WEDriK : une plateforme pour des analyses personnalisées dans les entrepôts de données évolutifs

Cécile Favre, Fadila Bentayeb, Omar Boussaïd

ERIC, Université Lumière Lyon 2
5 avenue Pierre Mendès-France
69676 Bron Cedex
{cfavre|bentayeb}@eric.univ-lyon2.fr
omar.boussaid@univ-lyon2.fr

Résumé. Dans la cadre d'une collaboration avec l'établissement bancaire LCL-Le Crédit Lyonnais, nous avons conçu et développé un entrepôt de données qui centralise des données provenant de différentes sources pour répondre aux besoins d'analyse des utilisateurs concernant les ventes issues d'opérations marketing. Le schéma initial de l'entrepôt permet de répondre à des besoins d'analyse communs à la plupart des utilisateurs. Mais comment répondre à des besoins d'analyse personnalisés ? Dans cet article, nous exposons une approche qui permet aux utilisateurs d'intégrer leurs propres connaissances du domaine afin d'enrichir les possibilités d'analyse de l'entrepôt, en faisant évoluer son schéma. Nous présentons également les éléments nécessaires à sa mise en œuvre (méta modèle, algorithmes,...). Les connaissances utilisateurs sont exprimées sous la forme de règles de type «si-alors». Ces règles sont utilisées pour créer des niveaux de granularité dans les hiérarchies de dimension. Nous avons développé cette approche avec une plateforme nommée WEDriK (data Warehouse Evolution Driven by Knowledge), afin de permettre aux utilisateurs de LCL de répondre à des besoins d'analyse personnalisés sur les données bancaires.

1 Introduction

Un entrepôt de données est conçu pour répondre à un ensemble de besoins d'analyse communs à la plupart des utilisateurs. Cependant, les utilisateurs peuvent avoir des besoins variés auxquels l'entrepôt n'est pas forcément en mesure de répondre, a fortiori dans une grande entreprise, dans laquelle les utilisateurs exercent de nombreux métiers. La création de magasins de données tentent de se rapprocher des besoins utilisateurs en fonction de leurs métiers. Néanmoins, chaque utilisateur dispose de connaissances particulières du domaine et de besoins d'analyse qui lui sont propres. C'est précisément le problème rencontré par l'établissement bancaire LCL[1], avec lequel nous collaborons.

Nous avons constitué un entrepôt de données (stocké en relationnel) pour analyser des données provenant de différentes sources qui concernent le marketing local (Khoualdia et Malaret,

[1]Collaboration avec la Direction d'Exploitation Rhône-Alpes Auvergne de LCL–Le Crédit Lyonnais (LCL) dans le cadre d'une Convention Industrielle de Formation par la Recherche (CIFRE)

2006). Le marketing local consiste à mener des actions marketing pour répondre à des besoins de vente spécifiques (atteindre des objectifs commerciaux, création d'une agence,...). Il s'agit alors, pour les responsables des équipes commerciales, de faire des demandes de ciblage de clients, sur une zone géographique donnée, et d'utiliser ces ciblages pour optimiser les ventes des conseillers. Par exemple, un responsable peut mener une action pour vendre des Plans d'Epargne Logement dans l'agence Lyon Terreaux pour atteindre les objectifs de vente sur ce produit. Il est alors nécessaire de pouvoir mesurer l'intérêt de ces demandes marketing.

Pour répondre à des besoins d'analyse personnalisés émergeant de cet entrepôt, nous suggérons que les utilisateurs soient impliqués dans le processus d'évolution du schéma de l'entrepôt puisque c'est ce schéma qui en détermine les possibilités d'analyse. Ainsi, l'évolution du schéma de l'entrepôt, et en particulier l'évolution des hiérarchies de dimension, doit être guidée par les utilisateurs eux-mêmes.

Dans Favre et al. (2006b) et Favre et al. (2006a), nous avons proposé un modèle d'entrepôt de données évolutif et sa formalisation. Notre modèle d'entrepôt de données à base de règles *R-DW* (Rule-based Data Warehouse) est composé d'une partie «fixe», correspondant à un schéma qui répond à des besoins d'analyse communs à un ensemble d'utilisateurs ; complétée par une partie «évolutive» définie par les règles d'agrégation répondant aux besoins d'analyse personnalisés (Grange et Jouaneton, 2006).

Dans cet article, nous inscrivons ce modèle dans une approche complète pour impliquer les utilisateurs dans le processus d'évolution de l'entrepôt[2], afin de leur fournir des analyses personnalisées. Cette approche est composée de quatre étapes : (1) l'acquisition des connaissances de l'utilisateur, (2) l'intégration de ces connaissances ; (3) l'évolution du schéma (mise à jour de la hiérarchie de dimension) en fonction des connaissances intégrées, et (4) l'analyse prenant en compte le nouveau schéma. Les connaissances utilisateurs sont représentées sous la forme de règles de type «si-alors». Ce type de règles permet de modéliser les connaissances de façon simple et explicite, et se prête bien à la représentation des connaissances utilisées ici sur la façon d'agréger les données. En effet, ces règles, dites d'agrégation, sont ensuite utilisées pour générer de nouveaux axes d'analyse en créant dans les hiérarchies de dimension des niveaux de granularité.

Par ailleurs, nous proposons dans cet article les éléments nécessaires à la mise en œuvre de notre approche à travers l'implémentation d'une plateforme baptisée WEDriK (data Warehouse Evolution Driven by Knowledge) : la définition d'un méta modèle pour gérer l'évolution du schéma de l'entrepôt, des algorithmes permettant l'évolution des hiérarchies de dimension elles-même ainsi que la vérification des règles exprimées par les utilisateurs... Cette plateforme permet alors aux utilisateurs de LCL de répondre à des besoins d'analyse personnalisés sur les données bancaires, comme nous le montrons dans cet article à travers un cas d'application réel.

Le reste de cet article est organisé de la façon suivante. Tout d'abord, nous discutons l'état de l'art dans la Section 2 sur la conception et l'évolution de schéma dans les entrepôts de données. Puis, nous introduisons un exemple motivant l'utilisation de notre approche dans la Section 3. Nous présentons ensuite notre approche dans la Section 4, avant d'en exposer son développement dans la Section 5. Enfin, nous concluons et indiquons les perspectives de ce travail dans la Section 6.

[2]Notons que nous traitons de l'évolution de l'entrepôt de données, mais notre approche pourrait être appliquée sur les magasins de données eux-mêmes

2 État de l'art

Pour concevoir le schéma d'un entrepôt, nous distinguons dans la littérature trois types d'approches : celles guidées par les sources de données, celles guidées par les besoins d'analyse et les approches mixtes qui combinent les deux premières. Les approches guidées par les sources de données ignorent les besoins d'analyse *a priori*. Elles concernent en particulier les travaux sur l'automatisation de la conception du schéma de l'entrepôt à partir de ceux des sources. Par exemple, Golfarelli et al. (1998) proposent une méthodologie semi-automatique pour construire un modèle d'entrepôt de données à partir des schémas E/R qui représentent les bases de données sources. L'inconvénient de ces approches est qu'elles supposent que le modèle ainsi généré pourra répondre aux besoins d'analyse, ce qui n'est pas forcément le cas. Les approches guidées par les besoins d'analyse proposent de définir le schéma de l'entrepôt en partant précisément des besoins décisionnels, comme l'a proposé Kimball (1996). Ainsi, les utilisateurs sont impliqués dans le processus de définition du schéma lors de la collecte des besoins d'analyse. Cela permet de garantir l'acceptation du système par les utilisateurs. Cependant la longévité de ce modèle est limitée, compte tenu du fait que le modèle dépend beaucoup des besoins exprimés par les personnes impliquées dans le processus de développement de l'entrepôt. En outre, il est non seulement difficile de déterminer de façon exhaustive les besoins d'analyse pour l'ensemble des utilisateurs à un instant donné, mais il est impossible de déterminer leurs besoins à venir. Par ailleurs, il faut que les besoins d'analyses recueillis puissent trouver une réponse dans les sources de données disponibles. L'utilité des approches mixtes est alors bien réelle puisqu'elles permettent de combiner les deux types d'approches précédents. Il s'agit en effet de mettre en adéquation des schémas candidats générés à partir des sources de données avec les besoins d'analyse exprimés par les utilisateurs (Bonifati et al., 2001; Nabli et al., 2005).

Il apparaît donc crucial, lors de la phase de conception de l'entrepôt, de prendre en compte à la fois les sources de données, et les besoins des utilisateurs. Mais, une fois l'entrepôt de données construit, ces deux paramètres sont amenés à subir des changements devant être répercutés sur le schéma de l'entrepôt, nécessitant une évolution de ce dernier. Dans la littérature, on peut distinguer deux alternatives pour remédier à ce problème : la mise à jour de schéma, et la modélisation temporelle. La première approche consiste à mettre à jour le schéma grâce à des opérateurs qui font évoluer un schéma donné (Blaschka et al., 1999; Hurtado et al., 1999). Dans ce cas, un seul schéma est supporté, et les évolutions que le schéma subit ne sont donc pas conservées. La deuxième alternative consiste, elle, à garder justement la trace de ces évolutions, en utilisant des labels de validité temporelle. Ces labels sont apposés soit au niveau des instances (Bliujute et al., 1998), soit au niveau des liens d'agrégation (Mendelzon et Vaisman, 2000), ou encore au niveau des versions du schéma (Bébel et al., 2004; Morzy et Wrembel, 2004). Les évolutions du schéma sont donc bien conservées et assurent la cohérence des analyses. Mais ce type de solutions nécessite une réimplémentation des outils d'analyse, de chargement de données,... afin de gérer les particularités de ces modèles. Un autre courant a émergé, se focalisant particulièrement sur la propagation des changements. Il s'agit de travaux portant sur les vues, basés sur l'hypothèse qu'un entrepôt est un ensemble de vues construites à partir des sources de données. Ainsi, il s'agit de ramener le problème de l'évolution des sources de données à celui de la maintenance des vues (Bellahsene, 2002).

Ces différentes approches constituent des alternatives intéressantes pour répondre au pro-

blème de l'évolution de schéma suite à une modification dans les sources de données. Ce sont des solutions techniques devant être mises en œuvre par l'administrateur pour faire évoluer l'entrepôt de données. Cependant, elles n'impliquent pas directement les utilisateurs dans le processus d'évolution. De ce fait, elles n'apportent pas de solution à l'émergence de nouveaux besoins d'analyse exprimés par les utilisateurs, et par conséquent, aucune solution au problème de la personnalisation des analyses.

Or, la personnalisation dans les entrepôts de données devient un enjeu crucial, permettant l'utilisation de ceux-ci par le plus grand nombre d'utilisateurs, assurant ainsi une meilleure possibilité de rentabilité de l'investissement réalisé pour cette technologie coûteuse. C'est une piste de recherche émergeante au sein de la communauté des chercheurs travaillant sur les entrepôts de données. Jusque là, les travaux se focalisent surtout sur la visualisation des grandes volumétries de données en se basant sur la modélisation et la prise en compte des préférences utilisateurs sous forme de profil (Bellatreche et al., 2005). Il s'agit alors d'affiner les requêtes pour n'afficher qu'une partie des données qui répond donc aux préférences. Notre approche, quant à elle, ne se situe pas dans une perspective de visualisation personnalisée grâce à la sélection d'informations disponibles pour tous, mais plutôt dans une prise en compte d'informations supplémentaires pour permettre des analyses personnalisées.

3 Exemple introductif

Pour illustrer notre approche de modélisation d'entrepôt de données évolutif à base de règles, nous utilisons, tout au long de cet article, le cas réel de la banque LCL. Pour étudier l'impact des demandes marketing, nous avons défini un modèle d'entrepôt en constellation (Figure 1) avec deux tables de faits : celle des résultats commerciaux (`TF-RESULTAT`) et celle des PNB (`TF-PNB`). Le PNB (Produit Net Bancaire) correspond à ce que rapporte un client à l'établissement bancaire. L'analyse du PNB par agence pourrait ainsi être mise en parallèle avec celle des résultats obtenus en fonction des demandes marketing réalisées. Nous nous intéressons ici à la partie encadrée du modèle concernant le PNB. Le PNB est une mesure qui est donc analysée selon les dimensions `CLIENT`, `AGENCE` et `ANNEE`. Il est possible d'agréger les données selon le niveau `UNITE_COMMERCIALE`, qui est un regroupement d'agences, tel que le montre le schéma de la dimension `AGENCE` de la Figure 2a.

Supposons qu'un utilisateur veuille analyser les données selon le type d'agence ; il sait qu'il en existe trois : type «étudiant» pour les agences ne comportant que des étudiants, type «non résident» lorsque les clients ne résident pas en France, et le type «classique» pour les agences ne présentant pas de particularité. Ces informations n'étant pas dans l'entrepôt, il est impossible pour lui d'obtenir cette analyse.

Nous proposons alors à l'utilisateur d'intégrer sa connaissance sur les types d'agence et les identifiants d'agences s'y rappportant pour générer le niveau de granularité `TYPE_AGENCE`, en définissant les valeurs de l'attribut `NomTypeAgence` qui caractérise ce niveau, et créer le lien d'agrégation avec le niveau `AGENCE`, selon le schéma de la dimension de la Figure 2b. Pour cela, l'utilisateur formule les règles suivantes :

$(R1)$ $si\ idAgence \in$ {'01903','01905','02256'} $alors\ NomTypeAgence =$ 'étudiant'

$(R2)$ $si\ idAgence =$ '01929' $alors\ NomTypeAgence =$ 'non résident'

$(R3)$ $si\ idAgence \notin$ {'01903','01905','02256','01929'} $alors\ NomTypeAgence =$ 'classique'

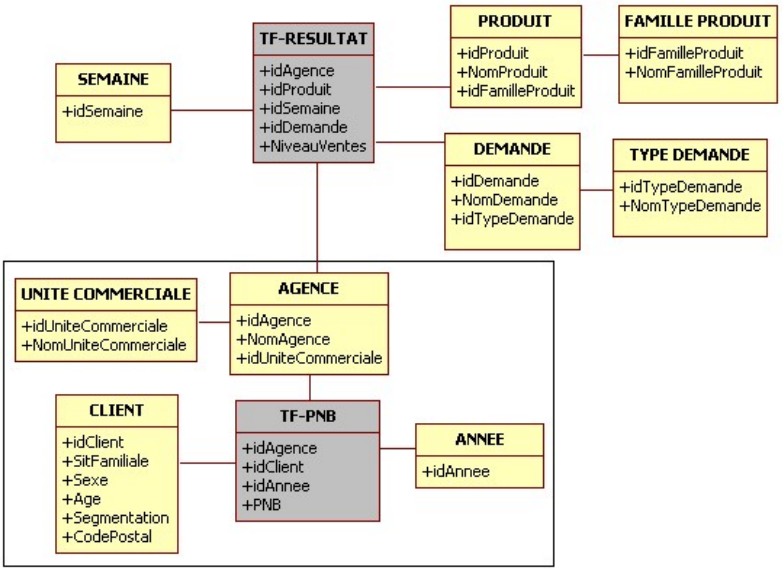

FIG. 1 – *Modèle en constellation pour l'analyse des demandes marketing de LCL*

Ainsi, de nouvelles analyses, basées sur ces règles, peuvent être effectuées. Par exemple, il sera possible de construire des agrégats du PNB, en considérant qu'ils relèvent d'une agence étudiante ($R1$), d'une agence de non résidents ($R2$), ou d'une agence classique ($R3$).

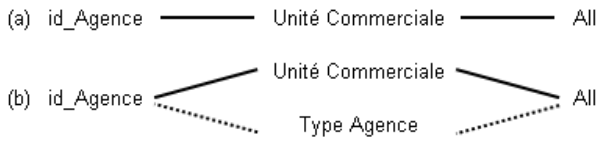

FIG. 2 – *Schémas de la dimension AGENCE : (a) initial, (b) enrichi par l'utilisateur*

4 Une approche utilisateur pour la mise à jour des hiérarchies de dimension

4.1 Architecture

L'architecture de notre approche se décompose en quatre étapes (Figure 3). Premièrement, une phase d'acquisition des connaissances permet de récolter celles-ci sous forme de règles de type «si-alors». Deuxièmement, une phase d'intégration permet de les stocker dans le SGBD

(Système de Gestion de Bases de Données). Troisièmement, la phase d'évolution permet de faire évoluer le schéma, en permettant la création de nouveaux niveaux de granularité au sein des hiérarchies de dimension. Le modèle induit par les règles évolue ainsi de façon incrémentale, selon les besoins exprimés par les utilisateurs, rendant possible la quatrième phase d'analyse sur le modèle enrichi.

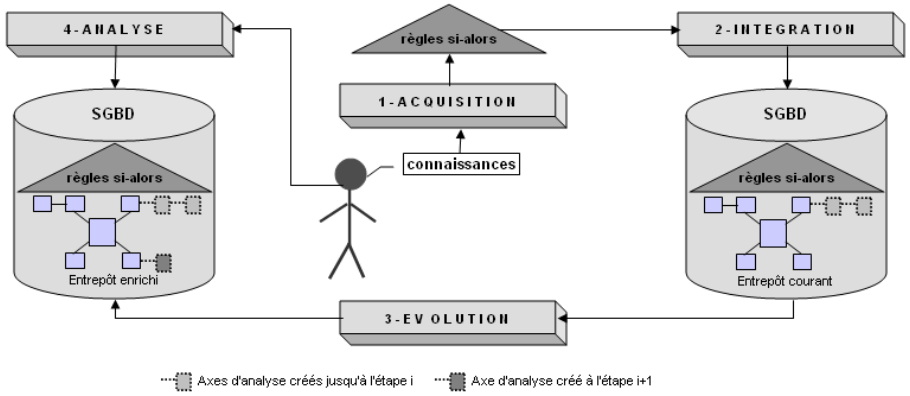

FIG. 3 – *Architecture à base de règles des entrepôts de données évolutifs*

Notre approche est centrée utilisateur, mais il est primordial que l'implication des utilisateurs ne compromette pas le schéma initial de l'entrepôt qui répond à des besoins d'analyse communs à l'ensemble des utilisateurs. Ainsi, les nouveaux besoins d'analyse ne doivent modifier ni la table des faits, ni les niveaux de granularité directement liés à celle-ci. C'est pourquoi notre approche se base sur le modèle *R-DW*, qui est composé de deux parties : une partie fixe et une partie évolutive. La partie fixe est composée d'une table de faits et des tables de dimensions qui y sont directement liées. La partie évolutive est définie par un ensemble de règles d'agrégation, qui permettent de générer de nouveaux niveaux de granularité.

4.2 Ajout et insertion de niveaux de granularité dans les hiérarchies de dimension

Le niveau de granularité créé peut être ajouté à la fin d'une hiérarchie (comme un niveau qui présente l'information la plus agrégée) ou inséré entre deux niveaux existants. Dans le second cas, il est alors nécessaire que le lien d'agrégation entre le niveau créé et le niveau supérieur existant soit défini automatiquement. Mais ce lien peut être créé seulement s'il est possible d'agréger sémantiquement les données du niveau créé vers le niveau supérieur existant. Par exemple, considérons le schéma de dimension de la Figure 2a. Une unité commerciale correspond à un regroupement d'agences, selon leur localisation géographique. Il est alors possible d'insérer entre les niveaux `AGENCE` et `UNITE COMMERCIALE`, un niveau qui correspondrait à un regroupement d'agences par quartier, où une unité commerciale correspondrait donc à un regroupement de ces groupes d'agences. En revanche, si l'on ajoute un nouveau niveau qui permet d'agréger les données des agences selon leur taille : petite, moyenne et grande, il est impossible de créer un lien d'agrégation entre la taille d'agence et le niveau existant `UNITE`

COMMERCIALE. Le niveau créé doit alors être ajouté à la fin d'une hiérarchie et non inséré entre AGENCE et UNITE COMMERCIALE.

4.3 Règles d'agrégation

Les règles d'agrégation sont des règles de type «*si-alors*». La clause «*si*» permet d'exprimer les conditions sur les attributs caractérisant le niveau de granularité inférieur, c'est à dire le niveau à partir duquel sera généré le nouveau. Les différentes règles définissant un nouveau niveau vont ainsi permettre de construire une partition des instances du niveau inférieur. Dans la clause «*alors*» figure la définition du niveau de granularité à créer, c'est à dire la définition des valeurs des attributs caractérisant ce nouveau niveau de granularité. Ainsi, chaque classe de la partition est mise en correspondance avec une instance du nouveau niveau (Figure 4). Les données concernant chaque instance du niveau inférieur pourront alors être agrégées en une instance du niveau créé. En effet, nous représentons ici les hiérarchies classiques, dans lesquelles une instance d'un niveau donné correspond à une instance du niveau supérieur.

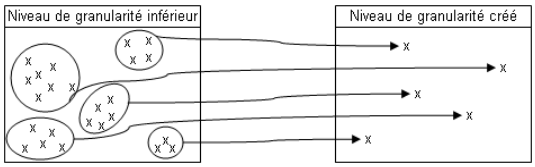

FIG. 4 – *Partition induite par les règles d'agrégation*

4.4 Contraintes sur les règles

Les règles permettent d'intégrer les besoins d'analyse des utilisateurs pour définir de nouveaux niveaux de granularité dans les hiérarchies de dimensions, rendant ainsi le modèle plus flexible. Cependant, cette flexibilité du modèle ne doit pas être apportée au détriment de sa cohérence, étant donné que la cohérence du modèle assure celle des analyses. Pour valider les règles exprimées par les utilisateurs, nous définissons alors trois types de contraintes : (1) contrainte d'intégrité, (2) contrainte de cohérence et (3) contrainte de complétude.

La contrainte d'intégrité doit d'une part assurer que les conditions exprimées sur les attributs dans les prémisses des règles sont conformes vis à vis du domaine de définition de ces attributs. D'autre part, la définition de la valeur de l'attribut dans la conclusion de la règle doit également respecter le domaine de définition de cet attribut.

La contrainte de cohérence est directement liée au fait que l'ensemble de règles qui définissent un niveau donné doivent former une partition des instances du niveau inférieur. Cela implique qu'il ne doit pas y avoir de recouvrement entre les conditions exprimées dans les règles, puisque les classes sont par définition mutuellement exclusives. Pour illustrer la problématique du recouvrement des règles définies par un utilisateur, considérons l'exemple de la définition de classes d'âges à partir des âges de la table CLIENT avec les règles qui suivent. Dans ce cas, il y a un recouvrement des règles qui rend impossible le calcul de la valeur de l'attribut NomClasseAge pour une personne âgée de 40 ans par exemple.

(R1) $si\ Age < 60\ alors\ NomClasseAge =$ 'moins de 60 ans'
(R2) $si\ Age < 80\ alors\ NomClasseAge =$ 'moins de 80 ans'
(R3) $si\ Age > 80\ alors\ NomClasseAge =$ 'plus de 80 ans'

La contrainte de complétude a pour but d'assurer que lorsqu'un niveau de granularité est construit, chaque instance, et donc chaque classe de la partition du niveau de granularité inférieur doit avoir une valeur correspondante dans le niveau de granularité créé. Ainsi, concernant l'exemple du type d'agence, il n'est pas possible d'exprimer seulement les règles définissant ce qu'est une agence classique et une agence de non résidents. En effet, dans ce cas, différents indicatifs d'agence (tous ceux correspondant à des agences étudiantes) ne seraient pas repris au niveau `TYPE_AGENCE`.

5 Mise en œuvre

Pour valider notre approche, nous avons développé une plateforme nommée WEDriK[3] (data Warehouse Evolution Driven by Knowledge). Notre plateforme est basée sur deux composants principaux : (1) le modèle de l'entrepôt, qui a été développé avec le SGBD Oracle 10g, et (2) une plateforme web qui permet l'intéraction avec les utilisateurs. Nous détaillons dans les sous-sections suivantes les éléments nécessaires au fonctionnement de la plateforme, ainsi que les fonctionnalités de celle-ci.

5.1 Méta modèle

Pour gérer l'évolution du schéma de l'entrepôt selon notre approche, nous proposons le méta-modèle présenté dans la Figure 5. Un entrepôt de données est un ensemble de tables. Ces tables sont soit des tables de dimension, soit des tables de faits. Chaque table de dimension possède un ou plusieurs niveaux de granularité qui constituent ainsi une hiérarchie de dimension. Chaque niveau peut être généré par un ensemble de règles d'agrégation. Chaque niveau possède un ensemble d'attributs et une clé primaire. Ainsi, chaque niveau correspond soit à un ensemble d'attributs explicites, soit à des attributs générés avec des règles. Parallèlement, les tables de faits présentent une ou plusieurs mesures, et une clé primaire qui est une composition des clés étrangères (clés primaires de certaines tables de dimension).

5.2 Evolution du schéma

Nous considérons les mises à jour dans les hiérarchies de dimension suivantes : l'ajout d'un niveau de granularité d'une part et l'insertion d'un niveau entre deux niveaux existants d'autre part. L'ajout d'un niveau se fait selon l'Algorithme 1. Pour créer le niveau L' avec l'ensemble des règles R, il faut tout d'abord créer la table L', en extrayant l'attribut dans la clause "alors" des règles de R (supposons, pour des raisons de clarté, qu'un seul attribut par niveau est généré). Nous insérons ensuite les différentes valeurs possibles pour cet attribut. Puis nous modifions la structure de la table L en ajoutant cet attribut en tant que clé étrangère. Nous mettons ensuite à jour dans la table L la valeur de cet attribut en appliquant dans la requête de mise à jour la condition exprimée dans la clause "si" des règles.

[3] http ://eric.univ-lyon2.fr/~cfavre/wedrik

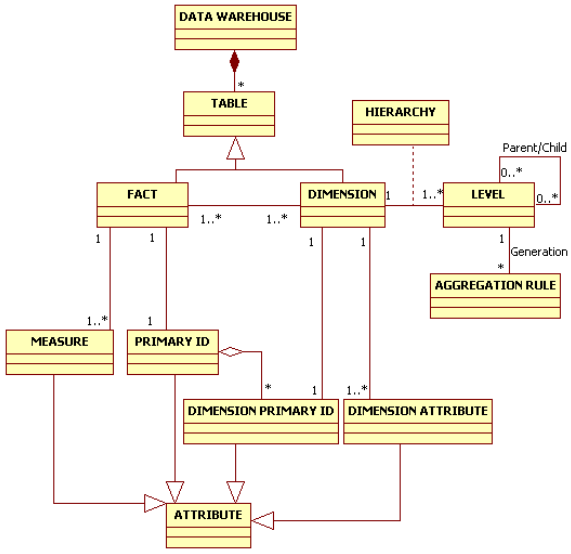

FIG. 5 – *Méta modèle pour gérer l'évolution du schéma de l'entrepôt*

Lorsqu'un niveau doit être inséré entre deux niveaux existants, l'utilisateur ne formule que le lien entre le niveau inférieur et le niveau créé. Le lien d'agrégation entre le niveau créé et le niveau supérieur existant doit être généré automatiquement. L'insertion d'un niveau de granularité se fait selon l'Algorithme 2. Pour insérer le niveau L' avec l'ensemble des règles R, entre le niveau $L1$ et $L2$, il faut d'abord procéder aux mêmes étapes que pour le simple ajout. Il faut ensuite compléter ces étapes afin d'établir le lien entre L' et $L2$. Pour cela, nous ajoutons un attribut à L' en tant que clé étrangère référençant $L2$. Nous avons ensuite à mettre à jour dans la table L' la valeur de cet attribut en déterminant pour chaque instance de L', quelle est l'instance qui correspond dans $L2$. Pour cela nous devons inférer à partir du lien qui existe entre $L1$ et $L2$. Une fois ce lien établi, si le lien entre $L1$ et $L2$ n'a plus lieu d'être, il est

Algorithme 1 Ajout d'un nouveau niveau de granularité

INPUT: niveau existant L, attribut généré A, $type(L'.A)$ le type de $L'.A$, ensemble de règles $R = \{r_j, 1 \leq v\}$, body(r_j) la prémisse de la règle r_j

OUPUT: niveau créé L'

{Créer le nouveau niveau de granularité L'}

1: CREATE TABLE L' ($L'.A$ $type(L'.A)$ AS PRIMARY KEY);
2: ALTER TABLE L ADD (A $type(L'.A)$);
3: **pour tout** ($r_j \in R$) **faire**
4: INSERT INTO L' VALUES (val)
5: UPDATE L SET $A = val$ WHERE $body(r_j)$
6: **fin pour**

alors possible de le supprimer, en modifiant la structure de la table $L1$ par la suppression de la clé étrangère la reliant à $L2$.

Algorithme 2 Insertion d'un nouveau niveau de granularité

INPUT: niveau existant inférieur $L1$, niveau existant supérieur $L2$, attribut liant $L1$ à $L2$ B, $type(L2.B)$ le type de $L2.B$, attribut généré A, $type(L'.A)$ le type de $L'.A$, ensemble de règles $R = \{r_j, 1 \leq v\}$, body(r_j) la prémisse de la règle r_j

OUPUT: niveau inséré L'

{Ajouter le nouveau niveau de granularité L'}

1: CREATE TABLE L' ($L'.A$ $type(L'.A)$ AS PRIMARY KEY);
2: ALTER TABLE $L1$ ADD (A $type(L'.A)$);
3: **pour tout** ($r_j \in R$) **faire**
4: INSERT INTO L' VALUES (val)
5: UPDATE $L1$ SET $A = val$ WHERE $body(r_j)$
6: **fin pour**

{Mettre à jour les liens d'agrégation avec L'}

7: ALTER TABLE L' ADD (B $type(L2.B)$);
8: UPDATE L' SET B=(SELECT DISTINCT B FROM L1 WHERE $L1.A$=$L'.A$);

5.3 Mise en œuvre des contraintes

Tout d'abord, la contrainte d'intégrité est validée à travers l'interface de saisie. Par exemple, l'attribut sur lequel peut se baser une règle est choisi dans une liste, c'est ainsi impossible d'exprimer une règle en fonction d'un attribut qui n'existe pas.

Concernant les contraintes induites par le concept de partition, nous testons la validité des règles en exploitant les données grâce à des requêtes, selon l'Algorithme 3 que nous proposons. Considérons l'ensemble des règles R qui détermine un nouveau niveau de granularité L' à partir du niveau inférieur L. Pour chaque règle $r \in R$, nous générons la requête correspondante $q \in Q$, qui contient dans la clause $WHERE$ les conditions exprimées dans la clause "si" de la règle. Chaque requête nous fournit un ensemble d'instances du niveau L. Premièrement, nous vérifions que l'intersection de tous ces ensembles pris deux à deux est vide. Deuxièmement, nous vérifions que l'union de ces ensembles correspond à l'ensemble des instances du niveau L.

5.4 Fonctionnalités

Les fonctionnalités de notre plateforme peuvent être décrites en suivant les quatre étapes de notre approche globale d'évolution de schéma.

Premièrement, concernant l'acquisition des connaissances, les utilisateurs définissent des règles à travers une interface (Figure 6), selon quatre étapes :

- Ils choisissent le niveau sur lequel sera basé le nouveau niveau ;
- Ils expriment les conditions qui permettent le regroupement des instances du niveau de base ;
- Ils définissent les modalités des attributs qui composent le nouveau niveau ;
- Ils répètent les deux étapes précédentes jusqu'à ce que toutes les instances du niveau inférieur aient une instance correspondante dans le niveau à créer.

Algorithme 3 Vérification des contraintes

INPUT: niveau existant L, ensemble des règles $R = \{r_j, 1 \leq v\}$, body(r_j) la prémisse de la règle r_j, Tab un tableau

OUPUT: Intersection_constraint_checked,Union_constraint_checked

{Initialisation}

1: Intersection_constraint_checked=true
2: Union_constraint_checked=true

{Génération des requêtes}

3: **pour** j = 1 to v **faire**
4: Tab[j]='SELECT * FROM L where body(r_j)'
5: **fin pour**

{Vérification que l'intersection des ensembles induits par les règles pris deux à deux soit vide}

6: **pour** j = 1 to v-1 **faire**
7: Q=Tab[j] INTERSECT Tab[j+1]
8: **si** $Q \neq$ **alors**
9: Intersection_constraint_checked=false
10: **fin si**
11: **fin pour**

{Vérification que l'union des ensembles induits par les règles corresponde au niveau L}

12: Q=Tab[1]
13: **pour** j = 2 to v **faire**
14: Q=Q UNION Tab[j]
15: **fin pour**
16: **si** $Q \neq$SELECT * FROM L **alors**
17: Union_constraint_checked=false
18: **fin si**
19: **return** Intersection_constraint_checked
20: **return** Union_constraint_checked

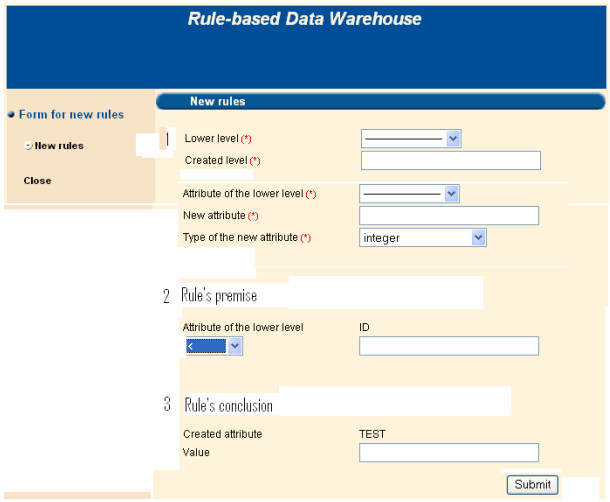

FIG. 6 – *Formulaire pour de nouvelles règles*

Deuxièmement, pour l'intégration des connaissances, les règles sont stockées à l'aide d'une table relationnelle. Les différents attributs de la table sont : (1) niveau inférieur, (2) clause *si*, (3) clause *alors*. Ainsi, les règles sont stockées dans le SGBD. Nous n'avons pas besoin d'un système devant gérer simultanément les règles et l'entrepôt de données stocké en relationnel. De plus, nous ne sommes ainsi pas limités par la taille de la mémoire pour exploiter les règles. Enfin, cela nous permet d'exploiter les possibilités offertes par le SGBD.

Troisièmement, l'évolution du modèle est faite selon les algorithmes d'ajout et d'insertion présentés précédemment, qui créent les niveaux de granularité et les liens d'agrégation adéquats avec les niveaux existants.

Quatrièmement, pour l'exploitation du nouveau modèle, notre prototype fournit le schéma d'analyse à la demande. En effet, comme le schéma évolue de façon continuelle, il est nécessaire que les utilisateurs puissent connaître quelles sont les possibilités d'analyse de l'entrepôt. Pour ce faire, un document XML (eXtensible Markup Language) est généré à partir de l'interrogation du méta modèle. L'avantage de XML est de permettre aux utilisateurs de naviguer à travers les hiérarchies du modèle et de décrire correctement les possibilités d'analyse. Ainsi, ce document va permettre aux utilisateurs de les aider dans leur choix d'analyse. Cette fonctionnalité exploite donc le standard qu'est XML, évitant le recours à un outil de visualisation utilisant un format propriétaire.

5.5 Cas d'application

Notre approche a été appliquée aux données bancaires qui concernent l'analyse du PNB. Les besoins portaient sur le type d'agences (étudiant, classique et non résident), la période (avant et après le rachat du Crédit Lyonnais par le Crédit Agricole en 2004), la possibilité de réaliser des analyses en fonction de l'âge des clients, par classes d'âge... Les règles exprimées par les utilisateurs et le schéma induit par ces règles sont présentés dans la Figure 7. Ainsi, à

partir de ce nouveau modèle, l'utilisateur pourra effectuer des analyses sur le PNB, en prenant en compte des niveaux de granularité générés par les règles comme `TYPE AGENCE`, `PERIODE`, `CLASSE AGE`...

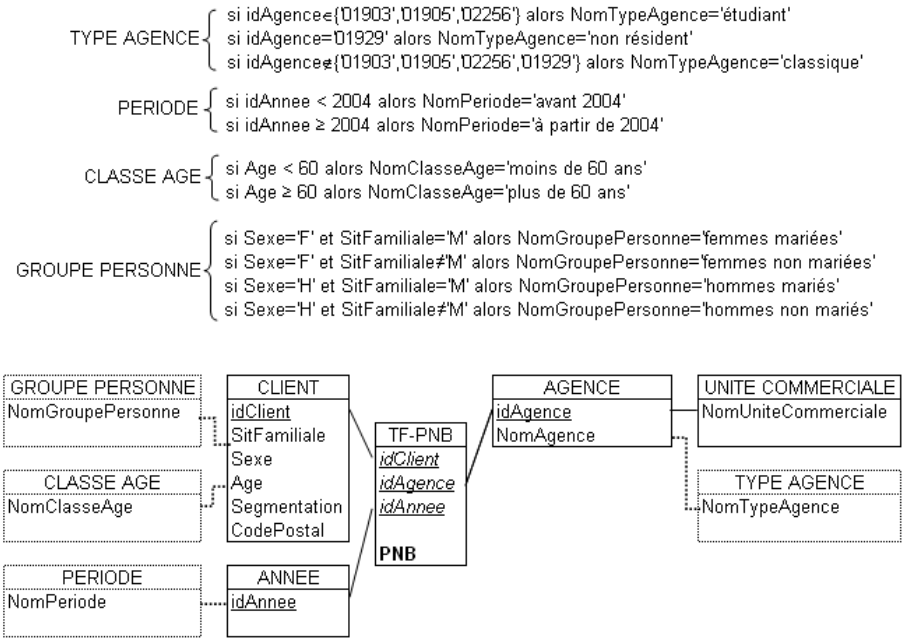

FIG. 7 – *Règles et schéma induit pour l'analyse du PNB*

6 Conclusion

Dans cet article, nous avons présenté une approche qui permet d'impliquer les utilisateurs dans l'évolution du schéma de l'entrepôt pour obtenir des analyses personnalisées, en fonction de leurs propres besoins d'analyse et de leurs propres connaissances du domaine. Nous avons présenté ici la mise en œuvre de cette approche, en proposant un méta modèle qui permet de gérer l'évolution du schéma, ainsi que les algorithmes d'évolution eux-mêmes et de validation des règles exprimées par les utilisateurs. La plateforme WEDriK que nous avons réalisée pour valider notre approche a permis d'appliquer notre modèle aux données de la banque LCL, et de montrer comment répondre à des besoins d'analyse personnalisés.

Ce travail ouvre différentes perspectives. Tout d'abord, nous devons réaliser une étude de performances et de complexité des algorithmes. Ensuite, les règles d'agrégation exprimées se basent sur les instances des dimensions. Pour faciliter la tâche d'acquisition des connaissances, il nous semble alors utile de permettre à l'utilisateur la définition d'une méta-règle qui permettrait de traduire la structure de l'agrégation, avant que cette méta-règle ne soit instanciée, pour permettre l'établissement des liens d'agrégation entre les instances des différents

niveaux de dimension. En outre, nous avons proposé une méthode pour valider les règles en utilisant les données disponibles de l'entrepôt. Ceci implique une validation répétée lors de la mise à jour des données, ce qui peut être mis en œuvre grâce aux triggers (procédures s'exécutant automatiquement dans le SGBD lorsqu'une action spécifique se produit). Néanmoins, nous souhaitons proposer une méthode de validation en se basant sur l'expression des règles. Par ailleurs, nous nous sommes placés dans le cas des hiérarchies classiques, en définissant des contraintes qui assurent qu'une instance dans un niveau inférieur correspond à une et une seule instance du niveau supérieur. Il serait intéressant de pouvoir traduire d'autres liens d'agrégation qui modélisent des situations réelles, tels que ceux évoqués par Malinowski et Zimányi (2004), en adaptant les contraintes posées sur les règles. D'autre part, notre approche permet de prendre en compte l'évolution des besoins d'analyse. Nous nous intéressons également à la prise en compte de l'évolution conjointe des sources de données et des besoins d'analyse. Enfin, il nous faut gérer la mise à jour des règles et étudier l'impact de celle-ci sur l'entrepôt de données.

Références

Bébel, B., J. Eder, C. Koncilia, T. Morzy, et R. Wrembel (2004). Creation and Management of Versions in Multiversion Data Warehouse. In *XIXth ACM Symposium on Applied Computing (SAC 04), Nicosia, Cyprus*, pp. 717–723. ACM Press.

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

Bellatreche, L., A. Giacometti, P. Marcel, H. Mouloudi, et D. Laurent (2005). A Personalization Framework for OLAP Queries. In *VIIIth ACM International Workshop on Data Warehousing and OLAP (DOLAP 05), Bremen, Germany*, pp. 9–18. ACM Press.

Blaschka, M., C. Sapia, et G. Höfling (1999). On Schema Evolution in Multidimensional Databases. In *Ist International Conference on Data Warehousing and Knowledge Discovery (DaWaK 99), Florence, Italy*, Volume 1676 of *LNCS*, pp. 153–164. Springer.

Bliujute, R., S. Saltenis, G. Slivinskas, et C. Jensen (1998). Systematic Change Management in Dimensional Data Warehousing. In *IIIrd International Baltic Workshop on Databases and Information Systems, Riga, Latvia*, pp. 27–41.

Bonifati, A., F. Cattaneo, S. Ceri, A. Fuggetta, et S. Paraboschi (2001). Designing Data Marts for Data Warehouses. *ACM Transactions on Software Engineering and Methodology 10*(4), 452–483.

Favre, C., F. Bentayeb, et O. Boussaïd (2006a). A Knowledge-driven Data Warehouse Model for Analysis Evolution. In *XIIIth International Conference on Concurrent Engineering : Research and Applications (CE 06), Antibes, France*, Volume 143 of *Frontiers in Artificial Intelligence and Applications*, pp. 271–278. IOS Press.

Favre, C., F. Bentayeb, et O. Boussaïd (2006b). A rule-based data warehouse model. In *23rd British National Conference on Databases (BNCOD 2006), Belfast, Northern Ireland*, Volume 4042 of *LNCS*. Springer.

Golfarelli, M., D. Maio, et S. Rizzi (1998). Conceptual Design of Data Warehouses from E/R Schemes. In *XXXIst Annual Hawaii International Conference on System Sciences (HICSS 98), Big Island, Hawaii, USA*, Volume 7, pp. 334–343.

Grange, J.-F. et B. Jouaneton (2006). Implémentation d'un entrepôt de données à base de règles, Projet Master 2 Informatique Décisionnelle et Statistique, Université Lyon 2.

Hurtado, C. A., A. O. Mendelzon, et A. A. Vaisman (1999). Updating OLAP Dimensions. In *IInd ACM International Workshop on Data Warehousing and OLAP (DOLAP 99), Kansas City, Missouri, USA*, pp. 60–66. ACM Press.

Khoualdia, M. et J. Malaret (2006). Construction d'un entrepôt de données bancaires, Projet Master 2 Informatique Décisionnelle et Statistique, Université Lyon 2.

Kimball, R. (1996). *The Data Warehouse Toolkit*. John Wiley & Sons.

Malinowski, E. et E. Zimányi (2004). OLAP Hierarchies : A Conceptual Perspective. In *XVIth International Conference on Advanced Information Systems Engineering (CAiSE 04), Riga, Latvia*, Volume 3084 of *LNCS*, pp. 477–491. Springer.

Mendelzon, A. O. et A. A. Vaisman (2000). Temporal Queries in OLAP. In *XXVIth International Conference on Very Large Data Bases (VLDB 00), Cairo, Egypt*, pp. 242–253. Morgan Kaufmann.

Morzy, T. et R. Wrembel (2004). On Querying Versions of Multiversion Data Warehouse. In *VIIth ACM International Workshop on Data Warehousing and OLAP (DOLAP 04), Washington, District of Columbia, USA*, pp. 92–101. ACM Press.

Nabli, A., A. Soussi, J. Feki, H. Ben-Abdallah, et F. Gargouri (2005). Towards an Automatic Data Mart Design. In *VIIth International Conference on Enterprise Information Systems (ICEIS 05), Miami, Florida, USA*, pp. 226–231.

Summary

We have conceived and deployed a data warehouse which collects data from several data sources concerning marketing operations. This work has been done with the collaboration of LCL, a french bank. The data warehouse schema was originally devised to answer needs common to the whole of users. But the same data warehouse schema cannot be used to answer specific users' needs. This article describes our approach to extend existing data warehouse schema by incorporating users' knowledge, therefore increasing the analysis capabilities of the data warehouse. Indeed we present our algorithms and our meta model. The users' knowledge are expressed in the form of if-then rules and are used to create new granularity levels in the dimension hierarchies. We implemented our approach with a platform named WEDriK (data Warehouse Evolution Driven by Knowledge), to allow LCL users to obtain personalized analysis on banking data.