-clés : Entrepôt de données, modèle, évolution, optimisation, maintenance de charge, requête décisionnelle, administrateur.

1 Introduction

Les entrepôts de données présentent une modélisation multidimensionnelle définie en fonction des sources de données et des besoins d'analyse. L'évolution de ces deux derniers «paramètres» peut alors nécessiter une évolution du modèle multidimensionnel, définissant ainsi un contexte évolutif.

Parallèlement, un des objectifs principaux de la constitution des entrepôts de données est l'analyse dite «en ligne» (OLAP : On-Line Analytical Processing). Or, Krompass et al. (2007) ont précisé que, dans le contexte des entrepôts de données, les requêtes peuvent être très rapides (quelques minutes) mais peuvent également prendre plusieurs heures. Il est alors crucial pour l'administrateur de pouvoir gérer la performance de l'entrepôt, en particulier dans le contexte évolutif. Se pose alors le problème de l'évaluation de ces performances dans ce contexte évolutif. Classiquement la gestion des performances doit se faire en fonction de l'utilisation de l'entrepôt. Ainsi, l'évaluation de performances se base sur un ensemble de requêtes utilisateurs, appelé charge.

Dans un contexte évolutif, nous pensons qu'il est alors intéressant pour l'administrateur d'avoir une démarche pro-active en matière de gestion de performances, et de ne pas attendre que l'entrepôt de données mis à jour soit utilisé par les utilisateurs pour obtenir une nouvelle charge de requêtes cohérente avec le modèle courant. Précédemment, nous avons proposé une première approche simpliste d'évolution de charge automatique (Favre et al., 2007). Dans cet article, nous nous focalisons davantage sur l'aide à l'administration que nous pouvons apporter pour la maintenance de la charge. L'objectif est d'aider l'administrateur d'une part à maintenir

les requêtes de la charge consistantes vis-à-vis de l'évolution du modèle de l'entrepôt et d'autre part à créer, si cela est intéressant, de nouvelles requêtes afin d'exprimer des analyses portant sur des possibilités d'analyse nouvellement définies par l'évolution du modèle.

Cet article est organisé comme suit. Tout d'abord, nous présentons dans la Section 2 un état de l'art qui porte sur l'évaluation et l'optimisation de performances dans les entrepôts d'une part, sur l'évolution de charge d'autre part. Ensuite, nous présentons notre approche de support pour l'évolution de charge dans la Section 3. Par la suite, nous illustrons notre approche sur l'étude de cas de LCL-Le Crédit Lyonnais dans la Section 4. Enfin, nous concluons cet article dans la Section 5.

2 État de l'art

Les entrepôts de données contenant un large volume de données, il est nécessaire d'avoir recours à des méthodes efficaces d'accès aux données pour répondre aux requêtes de manière optimale. Une alternative possible est d'utiliser des structures redondantes. En effet, parmi les techniques issues des implémentations relationnelles des entrepôts de données, la matérialisation de vues et l'indexation sont très efficaces pour améliorer le temps de réponse des requêtes décisionnelles (Rizzi et Saltarelli, 2003). Ainsi, un des aspects les plus importants dans la conception physique de l'entrepôt est justement de sélectionner un ensemble approprié de vues à matérialiser et d'index, qui permet de minimiser le temps de réponse des requêtes, compte-tenu d'un espace de stockage limité (Bellatreche, 2000). Pour résoudre cet aspect, un choix judicieux doit être basé sur le coût et guidé par les requêtes grâce à une extraction des candidats (index-vues) à partir d'une analyse syntaxique des requêtes (Agrawal et al., 2000). Il est crucial d'adapter la performance du système en fonction de son utilisation, la charge est alors supposée représenter les requêtes des utilisateurs sur l'entrepôt de données. La plupart des approches proposées dans la littérature sont basées sur l'idée qu'il existe une charge de référence qui représente les besoins cibles pour l'optimisation. Cependant, Golfarelli et Saltarelli (2003) ont mis en avant que les charges réelles sont beaucoup plus volumineuses que celles qui peuvent être prises en compte par les techniques de sélection d'index et de vues à matérialiser. Ainsi, le succès de la matérialisation de vues et de l'indexation dépend dans la réalité de l'expérience de l'administrateur.

L'évolution du modèle de l'entrepôt nécessite non seulement une évolution des structures redondantes dans le cas de la maintenance de vues matérialisées comme l'ont proposé Hurtado et al. (1999), mais aussi une évolution de la stratégie d'optimisation elle-même : la configuration d'index et de vues initialement choisie peut être amenée à évoluer. En matière de sélection de vues, différents travaux se sont alors intéressés à la sélection dynamique qui se base sur une évolution de la charge, adoptant différents points de vue sur cette dernière. Lawrence et Rau-Chaplin (2006) considèrent que la sélection de vues doit être réalisée à des intervalles de maintenance réguliers et se basent sur un changement observé ou attendu des fréquences de requêtes. Kotidis et Roussopoulos (1999), Theodoratos et Sellis (2000) supposent que des requêtes sont ajoutées à la charge initiale. Ainsi, ces différents travaux supposent que les requêtes de la charge initiale sont toujours consistantes. Or nous affirmons que ce n'est pas toujours le cas. En effet, un changement dans le modèle de l'entrepôt peut en effet rendre les requêtes inconsistantes. Dès lors, l'évolution de requêtes doit être envisagée.

3 Évolution de charge : notre approche

3.1 Cadre général

Notre proposition d'évolution de charge se place dans un cadre général qui permet d'illustrer l'intérêt de la maintenance de charge pour une gestion pro-active des performances (Figure 1). À partir de l'utilisation de l'entrepôt de données, une charge initiale contenant les requêtes des utilisateurs peut être définie. Une stratégie d'optimisation peut être choisie sur la base de cette charge (choix d'index et de vues à matérialiser). Lorsque le modèle de l'entrepôt de données évolue, les requêtes peuvent devenir inconsistantes. En prenant en compte à la fois la charge initiale et les changements appliqués sur le modèle de l'entrepôt de données, le processus d'aide à l'évolution de la charge de requêtes est appliqué. La charge contient alors les requêtes de la charge initiale qui ont été mises à jour pour rester consistantes par rapport au modèle courant de l'entrepôt, mais également des nouvelles requêtes dans le cas où les changements opérés ont généré de nouvelles possibilités d'analyse dans l'entrepôt de données. Ainsi, il est inutile d'attendre l'interrogation des utilisateurs de l'entrepôt mis à jour pour définir une nouvelle charge utilisée dans la sélection des structures d'optimisation.

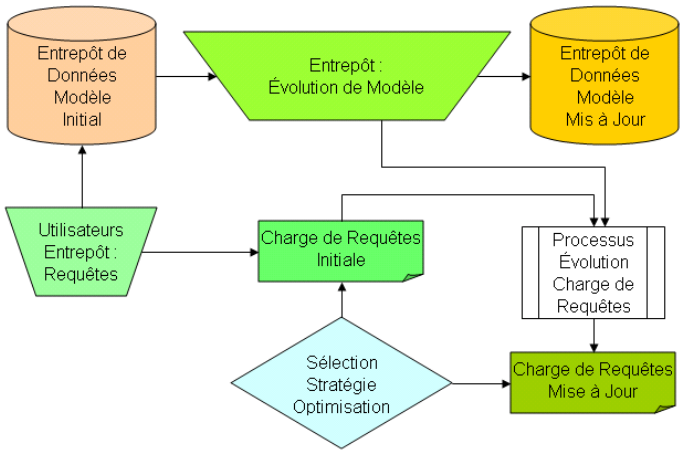

FIG. 1 – *Architecture pour l'évolution de la charge.*

Afin d'aider l'administrateur dans sa tâche, il est nécessaire d'être en mesure de lui indiquer quel type de modification il doit opérer sur les requêtes en fonction des modifications subies par l'entrêpôt. Nous nous basons alors sur la typologie que nous avons définie dans (Favre et al., 2007) pour représenter les changements s'opérant sur le schéma de l'entrepôt et l'impact de ces changements sur la charge. Pour résumer cette typologie, nous pouvons dire que dans le cas de la création d'un concept (mesure, niveau, etc.) dans le modèle, il s'agit d'envisager la création de nouvelles requêtes décisionnelles portant sur ce concept ; dans le cas de la modification et de la suppression d'un concept, il s'agit d'essayer de mettre à jour les requêtes en fonction de l'évolution, et de les supprimer lorsque la mise à jour n'est pas satisfaisante (par exemple, si une requête décisionnelle ne se basait que sur un niveau et que ce dernier est supprimé).

3.2 Traitement des requêtes

Afin d'aider l'administrateur pour la maintenance des requêtes, il s'agit de lui indiquer quelles opérations réaliser, sur quelles requêtes, etc. Ceci est rendu possible grâce à une représentation captant les concepts utilisés par chacune des requêtes de la charge qui permet de détecter quelles sont les requêtes à modifier et à l'utilisation de la typologie guidant les modifications.

Les requêtes décisionnelles, dans un contexte relationnel, sont basées sur des clauses qui exploitent les tables et leurs attributs. Ainsi, nous proposons de définir un contexte de détection constitué de deux matrices : une matrice «requêtes-tables» et une matrice «requêtes-attributs». La matrice «requêtes-tables» a pour lignes les requêtes de la charge et pour colonnes les tables de l'entrepôt. L'utilisation d'une table dans une des clauses de la requête est symbolisée par un 1 et son absence par un 0. La matrice «requêtes-attributs» a pour lignes les requêtes de la charge et pour colonnes les attributs de l'entrepôt. L'utilisation d'un attribut dans une des clauses de la requête est symbolisée par un 1 et son absence par un 0.

Ainsi, vis-à-vis des modifications subies par l'entrepôt, on détecte de façon immédiate les requêtes impactées en interrogeant ces deux matrices par rapport aux concepts qui ont subi des modifications. En combinant l'utilisation de la typologie et de ces deux matrices, il est alors possible d'indiquer à l'administrateur quelles requêtes modifier et comment, ce qui constitue une véritable aide pour lui si l'on considère les nombreuses requêtes pouvant constituer une charge dans un contexte réel (Golfarelli et Saltarelli, 2003).

Souvent, une évolution de l'entrepôt peut induire un enrichissement des possibilités d'analyse. Que ce soit la création de mesure, la création d'un niveau de hiérarchie, etc., il est intéressant d'ajouter des requêtes dans la charge correspondant à l'exploitation de ces nouvelles possibilités d'analyse. Dans cet article, l'idée est de suggérer à l'administrateur sur quels éléments peuvent se baser les nouvelles requêtes. Concernant la création d'élément, l'administrateur peut définir différentes requêtes portant sur ce nouvel élément s'il estime que cela est pertinent. Une fois ces requêtes ajoutées dans la charge, ces dernières sont bien évidemment représentées dans les deux matrices, puisqu'elles seront peut-être elles-mêmes amenées à subir des modifications en répercussion à d'autres évolutions.

Notons que disposant de cette charge cohérente vis-à-vis du nouveau modèle, l'administrateur peut appliquer une méthode qui le guidera dans le choix d'une configuration d'optimisation pour l'entrepôt de données, qui remettra ou non en cause la configuration précédente.

4 Étude de cas LCL-Le Crédit Lyonnais

Nous développons actuellement notre prototype d'aide à la maintenance de charge pour la gestion de performances dans un contexte évolutif sous Oracle 10g. Les matrices sont stockées sous forme de tables relationnelles qui pourront facilement être maintenues pour prendre en compte l'ajout et l'évolution des requêtes, évitant une reconstruction systématique des matrices. Les changements opérés sur le modèle sont stockés dans deux tables relationnelles temporaires, jusqu'à ce que le changement ait été pris en compte dans la charge : une pour les changements sur les tables et une pour les changements sur les attributs. Ces tables présentent différentes informations utiles telles que le type de changement opéré, le nom de l'élément qui a subi le changement, etc.

Pour illustrer notre approche, nous proposons une étude de cas réel simplifié défini avec l'établissement bancaire LCL-Le Crédit Lyonnais. L'entrepôt de données concerne l'analyse du «Net Banking Income» ou NBI (Figure 2). Le NBI correspond à ce que rapporte un client à l'établissement bancaire. Cette mesure est observée par rapport aux dimensions suivantes : CUSTOMER (client), AGENCY (agence) and YEAR (année). La dimension AGENCY présente une hiérarchie de dimension avec le niveau COMMERCIAL_DIRECTION (unité commerciale), qui correspond à un regroupement géographique d'agences.

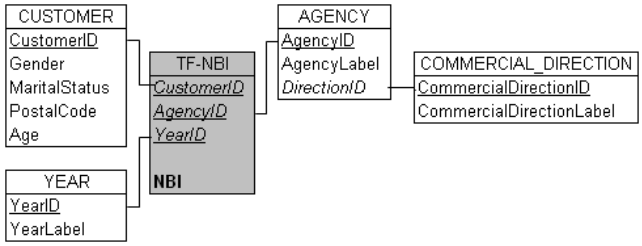

FIG. 2 – *Schéma initial de l'entrepôt pour l'analyse du NBI.*

À partir de l'utilisation de l'entrepôt de données, une charge de requêtes décisionnelles est définie. L'objectif visé étant l'optimisation des temps de réponse des analyses, nous considérons que la charge ne contient que des requêtes décisionnelles. Classiquement, le processus d'analyse consiste à résumer les données en utilisant (1) des opérateurs d'agrégation appliqués sur la mesure, tel que l'opérateur SUM ou AVG ; (2) des clauses GROUP BY permettant le regroupement. S'agissant de requêtes décisionnelles, nous utilisons plus spécifiquement des clauses GROUP BY CUBE et GROUP BY ROLLUP.

Dans un souci pédagogique, nous considérons un extrait de la charge initiale comprenant cinq requêtes qui permettent d'analyser le NBI sous différents points de vue (Figure 3).

Q1: **SELECT** Gender, Agency Label, YearLabel, AVG (NBI) **FROM** AGENCY, YEAR, TF-NBI WHERE AGENCY.AgencyID=TF-NBI.AgencyID AND YEAR.YearID=TF-NBI.YearID **GROUP BY CUBE** (Gender, Agency Label, YearLabel);

Q2: **SELECT** CommercialDirectionLabel, YearLabel, AVG (NBI) **FROM** AGENCY, COMMERCIAL_DIRECTION, TF-NBI WHERE AGENCY.AgencyID=TF-NBI.AgencyID AND COMMERCIAL_DIRECTION.CommercialDirectionID= AGENCY.DirectionID **GROUP BY CUBE** (CommercialDirectionLabel, YearLabel);

Q3: **SELECT** MaritalStatus, YearLabel, SUM (NBI) **FROM** CUSTOMER, TF-NBI WHERE CUSTOMER.CustomerID=TF-NBI.CustomerID **GROUP BY CUBE** (MaritalStatus, YearLabel);

Q4: **SELECT** CommercialDirectionLabel, AgencyLabel, SUM (NBI) **FROM** AGENCY, COMMERCIAL_DIRECTION, TF-NBI WHERE AGENCY.AgencyID=TF-NBI.AgencyID AND COMMERCIAL_DIRECTION.CommercialDirectionID=AGENCY.DirectionID AND YearLabel='2000' **GROUP BY ROLLUP** (CommercialDirectionLabel, AgencyLabel);

Q5: **SELECT** AgencyLabel, AVG (NBI) **FROM** AGENCY, COMMERCIAL_DIRECTION, TF-NBI WHERE AGENCY.AgencyID=TF-NBI.AgencyID AND COMMERCIAL_DIRECTION.CommercialDirectionID=AGENCY.DirectionID AND YearLabel='2000' AND CommercialDirectionLabel='Lyon' **GROUP BY** AgencyLabel;

FIG. 3 – *Extrait de la charge initiale pour l'analyse du NBI.*

Le contexte de détection pour cet extrait de charge est constitué de la matrice requêtes-tables de la Figure 4 et de la matrice requêtes-attributs de la Figure 5.

Les changements suivants sont opérés sur le schéma de l'entrepôt :

Aide à la maintenance de charge

	t1	t2	t3	t4	t5
q1	1	1	1	0	0
q2	1	1	0	0	1
q3	1	0	0	1	0
q4	1	1	0	0	1
q5	1	1	0	0	1

t1	TF-NBI
t2	AGENCY
t3	YEAR
t4	CUSTOMER
t5	COMMERCIAL_DIRECTION

FIG. 4 – *Matrice requêtes-tables pour l'extrait de la charge pour l'analyse du NBI.*

	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14
q1	1	1	0	0	0	0	1	0	1	0	1	1	1	1
q2	1	0	1	1	1	0	0	0	1	0	1	0	0	1
q3	0	0	0	0	0	1	0	1	0	1	1	0	0	1
q4	1	1	1	1	1	0	0	0	1	0	1	0	0	0
q5	1	1	1	1	0	0	0	0	1	0	1	0	0	0

a1	AGENCY.AgencyID
a2	AGENCY.AgencyLabel
a3	AGENCY.DirectionID
a4	COMMERCIAL_DIRECTION.CommercialDirectionID
a5	COMMERCIAL_DIRECTION.CommercialDirectionLabel
a6	CUSTOMER.CustomerID
a7	CUSTOMER.Gender
a8	CUSTOMER.MaritalStatus
a9	TF-NBI.AgencyID
a10	TF-NBI.CustomerID
a11	TF-NBI.NBI
a12	TF-NBI.YearID
a13	YEAR.YearID
a14	YEAR.YearLabel

FIG. 5 – *Matrice requêtes-attributs pour l'extrait de la charge pour l'analyse du NBI.*

– ajout de l'attribut Segment dans la dimension CUSTOMER ;
– renommage du niveau COMMERCIAL_DIRECTION en DIRECTION ;
– renommage des attributs CommercialDirectionID et CommercialDirectionLabel respectivement en DirectionID et DirectionLabel ;
– insertion du niveau COMMERCIAL_UNIT entre les niveaux AGENCY et DIRECTION.

Nous disposons alors du schéma de la Figure 6.

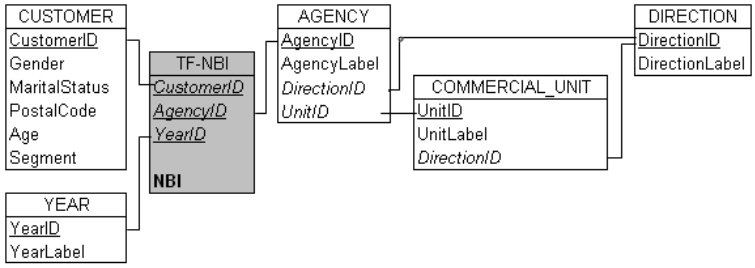

FIG. 6 – *Schéma mis à jour de l'entrepôt pour l'analyse du NBI.*

Différentes indications peuvent alors être fournies à l'administrateur pour l'aider à maintenir la charge qu'il pourra utiliser pour la gestion des performances. Il s'agit d'indiquer d'une part les requêtes devant être modifiées et d'autre part les possibilités en terme d'ajout de requêtes. Les attributs concernés par les modifications sont a4 (CommercialDirectionID) et a5 (CommercialDirectionLabel). Dans la matrice requêtes-attributs, les requêtes pour lesquelles la valeur 1 est présente à l'intersection avec ces attributs sont les requêtes q2 et q4, elles doivent être modifiées selon ces deux attributs, et la requête q5 par rapport à l'attribut a4. Par rapport

aux modifications concernant des tables, c'est la table t5 (COMMERCIAL_DIRECTION) qui est concernée. Dans la matrice requêtes-tables, les requêtes pour lesquelles la valeur 1 est présente à l'intersection avec la table t5 sont les requêtes q2, q4 et q5.

Pour des raisons pratiques, nous préférons communiquer à l'administrateur les modifications à considérer pour chaque requête :

- q2 : table COMMERCIAL_DIRECTION, attributs CommercialDirectionID et CommercialDirectionLabel
- q4 : table COMMERCIAL_DIRECTION, attributs CommercialDirectionID et CommercialDirectionLabel
- q5 : table COMMERCIAL_DIRECTION, attribut CommercialDirectionID

Concernant l'ajout de requêtes dans la charge, il serait intéressant d'utiliser l'attribut CUSTOMER.Segment et la table (le niveau) COMMERCIAL_UNIT.

En exploitant ces informations, l'administrateur peut ainsi maintenir la charge (Figure 7). Ainsi, les requêtes q2, q4 et q5 ont été mises à jour afin de rester consistantes avec le schéma mis à jour de l'entrepôt. Par ailleurs, l'administrateur a ajouté les requêtes q6 et q7 pour prendre en compte les besoins d'analyse portant sur le niveau créé COMMERCIAL_UNIT et sur l'attribut Segment ajouté. Il peut alors appliquer un algorithme de sélection de structures d'optimisation sur cette charge et ajuster l'optimisation de l'entrepôt en fonction du résultat obtenu.

Q1: **SELECT** Gender, AgencyLabel, YearLabel, AVG (NBI) **FROM** AGENCY, YEAR, TF-NBI WHERE AGENCY.AgencyID=TF-NBI.AgencyID AND YEAR.YearID=TF-NBI.YearID **GROUP BY CUBE** (Gender, AgencyLabel, YearLabel);

Q2: **SELECT** *DirectionLabel*, YearLabel, AVG (NBI) **FROM** AGENCY, *DIRECTION*, TF-NBI WHERE AGENCY.AgencyID=TF-NBI.AgencyID AND *DIRECTION.DirectionID*=AGENCY.DirectionID **GROUP BY CUBE** (*DirectionLabel*, YearLabel);

Q3: **SELECT** MaritalStatus, YearLabel, SUM (NBI) **FROM** CUSTOMER, TF-NBI WHERE CUSTOMER.CustomerID=TF-NBI.CustomerID **GROUP BY CUBE** (MaritalStatus, YearLabel);

Q4: **SELECT** *DirectionLabel*, AgencyLabel, SUM (NBI) **FROM** AGENCY, *DIRECTION*, TF-NBI WHERE AGENCY.AgencyID=TF-NBI.AgencyID AND *DIRECTION.DirectionID*=AGENCY.DirectionID AND YearLabel='2000' **GROUP BY ROLLUP** (*DirectionLabel*, AgencyLabel);

Q5: **SELECT** AgencyLabel, AVG (NBI) **FROM** AGENCY, *DIRECTION*, TF-NBI WHERE AGENCY.AgencyID=TF-NBI.AgencyID AND *DIRECTION.DirectionID*=AGENCY.DirectionID AND YearLabel='2000' AND *DirectionLabel*='Lyon' **GROUP BY** AgencyLabel;

Q6: ***SELECT*** *AgencyLabel, UnitLabel, AVG (NBI)* ***FROM*** *AGENCY, COMMERCIAL_UNIT, TF-NBI WHERE AGENCY.AgencyID=TF-NBI.AgencyID AND COMMERCIAL_UNIT.UnitID=AGENCY.UnitID* ***GROUP BY ROLLUP*** *(UnitLabel , AgencyLabel);*

Q7: ***SELECT*** *Segment, AVG (NBI)* ***FROM*** *CUSTOMER, TF-NBI WHERE CUSTOMER.CustomerID=TF-NBI.CustomerID AND* ***GROUP BY*** *Segment;*

FIG. 7 – *Extrait de la charge mise à jour pour l'analyse du NBI.*

5 Conclusion

Dans cet article, nous avons abordé le problème de l'évolution de charge dans les entrepôts de données, du point de vue de l'aide à l'administration, en exploitant un contexte de détection des changements à opérer sur les requêtes. L'avantage de notre approche est de permettre à l'administrateur d'adopter une démarche pro-active pour gérer les performances de l'entrepôt de données évolutif en s'occupant de la maintenance de la charge.

Un prototype est actuellement en cours d'implémentation. Par la suite, nous souhaitons aider davantage l'administrateur dans la maintenance et la création de nouvelles requêtes en

allant vers une approche semi-automatique. Il s'agit par exemple de proposer un algorithme de génération de requêtes qui exploiterait certains paramètres que l'administrateur préciserait, tels que le nombre de requêtes à créer, le nombre moyen d'axes exploités, etc. Ces requêtes générées pourraient alors être soumises à l'expertise de l'administrateur. Concernant la mise à jour de requêtes, il s'agit de s'inspirer des travaux réalisés dans le cadre de la maintenance de vues par la réécriture des requêtes les définissant (Bellahsene, 2002) en prenant en compte les spécificités des requêtes décisionnelles et leur pertinence dans la charge.

Références

Agrawal, S., S. Chaudhuri, et V. R. Narasayya (2000). Automated Selection of Materialized Views and Indexes in SQL Databases. In *VLDB 00*, pp. 496–505.

Bellahsene, Z. (2002). Schema Evolution in Data Warehouses. *Knowledge and Information Systems 4*(3), 283–304.

Bellatreche, L. (2000). *Utilisation des vues matérialisées, des index et de la fragmentation dans la conception logique et physique d'un entrepôt de données*. Thèse de doctorat, France.

Favre, C., F. Bentayeb, et O. Boussaid (2007). Evolution of Data Warehouses' Optimization : a Workload Perspective. In *DaWaK 07*, Volume 4654 of *LNCS*, pp. 13–22.

Golfarelli, M. et E. Saltarelli (2003). The Workload You Have, the Workload You Would Like. In *DOLAP 03*, pp. 79–85.

Hurtado, C. A., A. O. Mendelzon, et A. A. Vaisman (1999). Maintaining Data Cubes under Dimension Updates. In *ICDE 99*, pp. 346–355.

Kotidis, Y. et N. Roussopoulos (1999). DynaMat : A Dynamic View Management System for Data Warehouses. *SIGMOD Rec. 28*(2), 371–382.

Krompass, S., U. Dayal, H. A. Kuno, et A. Kemper (2007). Dynamic workload management for very large data warehouses : Juggling feathers and bowling balls. In *VLDB 07*, pp. 1105–1115.

Lawrence, M. et A. Rau-Chaplin (2006). Dynamic View Selection for OLAP. In *DaWaK 06*, pp. 33–44.

Rizzi, S. et E. Saltarelli (2003). View Materialization vs. Indexing : Balancing Space Constraints in Data Warehouse Design. In *CAiSE 03*, pp. 502–519.

Theodoratos, D. et T. Sellis (2000). Incremental Design of a Data Warehouse. *Journal of Intelligent Information Systems 15*(1), 7–27.

Summary

In the context of data warehouses evolution, we propose to help the administrator in the maintenance of the workload (set of queries) which is used in the performance evaluation process. By propagating data warehouse evolutions within the workload, the administrator can manage performances in a pro-active way, without waiting for the users' queries on the updated data warehouse to build a valid workload to optimize performances.