Evolution et personnalisation des analyses dans les entrepôts de données

Une approche orientée utilisateur

Cécile Favre — Fadila Bentayeb — Omar Boussaid

Laboratoire ERIC
Université de Lyon - Campus Porte des Alpes
5 av. Pierre Mendès-France, 69676 Bron Cedex
{cfavre|bentayeb}@eric.univ-lyon2.fr, omar.boussaid@univ-lyon2.fr

RÉSUMÉ. *Dans le cadre d'une collaboration avec LCL-Le Crédit Lyonnais, nous avons conçu un entrepôt de données à partir des sources de données de l'établissement bancaire pour répondre aux besoins d'analyse des utilisateurs concernant les opérations marketing. Cependant, ces besoins sont amenés à évoluer rapidement. Dans cet article, nous proposons alors une approche originale d'évolution du modèle guidée par les utilisateurs, leur fournissant ainsi des analyses personnalisées. Notre approche est composée de quatre phases : (1) acquisition de la connaissance utilisateur sous la forme de règles d'agrégation ;(2) intégration de cette connaissance pour transformer ces règles en une table de mapping ; (3) mise à jour du modèle de l'entrepôt ; (4) analyse en ligne. Pour valider notre approche, nous avons implémenté un prototype, nommé WEDrik (data Warehouse Evolution Driven by Knowledge), sous le SGBD Oracle 10g. Nous avons appliqué notre approche sur des données bancaires de LCL pour montrer l'intérêt de celle-ci dans un contexte réel.*

ABSTRACT. *We designed a data warehouse with the data sources of LCL-Le Crédit Lyonnais meeting users' needs regarding marketing operations decision. However, the nature of the work of users implies that their requirements are often changing and do not reach a final state. In this paper, we propose an original and global approach to achieve a user-driven model evolution that provides answers to personalized analysis needs. Our approach is composed of four phases: (1) users' knowledge acquisition under the form of aggregation rules, (2) knowledge integration to transform rules into a mapping table, (3) data warehouse model update, and (4) on-line analysis. To validate our approach, we developed a prototype called WEDrik (data Warehouse Evolution Driven by Knowledge) within the Oracle 10g DBMS. Furthermore, we applied our approach on banking data of LCL in order to prove its interest.*

MOTS-CLÉS : *Entrepôt de données, évolution de modèle, personnalisation des analyses, règles d'agrégation, table de mapping.*

KEYWORDS: *Data warehouse, model evolution, analysis personnalization, aggregation rules, mapping table.*

1. Introduction

Dans le cadre d'une collaboration avec la banque LCL-Le Crédit Lyonnais à travers une Convention Industrielle de Formation par la Recherche (CIFRE) , l'objectif était de développer un système d'information décisionnel pour le marketing local. Le marketing local consiste à mener des actions commerciales pour répondre à des besoins de vente spécifiques lors de la création d'une agence par exemple. Il s'agit alors, pour les responsables commerciaux, de faire des demandes de ciblage de clients et d'utiliser ces ciblages pour optimiser les ventes des conseillers. Il est alors nécessaire de pouvoir mesurer l'intérêt de ces demandes marketing. Pour cela, nous avons sélectionné, extrait, intégré et consolidé les données pertinentes provenant de sources hétérogènes dans un entrepôt de données implémenté dans un contexte relationnel.

Un entrepôt de données est conçu pour répondre à un ensemble de besoins d'analyse recensés auprès des utilisateurs à un moment donné. Mais 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. Il est donc difficile de recenser de façon exhaustive les besoins d'analyse des utilisateurs et il est quasiment impossible de prévoir les besoins d'analyse futurs. Notre objectif est donc de fournir une solution pour répondre à l'évolution et à la personnalisation des besoins d'analyse dans un entrepôt de données existant.

Pour répondre à cet objectif, nous proposons que les utilisateurs soient impliqués dans le processus d'évolution du modèle de l'entrepôt puisque c'est ce modèle qui en détermine les possibilités d'analyse. Ces besoins d'analyse sont caractérisés par la volonté d'observer de nouveaux agrégats. Ainsi, l'évolution du modèle de l'entrepôt que l'on considère concerne l'évolution des hiérarchies de dimension, en particulier la création d'un nouveau niveau de granularité. Cette évolution de modèle se fait de façon incrémentale, au fur et à mesure que les besoins d'analyse émergent et qu'ils sont exprimés par les utilisateurs. En permettant à ces derniers de faire évoluer les possibilités d'analyse de l'entrepôt, nous augmentons ainsi leur intéraction avec le système décisionnel. Ils deviennent alors des acteurs impliqués dans le processus décisionnel au-delà de la navigation qu'ils font habituellement dans les analyses induites par le modèle.

Plus précisément, notre idée consiste à intégrer la connaissance spécifique des utilisateurs sous la forme de règle 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 le nouveau niveau de granularité. Nous montrons dans cet article comment exploiter ces règles d'agrégation pour réaliser non seulement l'évolution de schéma, mais également l'intégration des données nécessaires qui sont contenues dans ces règles, à travers la proposition d'une approche complète. Cette approche répond alors aux besoins d'analyse personnalisés des utilisateurs et fournit ainsi une approche d'évolution de modèle guidée par les utilisateurs.

Par ailleurs, nous proposons et discutons des algorithmes permettant l'évolution des hiérarchies de dimension. Ces algorithmes sont exploités dans la mise en œuvre de notre approche qui prend la forme de l'implémentation d'une plateforme baptisée WEDriK[1] (data Warehouse Evolution Driven by Knowledge) dans un contexte relationnel, au sein du Système de Gestion de Bases de Données (SGBD) Oracle 10g.

1. http ://eric.univ-lyon2.fr/∼cfavre

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. Nous discutons tout d'abord l'état de l'art concernant l'évolution de schéma dans les entrepôts de données dans la Section 2. Puis, nous présentons, dans la Section 3, notre approche d'évolution de hiérarchies de dimension guidée par les utilisateurs. Nous exposons ensuite la mise en œuvre de notre approche dans la Section 4. Dans la Section 5, nous déroulons étape par étape un exemple réel issu de l'étude de cas de LCL, et montrons comment utiliser notre approche à des fins d'analyse. Enfin, nous concluons et indiquons les perspectives de ce travail dans la Section 6.

2. État de l'art

Les travaux de recherche sur la conception de schéma des entrepôts de données témoignent aujourd'hui de la nécessité de prendre en compte à la fois les sources de données et les besoins d'analyse [NAB 05]. 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 alternative consiste à migrer les données d'un ancien schéma vers le plus récent en proposant des opérateurs pour faire évoluer le schéma [BLA 99, HUR 99]. Dans ce cas, un seul schéma est supporté, et les évolutions que le schéma subit ne sont donc pas conservées. Un autre courant s'incrit dans cette alternative de mise à jour, mais se base sur l'hypothèse que, conceptuellement, un entrepôt de données est un ensemble de vues matérialisées 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 [BEL 02]. Dans cette alternative, le fait de ne pas garder trace des évolutions peut induire des problèmes de cohérence du point de vue des analyses.

La deuxième alternative consiste, elle, à garder justement la trace des évolutions, en utilisant des étiquettes de validité temporelle. Ces étiquettes sont apposées soit au niveau des instances, soit au niveau des liens d'agrégation, ou encore au niveau des versions du schéma.

Le premier courant propose ainsi de gérer la temporalité des instances de dimensions [BLI 98] grâce à un schéma en étoile temporel. Le principe est d'omettre la dimension temps qui permet habituellement l'historisation des données et d'ajouter une étiquette temporelle au niveau de chacune des instances des tables de dimensions et de faits de l'entrepôt.

Le deuxième courant propose, quant à lui, de gérer la temporalité des liens d'agrégation [MEN 00]. Le chemin d'agrégation défini pour une instance le long d'une hiérarchie peut alors évoluer au cours du temps. Pour interroger ce modèle, les auteurs proposent un langage de requêtes nommé TOLAP.

Le troisième courant consiste à gérer différentes versions du modèle de l'entrepôt, chaque version étant valide pendant une durée donnée. Le modèle proposé dans [EDE 01] propose des fonctions de mise en correspondance qui permettent la conversion entre des versions de structures. Dans [BOD 03], les auteurs proposent une approche qui permet à l'utilisateur d'obtenir des analyses en fonction de ses besoins de comparaisons des données. En effet, le modèle proposé permet de choisir dans quelle version analyser les données (en temps consistant, dans une version antérieure, ou dans une nouvelle version). Dans [RAV 06],

un modèle multidimensionnel en temps consistant est proposé pour gérer des évolutions sur un modèle en constellation. Le versionnement permet également de répondre à des «*what-if analysis*», en créant des versions alternatives, en plus des version temporelles, pour simuler des changements de la réalité [BEB 04]. Différents travaux se sont par ailleurs focalisés sur la réalisation d'analyses prenant en compte différentes versions [GOL 06, MOR 04].

Dans le cadre de l'alternative de la modélisation temporelle, les évolutions du schéma sont donc bien conservées et assurent la cohérence des analyses. Mais ce type de solutions nécessite une réimplémentation des outils de chargement de données, d'analyse avec la nécessité d'étendre les langages de requêtes,... afin de gérer les particularités de ces modèles. Il est donc nécessaire dans ce cas de prévoir au moment de la conception comment vont être gérées les évolutions à venir.

Les approches présentées ici trouvent leur intérêt pour répondre au problème de l'évolution de schéma suite à une modification dans les sources de données. Ce sont des solutions 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. En effet, une fois l'entrepôt créé, les utilisateurs peuvent uniquement réaliser les analyses prévues par le modèle. De ce fait, aucune solution n'est apportée à l'émergence de nouveaux besoins d'analyse exprimés par les utilisateurs, et par conséquent au problème de la personnalisation des analyses.

Or la personnalisation dans les entrepôts de données devient un enjeu crucial, ouvrant ainsi une nouvelle voie de recherche. Les travaux dans ce domaine se focalisent en particulier sur la visualisation de données basée sur la modélisation des préférences utilisateurs et sur l'exploitation du concept de profil. Par exemple, dans [BEL 05], les auteurs proposent d'affiner les requêtes pour montrer une partie des données qui répond aux préférences utilisateurs. Dans ce cas, cette approche consiste à personnaliser l'analyse en sélectionnant une partie des informations qui intéressent l'utilisateur. Ainsi la personnalisation se situe au niveau de la visualisation, et pas au niveau du contenu même de l'analyse.

L'analyse dans les entrepôts de données est fortement liée aux hiérarchies de dimension qui sont définies. Lors de la conception de ces hiérarchies, l'approche naïve consiste à les faire émerger en fonction des besoins d'analyse et des sources de données qui sont à disposition. Pour rendre l'approche moins naïve, il a été proposé de définir des hiérarchies à un niveau conceptuel, puis logique, en les déterminant en fonction des relations de généralisation et d'agrégation de la modélisation UML (Unified Modeling Language) des besoins [AKO 01]. Les auteurs définissent leur approche en différentes étapes dont une doit inclure la confrontation avec les sources de données, mais ils ne présentent pas ce point, alors que cela constitue un problème majeur de la définition des hiérarchies.

Les hiérarchies doivent non seulement être stucturées mais également alimentées. L'a précédente approche ne répond qu'au premier aspect. Dans [MAZ 06], les auteurs proposent d'enrichir des hiérarchies de dimension à la fois pour la structure et les données, et ce de façon automatique. En partant du principe qu'une hiérarchie de dimension représente des relations sémantiques entre des valeurs, ils proposent d'exploiter les relations d'hypéronymie (is-a-kind-of) et de méronymie (is-a-part-of) de WordNet[2].

Dans notre approche nous proposons d'enrichir les possibilités d'analyse en ajoutant de l'information fournie par les utilisateurs eux-mêmes. Nous répondons ainsi au besoin de personnalisation des analyses. Dans un processus décisionnel dit centré utilisateur, nous étendons alors le rôle de celui-ci, qui ne va pas

2. http ://wordnet.princeton.edu/

seulement naviguer dans les données selon des chemins prédéfinis, mais aussi en créer de nouveaux pour réaliser de nouvelles analyses et ainsi prendre de meilleures décisions. L'évolution que nous proposons se situe à la fois au niveau de la structure et des données que nous puisons directement dans les connaissances utilisateurs, ce qui fait l'originalité de notre approche.

Avec notre approche, nous nous inscrivons dans une alternative de mise à jour du modèle (schéma et données) de l'entrepôt. Ce choix est motivé par plusieurs aspects. D'une part, notre approche permet ainsi d'enrichir un entrepôt de données existant, sans que cette évolution n'ait dû être prévue au moment de la conception avec des outils spécifiques. Nous pensons que c'est un atout dans un contexte où le versionnement n'est pas forcément encore présent dans la pratique (c'est le cas de l'entreprise avec laquelle nous travaillons), même si de nombreux travaux de recherche s'y intéressent. D'autre part, il s'avère que l'ajout de nouveaux niveaux de granularité ne remet pas en cause la cohérence des analyses existantes. Ainsi la maintenance de ce point de vue est limitée.

Dans un précédent travail [FAV 07], nous avons proposé un modèle formel d'entrepôt de données basé sur ces règles d'agrégation nommé *R-DW* (Rule-based Data Warehouse) pour permettre une évolution du schéma. Il est composé d'une partie «fixe», correspondant à un schéma qui répond à des besoins d'analyse communs à un ensemble d'utilisateurs ; cette partie est complétée par une partie «évolutive» définie par les règles d'agrégation répondant aux besoins d'analyse personnalisés. Nous avons inscrit ce modèle dans une architecture globale pour mettre en œuvre cette évolution. Dans cet article, nous étendons nos précédents travaux en formalisant précisément chacune des différentes phases de cette approche globale et en montrant son exploitation dans un contexte relationnel pour permettre l'intégration de nouveaux besoins d'analyse dans un entrepôt de données existant, afin de réaliser une évolution du modèle guidée par les utilisateurs.

3. Une approche orientée utilisateur pour l'évolution des hiérarchies de dimension

L'architecture globale de notre approche d'évolution des hiérarchies de dimension guidée par les utilisateurs est présentée dans la Figure 1. La première phase de cette approche est l'*acquisition* des connaissances utilisateurs sous la forme de règles «si-alors». Ensuite, dans la phase d'*intégration*, ces règles sont transformées en une table de mapping dans le SGBD. Ensuite, la phase d'*évolution* permet de créer le nouveau niveau de granularité, enrichissant une hiérarchie de dimension existante, ou en créant une nouvelle. Enfin, la phase d'*analyse* permet de réaliser des analyses sur le modèle de l'entrepôt mis à jour. Il s'agit d'un processus d'évolution incrémentale puisque chaque fois de nouveaux besoins viennent enrichir le modèle. Nous détaillons dans cette section les propositions faites concernant les trois premières phases.

3.1. *Phase d'acquisition*

Nous considérons ici une connaissance utilisateur spécifique qui permet de définir de nouvelles données agrégées dans l'entrepôt de données. Plus précisément, cette connaissance définit comment agréger des données d'un niveau existant vers un nouveau niveau qui est créé. Elle est représentée sous la forme de règles de type «si-alors». Ces règles ont l'avantage d'être très compréhensibles pour les utilisateurs parce qu'elles représentent l'information de façon explicite et sont particulièrement adaptées pour définir le lien d'agrégation qui existe entre deux niveaux de granularité successifs dans une hiérarchie de dimension.

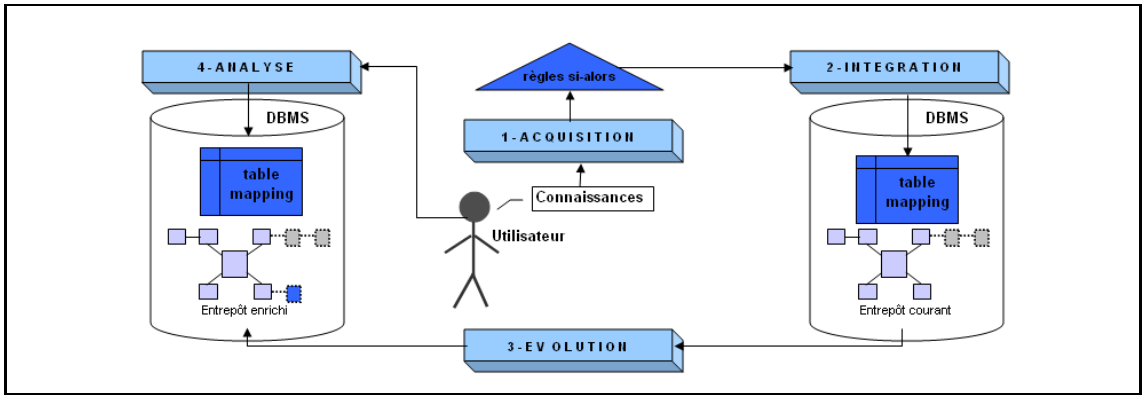


Figure 1. *Architecture pour l'évolution de modèle d'entrepôt de données guidée par les utilisateurs.*

Pour réaliser la phase d'acquisition des connaissances, nous proposons de définir d'abord une méta-règle d'agrégation qui représente la structure du lien d'agrégation. En effet, la méta-règle permet à l'utilisateur de définir, quel niveau de granularité il veut créer, les attributs caractérisant ce nouveau niveau, à partir de quel niveau de granularité il est créé et les attributs du niveau existant utilisés pour créer le lien d'agrégation.

Soit EL le niveau existant, $\{EA_i,\ i = 1..m\}$ le sous-ensemble des m attributs pris parmis les m' attributs existants de EL, sur lesquels vont porter les conditions ; GL le niveau généré et $\{GA_j,\ j = 1..n\}$ l'ensemble des n attributs générés de GL. La méta-règle est définie de la façon suivante :

$$MR : \textit{if } ConditionOn(EL, \{EA_i,\ i = 1..m\}) \textit{ then } GenerateValue(GL, \{GA_j,\ j = 1..n\})$$

Une fois la structure du lien d'agrégation définie, il faut que le lien soit réalisé au niveau des instances elles-mêmes. Il s'agit donc de définir pour chaque instance du niveau existant, quelle est l'instance du niveau supérieur créé correspondant. L'idée est donc d'exprimer des règles mettant en correspondance les instances, en regroupant des instances du niveau inférieur et en les rapprochant de l'instance du niveau supérieur. Ainsi, pour instancier la méta-règle, l'utilisateur définit une règle par instance du niveau généré afin d'y associer les instances du niveau existant.

Comme définie dans [FAV 07], une règle d'agrégation est basée sur un ensemble $\mathcal{T}$ de z termes de règles, notés RT_x, tel que :

$$\mathcal{T} = \{RT_x, 1 \leq x \leq z\} = \{EA_x\ op_x\ \{ens|val\}_x\}$$

où EA_x est un attribut du niveau inférieur, op_x est un opérateur soit relationnel ($=$, $<$, $>$, $\leq$, $\geq$, $\neq$, ...), soit ensembliste ($\in$, $\notin$, ...) et $\{ens|val\}_x$ correspond respectivement soit à un ensemble de valeurs, soit à une valeur finie, toutes ces valeurs appartenant au domaine de définition de EA_x.

Nous notons ici $c_x = op_x\ \{ens|val\}_x$ la condition portée sur l'attribut EA_x incluant l'opérateur et la valeur ou l'ensemble de valeurs. Soit q le nombre d'instances de GL, un ensemble $R = \{r_d, d = 1..q\}$ de q règles d'agrégation doit être défini. Ainsi, la condition se rapportant à l'attribut EA_i dans la règle r_d est notée c_{di}. Par ailleurs, nous notons v_{dj} la valeur attribuée à l'attribut généré GA_j dans la règle r_d.

Une règle d'agrégation est une règle de type «*si-alors*». La conclusion de la règle (clause «alors») définit les valeurs des attributs du niveau généré GL. La prémisse de la règle (clause «si») est basée sur une

composition de conjonctions des termes de règles. Une règle r_d qui instancie la méta-règle MR est notée :

$r_d : if\ RT_1\ AND\ ...\ AND\ RT_x\ AND\ ...\ AND\ RT_z$
$then\ GA_1 = v_{d1}\ AND\ ...\ AND\ GA_j = v_{dj}\ AND\ ...\ AND\ GA_n = v_{dn}$

Nous avons donc en utilisant le concept de condition :

$r_d : if\ EA_1\ c_{d1}\ AND\ ...\ AND\ EA_i\ c_{di}\ AND\ ...\ AND\ EA_m\ c_{dm}$
$then\ GA_1 = v_{d1}\ AND\ ...\ AND\ GA_j = v_{dj}\ AND\ ...\ AND\ GA_n = v_{dn}$

Il existe différents types de lien d'agrégation au sein d'une hiérarchie de dimension [MAL 04]. Nous considérons ici le cas classique dans lequel une instance d'un niveau inférieur correspond à une instance du niveau supérieur, et toutes les instances du niveau inférieur ont une instance correspondante dans le niveau supérieur. Ainsi, les règles d'agrégation définissent une partition des instances du niveau existant, et chaque classe de cette partition est associée à une instance du niveau créé (Figure 2). Nous devons alors vérifier que les règles d'agrégation satisfont cette définition. Pour réaliser cette tâche, nous proposons d'utiliser les outils offerts par le SGBD. Ainsi, nous exploitons la représentation relationnelle des règles d'agrégation, en l'occurence une table de mapping. La transformation est réalisée durant la phase d'intégration.

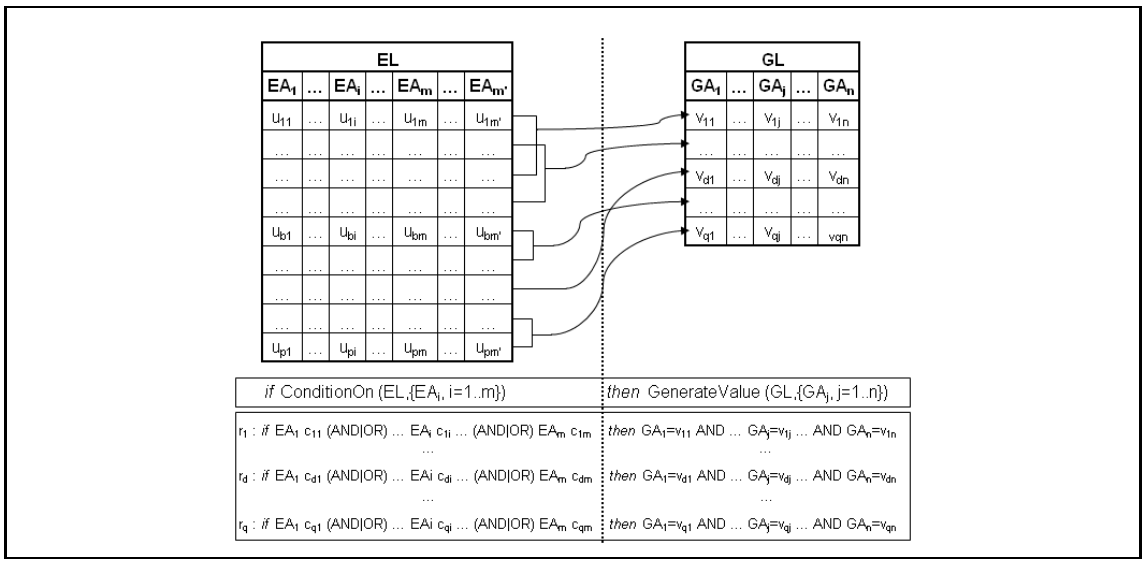


Figure 2. *Partition induite par les règles d'agrégation.*

3.2. *Phase d'intégration*

La phase d'intégration consiste à transformer les règles «si-alors» en une table de mapping et à la stocker dans le SGBD au moyen d'une structure relationnelle. Pour un ensemble donné de règles qui définit un nouveau niveau de granularité, nous y associons une table de mapping (Figure 3). Pour réaliser cette transformation, nous exploitons dans un premier temps la méta-règle d'agrégation pour définir la structure de la table de mapping, et dans un deuxième temps, nous utilisons les règles d'agrégation elles-mêmes pour

insérer les enregistrements correspondants dans la table de mapping. Etant dans un contexte relationnel, chaque niveau de granularité correspond à une table relationnelle.

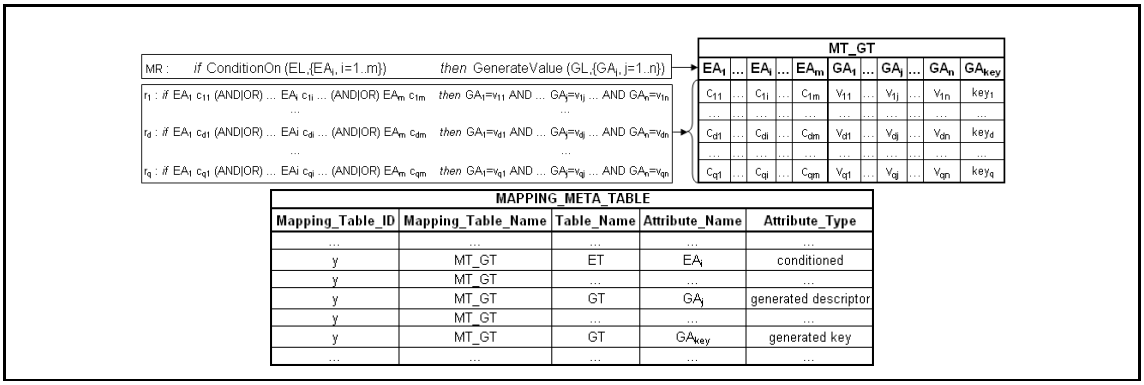


Figure 3. *Intégration des règles.*

Soient ET (resp. GT) la table existante (resp. générée), correspondant à EL (resp. GL) dans un contexte logique. Soient $\{EA_i,\ i = 1..m\}$ l'ensemble des m attributs de ET, $\{GA_j,\ j = 1..n\}$ l'ensemble des n attributs générés de GT et GA_{key} l'attribut ajouté automatiquement qui sera la clé primaire de GT. La table de mapping MT_GT construite à partir de la méta-règle MR correspond à la relation suivante :

$$MT\_GT(EA_1,\ ...,\ EA_i,\ ...,\ EA_m,\ GA_1,\ ...,\ GA_j,\ ...,\ GA_n,\ GA_{key})$$

Les règles d'agrégation permettent d'insérer dans cette table de mapping les conditions sur les attributs $\{EA_i,\ i = 1..m\}$, et les valeurs des attributs générés correspondants $GA_j,\ j = 1..n$. Pour l'attribut GA_{key}, la valeur est ajoutée de façon incrémentale. Ainsi, pour chaque enregistrement d de la table de mapping, on a les conditions appliquées aux attributs ($c_{di},\ i = 1..m$) définissant quelles instances de la table ET sont concernées, et à quelle instance elles correspondent dans GT, cette instance étant caractérisée par les valeurs des attributs de la table GT ($v_{dj},\ j = 1..n$).

En outre, nous définissons une méta-table de mapping pour rassembler les informations sur les différentes tables de mapping. En effet, leur structure peut différer en fonction du nombre d'attributs supportant des conditions et du nombre d'attributs générés caractérisant le nouveau niveau. La structure de cette méta-table est représentée par la relation suivante (une illustration est fournie dans la Figure 3).

$$MAPPING\_META\_TABLE(Mapping\_Table\_ID, Mapping\_Table\_Name, Table\_Name, Attribute\_Name, Attribute\_Type)$$

où $Mapping\_Table\_ID$ et $Mapping\_Table\_Name$ correspondent respectivement à l'identifiant et au nom de la table de mapping ; $Attribute\_Name$ et $Table\_Name$ caractérisent l'attribut impliqué dans le processus d'évolution et sa table d'appartenance ; $Attribute\_Type$ fournit le rôle de cet attribut. Plus précisément, ce dernier attribut a trois modalités : «$conditioned$» s'il apparaît dans la clause *si* et «$generated\ descriptor$» ou «$generated\ key$» s'il est dans la clause *alors* selon qu'il s'agit d'un attribut clé ou non. Notons que même si un attribut a le rôle de «$generated\ descriptor$» ou «$generated\ key$» dans une table de mapping, il peut avoir le rôle «$conditioned$» dans une autre table de mapping pour servir à la construction d'un autre niveau. Ainsi, il est possible de construire deux niveaux successifs dans une hiérarchie par notre méthode.

3.3. *Phase d'évolution*

L'évolution du modèle de l'entrepôt de données consiste, dans notre approche, à mettre à jour les hiérarchies de dimension d'un entrepôt courant en créant de nouveaux niveaux de granularité. Il s'agit donc non seulement de créer la structure de ce niveau mais également de l'alimenter avec les données requises. Le niveau 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, nous parlons alors respectivement d'ajout ou d'insertion d'un niveau. Ce choix est fait par l'utilisateur. Il s'agit de savoir si cela a un sens d'agréger les données du niveau que l'on crée vers un niveau déjà existant. Néanmoins, dans les deux cas, les règles exprimées par l'utilisateur sont les mêmes, c'est à dire qu'elles représentent uniquement le lien d'agrégation entre le niveau inférieur et le niveau créé.

Le processus d'évolution (Figure 4) est réalisé à partir de la table de mapping dédiée à la création du niveau et des instances qui lui sont associées dans la méta-table. Quel que soit le type d'évolution appliqué, la structure de la table GT est donc créée selon les attributs de la méta-table $MAPPING\_META\_TABLE$ qui comportent la mention «$generated\ key$», i.e. GA_{key}, pour la clé de la table et «$generated\ descriptor$» pour les autres attributs $\{GA_j,\ j = 1..n\}$. Les instances de cette table sont insérées selon le contenu de la table de mapping MT_GT, en l'occurence les valeurs de la clé $\{key_d,\ d = 1..q\}$ et les valeurs des autres attributs $\{v_{dj},\ d = 1..q,\ j = 1..n\}$. La structure de la table ET est mise à jour avec l'ajout de l'attribut GA_{key} qui constitue une clé étrangère référençant la table GT (coloration en gris clair dans la Figure 4). La valeur de cet attribut est mise à jour pour chacune des instances de la table en fonction des conditions exprimées sur les attributs de ET dans la table de mapping.

Dans le cas où GT est inséré entre ET et $ET2$, la table GT comporte également un attribut L qui constituera la clé étrangère référençant l'attribut L de $ET2$ pour établir le lien (liaison en pointillé sur la Figure 4). Cet attribut est alimenté grâce à la valeur qu'il a dans la table ET (coloration en gris foncé dans la Figure 4). Ainsi, dans le cas d'une insertion de niveau, le lien entre le niveau créé et le niveau supérieur est défini de façon automatique.

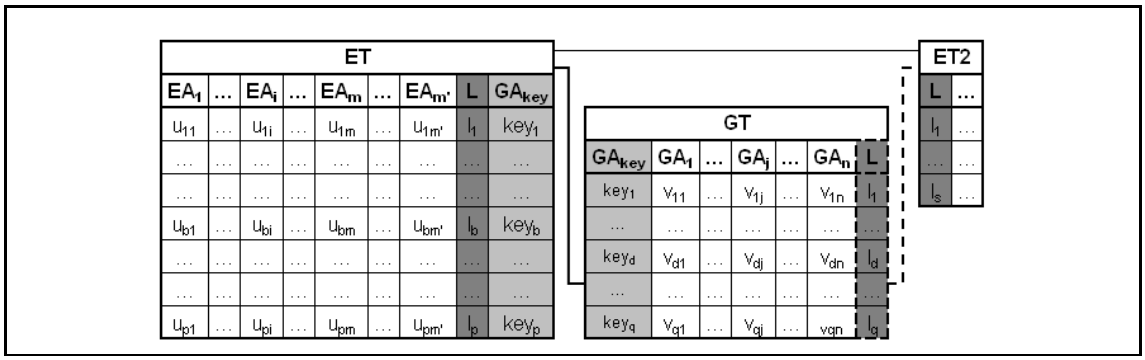


Figure 4. *Evolution de la hiérarchie de dimension.*

C'est cette phase d'évolution qui permet de répondre aux besoins d'analyse émergents. En effet, elle concrétise la possibilité d'obtenir de nouveaux agrégats grâce à la création du nouveau niveau de granularité et de son alimentation avec des données qui transcrivent les connaissances utilisateurs exprimées dans

la méta-règle et dans les règles. Ainsi le modèle évolue de façon incrémentale par ajout de niveau de granularité dans les hiérarchies de dimension.

Les limites de cette évolution se situent à deux niveaux : sémantique et physique. Au niveau sémantique, étant donné que les niveaux correspondent sémantiquement à des niveaux d'agrégation, il est possible que l'agrégation au dessus d'un niveau donné n'ait pas de sens. Au niveau physique, il y a l'aspect capacité de stockage et l'aspect temps de réponse. Le stockage n'est pas forcément très limitatif puisque, par définition, les niveaux rajoutés en fin de hiérarchie comportent successivement de moins en moins d'instances. Néanmoins, le temps de réponse peut constituer un réel problème. En effet, dans ce contexte relationnel, l'opération d'ajout d'un niveau correspond à une opération de jointure supplémentaire lors de l'analyse pour ce niveau. Il s'agit alors de mettre en œuvre des stratégies d'optimisation telles que les structures d'indexation ou de précalcul des agrégats selon l'utilisation importante ou non du nouveau niveau.

4. Mise en œuvre de l'approche d'évolution

Pour valider notre approche, nous avons développé une plateforme nommée WEDriK (data Warehouse Evolution Driven by Knowledge) sous le SGBD Oracle 10g avec une interface Web permettant aux utilisateurs d'interagir avec l'entrepôt de données pour la saisie des règles et l'analyse. Pour réaliser la mise à jour du modèle de l'entrepôt telle que nous l'avons présentée dans la section précédente, nous avons proposé un ensemble d'algorithmes qui permettent de réaliser l'ensemble des phases : transformation des règles en une table de mapping, vérification de la validité des règles selon la propriété de partition, ajout et insertion d'un niveau de granularité. Nous nous focalisons dans cette section à ces deux derniers algorithmes qui permettent de réaliser l'évolution souhaitée du modèle, en générant les requêtes nécessaires.

4.1. *Algorithme d'ajout de niveau de granularité*

Pour créer une nouvelle table GT, à partir d'une table existante ET, nous avons conçu un algorithme qui permet l'ajout d'une table à la fin d'une hiérarchie en exploitant une table de mapping MT (Algorithme 1). L'opération s'effectue en trois étapes : (1) créer la table GT, (2) insérer dans GT les valeurs nécessaires en fonction de la table de mapping, (3) lier la table GT avec la table existante ET.

4.2. *Algorithme d'insertion de niveau de granularité*

Afin d'insérer une nouvelle table GT entre deux tables existantes (ET et $ET2$), nous proposons l'Algorithme 2 qui permet la création et l'insertion d'une table entre deux autres, en prenant en compte une table de mapping MT, qui contient uniquement le lien entre la table inférieur ET et la table générée GT. Les étapes sont les mêmes que pour l'ajout d'un niveau. Ensuite, le lien d'agrégation est déterminé de façon automatique entre GT et $ET2$, en l'inférant à partir du lien existant entre ET et $ET2$. Une fois ce nouveau lien établi, si le lien entre ET et $ET2$ n'a plus lieu d'être, il est possible de le supprimer.

Rappelons que dans le cas de l'insertion (qui est décidée par l'utilisateur), il faut que l'agrégation du niveau créé vers un niveau déjà existant ait un sens. A défaut de pouvoir le vérifier, nous réalisons une vérification des incohérences possibles au niveau des instances dans la phase d'intégration des règles.

Algorithm 1 Pseudo-code pour ajouter un nouveau niveau de granularité

INPUT: (1) table existante ET, (2) table de mapping MT, (3) $\{EA_i,\ i = 1..m\}$ l'ensemble des m attributs de MT qui contiennent les conditions sur les attributs, (4) $\{GA_j,\ j = 1..n\}$ l'ensemble des n attributs générés de MT qui contiennent les valeurs des attributs générés, (5) GA_{key} l'attribut clé primaire de GT

OUPUT: table générée GT

{Création de la table GT}

1: CREATE TABLE GT (GA_{key}, $\{GA_j\}$);

{Mise à jour de la structure de ET pour la lier à GT}

2: ALTER TABLE ET ADD (GA_{key});

{Alimentation de GT, liaison entre ET et GT}

3: **for all** (tuple t of MT) **do**

4: INSERT INTO GT VALUES (GA_{key},$\{GA_j\}$);

5: UPDATE ET SET $ET.GA_{key} = GT.GA_{key}$ WHERE $\{EA_i\}$;

6: **end for**

7: **return** GT

Algorithm 2 Pseudo-code pour insérer un nouveau niveau de granularité

INPUT: table existante inférieure ET, table existante supérieure $ET2$, table de mapping MT, $\{EA_i,\ i = 1..m\}$ l'ensemble des m attributs de MT qui contiennent les conditions sur les attributs, $\{GA_j,\ j = 1..n\}$ l'ensemble des n attributs générés de MT qui contiennent les valeurs des attributs générés, GA_{key} l'attribut clé primaire de GT, L l'attribut liant ET à $ET2$

OUPUT: table générée GT

{Création de la table GT}

1: CREATE TABLE GT (GA_{key}, $\{GA_j\}$, L);

{Mise à jour de la structure de ET pour la lier à GT}

2: ALTER TABLE ET ADD (GA_{key});

{Alimentation de GT, liaison entre ET et GT, liaison entre GT et $ET2$}

3: **for all** (tuple of MT) **do**

4: INSERT INTO GT VALUES (GA_{key},$\{GA_j\}$);

5: UPDATE ET SET $ET.GA_{key} = GT.GA_{key}$ WHERE $\{EA_i\}$;

6: UPDATE GT SET L=(SELECT DISTINCT L FROM ET WHERE $ET.GA_{key}$=$GT.GA_{key}$ AND $\{EA_i\}$);

7: **end for**

8: **return** GT

4.3. *Complexité des algorithmes*

Pour étudier la complexité des algorithmes, nous nous focalisons sur les opérations d'écriture et de lecture. Nous supposons ici que la création de la structure d'une table ou que la modification d'une structure de table existante par l'ajout d'un attribut sont négligeables. La complexité est mesurée en terme de quantité d'opérations de lecture ou d'écriture d'un attribut pour un enregistrement donné.

Pour l'ajout d'un niveau de granularité, il y a deux types d'opérations d'écriture : les insertions dans la nouvelle table GT et les mises à jour dans la table existante ET pour créer le lien entre ces deux tables. La complexité de ces opérations est de $\omega((p+n+1)\times q+p)$. En effet, dans la table GT, il y a $n+1$ attributs et q enregistrements. Donc il y a en nombre d'opérations d'écriture (insertion des valeurs) : $(n+1)\times q$. Pour faire le lien entre ET et GT, la valeur de l'attribut représentant la clé étrangère est mise à jour pour chaque enregistrement de la table ET, soit concernant le nombre d'opérations d'écriture : p. Ainsi, le total

des opérations d'écriture est de : $p + (n + 1) \times q$. Pour les opérations de lecture, afin de mettre à jour la valeur de la clé étrangère, pour chacun des q enregistrements de la méta-table, les p enregistrements de la table ET sont parcourus, ainsi on obtient pour les opérations de lecture : $q \times p$.

L'insertion d'un niveau de granularité entre deux niveaux existants comprend, en plus des opérations décrites précédemment dans le cas de l'ajout, une opération de mise à jour pour lier la table générée GT avec la table existante de niveau supérieur $ET2$. La complexité totale de ces opérations est de $\omega((p + n + 2) \times q + p)$. En effet, pour mettre à jour l'attribut représentant la clé étrangère dans le niveau créé GT, il y a q opérations. L'opération de lecture qui y est associée est assimilable à celle qui est réalisée lors de l'ajout et n'augmente donc pas la complexité.

Ainsi notre approche est plus pertinente lorsque q (nombre d'enregistrements dans la table générée GT) et p (nombre d'enregistrements dans la table existante ET) sont petits du point de vue de la complexité, ce qui est le cas pour des niveaux de granularité élevé. C'est également le cas d'un point de vue pratique, puisque l'utilisateur doit définir une règle d'agrégation par instance du niveau créé. Ainsi, si q est élevé, l'utilisateur doit définir un grand nombre de règles, ce qui rend l'approche très fastidieuse. Notre approche trouve donc son utilité pour créer de nouveaux agrégats lorsque les niveaux d'agrégation créés comportent peu d'instances de regroupement.

5. Cas d'étude de la banque LCL : un exemple illustratif

5.1. *Présentation du cas*

Pour illustrer l'usage et le fonctionnement de WEDrik, nous avons sélectionné un exemple simplifié issu du cas réel de la banque LCL. Pour étudier l'impact des demandes marketing, nous avons défini un modèle d'entrepôt en constellation dont nous considérons ici un extrait présenté dans la Figure 5 permettant d'étudier le NBI (Net Banking Income). Le NBI correspond à ce que rapporte un client à l'établissement bancaire. C'est une mesure qui est analysée selon les dimensions CUSTOMER (client), AGENCY (agence) et YEAR (année). Il est possible d'agréger les données selon le niveau COMMERCIAL_UNIT (unité commerciale), qui est un regroupement d'agences selon leur position géographique, tel que le montre le schéma de la dimension AGENCY de la Figure 5a.

Supposons qu'un utilisateur veuille analyser le NBI 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 propre connaissance sur les types d'agence afin de mettre à jour la hiérarchie de la dimension agence en ajoutant le niveau AGENCY_TYPE au dessus du niveau AGENCY selon le schéma de la Figure 5b. Afin de réaliser une analyse du NBI en fonction du type d'agence, notre approche est utilisée et suit les étapes décrites par la suite.

5.2. *Déroulement de notre approche*

Considérons les extraits de la table de dimension AGENCY et de la table de faits TF-NBI de la Figure 6.

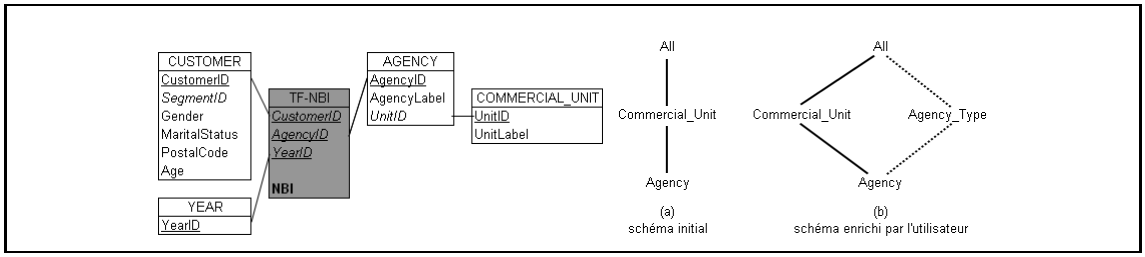


Figure 5. *Schéma de l'entrepôt de données pour l'analyse du NBI et schémas de la dimension AGENCY .*

AGENCY

AgencyID	*AgencyLabel*	...
01000	LYON REPUBLIQUE	...
01029	LYON GERLAND	...
01903	LYON III UNIVERSITE	...
01905	LYON LA DOUA	...
01929	AGENCE INTERNATIONALE	...
02256	CLERMONT LAFAYETTE (Etud)	...
02600	GRENOBLE	...
03730	ANNONAY	...

TF-NBI

AgencyID	*CustomerID*	*YearID*	*NBI (€)*
01000	1	2006	2000
01903	2	2006	1000
01000	3	2006	4000
03730	4	2006	2000
01929	5	2006	5000
01000	6	2006	2000
01029	7	2006	1000
02600	8	2006	1000
01905	9	2006	2000
01929	10	2006	3000

Figure 6. *Extraits des tables pour l'exemple du NBI.*

5.2.1. *Phase d'acquisition.*

Lors de la phase d'*acquisition*, l'utilisateur définit la méta-règle d'agrégation MR pour spécifier la structure du lien d'agrégation pour le type d'agence. Elle exprime donc le fait que le niveau AGENCY_TYPE va être caractérisé par l'attribut AgencyTypeLabel et il sera créé au dessus du niveau AGENCY ; les regroupements des instances de AGENCY se baseront sur des conditions exprimées sur l'attribut AgencyID.

if ConditionOn(AGENCY,{AgencyID})
then GenerateValue(AGENCY_TYPE,{AgencyTypeLabel})

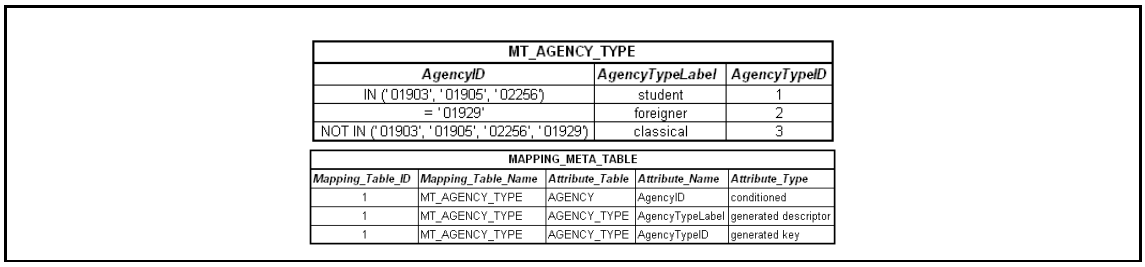
Puis l'utilisateur définit les règles d'agrégation qui instancient la méta-règle pour créer les différents types d'agence. Ainsi, il définit une règle pour chaque type d'agence, en exprimant à chaque fois la condition sur l'attribut AgencyID et en affectant à l'attribut généré AgencyTypeLabel sa valeur correspondante :

(R1) *if* AgencyID $\in$ {'01903','01905','02256'} *then* AgencyTypeLabel='student'
(R2) *if* AgencyID = '01929' *then* AgencyTypeLabel='foreigner'
(R3) *if* AgencyID $\notin$ {'01903','01905','02256','01929'} *then* AgencyTypeLabel='classical'

5.2.2. *Phase d'intégration.*

La phase d'*intégration* exploite la méta-règle et les règles d'agrégation R1, R2 et R3 pour générer la table de mapping MT_AGENCY_TYPE et les informations concernant cette table sont insérées dans la méta-table MAPPING_META_TABLE (Figure 7). Ainsi la table de mapping contient les attributs AgencyID, AgencyTypeLabel respectivement, que l'on retrouve dans la méta-table MAPPING_META_TABLE et dont les

rôles sont respectivement «*conditioned*» et «*generated descriptor*». La clé AgencyTypeID est rajoutée automatiquement et figure donc dans la méta-table avec le rôle «*generated key*».



MT_AGENCY_TYPE

AgencyID	*AgencyTypeLabel*	*AgencyTypeID*
IN ('01903', '01905', '02256')	student	1
= '01929'	foreigner	2
NOT IN ('01903', '01905', '02256', '01929')	classical	3

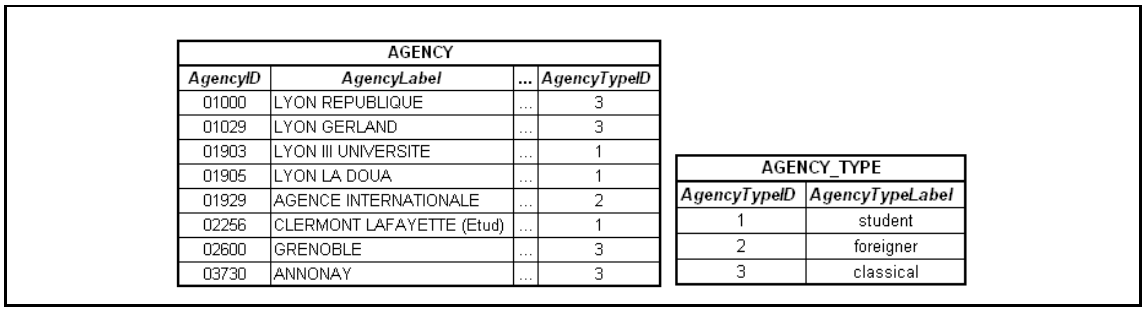
MAPPING_META_TABLE

Mapping_Table_ID	*Mapping_Table_Name*	*Attribute_Table*	*Attribute_Name*	*Attribute_Type*
1	MT_AGENCY_TYPE	AGENCY	AgencyID	conditioned
1	MT_AGENCY_TYPE	AGENCY_TYPE	AgencyTypeLabel	generated descriptor
1	MT_AGENCY_TYPE	AGENCY_TYPE	AgencyTypeID	generated key

Figure 7. *Table et méta-table de mapping pour le niveau* AGENCY_TYPE.

5.2.3. *Phase d'évolution.*

L'évolution choisie par l'utilisateur correspond à un ajout. En effet, il n'y a pas de lien possible d'agrégation entre le type d'agence et l'unité commerciale qui correspond à un regroupement géographique des agences. La phase d'*évolution* qui suit permet donc d'une part de créer et d'alimenter la table AGENCY_TYPE ; et d'autre part de mettre à jour la table AGENCY pour la relier à la table AGENCY_TYPE, avec l'ajout de l'attribut AgencyTypeID et la mise à jour de ses valeurs (Figure 8).



AGENCY

AgencyID	*AgencyLabel*	...	*AgencyTypeID*
01000	LYON REPUBLIQUE	...	3
01029	LYON GERLAND	...	3
01903	LYON III UNIVERSITE	...	1
01905	LYON LA DOUA	...	1
01929	AGENCE INTERNATIONALE	...	2
02256	CLERMONT LAFAYETTE (Etud)	...	1
02600	GRENOBLE	...	3
03730	ANNONAY	...	3

AGENCY_TYPE

AgencyTypeID	*AgencyTypeLabel*
1	student
2	foreigner
3	classical

Figure 8. *La table* AGENCY_TYPE *créée et la mise à jour de la table* AGENCY.

5.2.4. *Phase d'analyse.*

Finalement, la phase d'*analyse* permet d'exploiter le modèle enrichi d'un nouvel axe d'analyse. Classiquement, dans un environnement d'analyse en ligne, les requêtes décisionnelles consistent en la création d'agrégats en fonction des niveaux de granularité dans les dimensions. En effet, étant donné un modèle, le processus d'analyse permet de résumer les données en exploitant (1) des opérateurs d'agrégation tels que SUM et (2) des clauses de regroupement telles que GROUP BY. Dans notre cas, l'utilisateur souhaitait une analyse sur la somme du NBI en fonction des types d'agence qu'il avait définis. La requête correspondante et le résultat de cette requête sont présentés dans la Figure 9.

Nous avons présenté ici le déroulement d'un exemple relativement simple de la connaissance qui peut être exploitée pour créer un niveau d'agrégation supplémentaire. Notons que les règles offrent un pouvoir

```
SELECT AgencyTypeLabel, SUM(NBI)
FROM TF-NBI, AGENCY, AGENCY_TYPE
WHERE TF-NBI.AgencyID=AGENCY.AgencyID
AND AGENCY.AgencyTypeID=AGENCY_TYPE.AgencyTypeID
GROUP BY AgencyTypeLabel ;
```

AgenyTypeLabel	NBI (€)
student	3000
foreigner	8000
classical	12000

Figure 9. *Requête et résultat de l'analyse du NBI en fonction du type d'agence.*

d'expression assez fort sur les conditions permettant de définir des groupes d'instances dans le niveau sur lequel on se base pour créer le nouveau niveau. La définition de la méta-règle permet de rendre la saisie des règles très facile grâce à une interface conviviale qui guide cette saisie.

6. Conclusion

Dans cet article, nous avons proposé une approche qui permet d'impliquer les utilisateurs dans l'évolution du modèle de l'entrepôt pour obtenir des analyses personnalisées, dans un contexte relationnel. Cette approche est fondée sur un modèle d'entrepôt de données qui évolue de façon incrémentale au gré des besoins d'analyse émis par les utilisateurs. Ces besoins ont la caractéristique de prendre en compte les connaissances des utilisateurs sous forme de règles d'agrégation. Ces règles subissent un processus de transformation, dont nous avons formalisé chacune des étapes, pour finalement mettre à jour les hiérarchies de dimension en ajoutant ou en insérant un niveau de granularité, procurant ainsi aux utilisateurs de nouvelles possibilités d'analyse. La plateforme que nous avons réalisée a permis d'appliquer notre approche sur les données de la banque LCL pour répondre à des besoins exprimés par des responsables commerciaux et de montrer ainsi l'intérêt de notre approche dans un contexte réel d'application.

Ce travail ouvre différentes perspectives. D'une part, nous voulons prendre en compte l'évolution des connaissances elles-mêmes, en réfléchissant aux stratégies de mise à jour et de versionnement au niveau des règles. De plus, nous voulons traiter d'autres types de hiérarchie [MAL 04], en adaptant les contraintes posées sur les règles. En outre, lorsque le domaine des valeurs à considérer dans les règles est vaste, la saisie peut devenir fastidieuse. Nous envisageons d'étendre l'expression des connaissances afin de permettre par exemple l'application de fonctions de transformation sur les données existantes pour déterminer les nouvelles données. Par ailleurs, il serait intéressant d'adapter cette approche au niveau de la phase d'extraction, transformation et chargement des données, permettant à l'utilisateur d'exploiter directement les données provenant des sources. Enfin, nous nous intéressons également à la prise en compte de l'évolution conjointe des sources de données et des besoins d'analyse pour assurer la cohérence du modèle de l'entrepôt.

7. Bibliographie

[AKO 01] AKOKA J., COMYN-WATTIAU I., PRAT N., « Dimension Hierarchies Design from UML Generalizations and Aggregations », *XXth International Conference on Conceptual Modeling (ER 01), Yokohama, Japan*, vol. 2224 de *LNCS*, Springer, 2001, p. 442–455.

[BEB 04] BEBEL B., EDER J., KONCILIA C., MORZY T., WREMBEL R., « Creation and Management of Versions in Multiversion Data Warehouse », *XIXth ACM Symposium on Applied Computing (SAC 04), Nicosia, Cyprus*, ACM

Press, 2004, p. 717–723.

[BEL 02] BELLAHSENE Z., « Schema Evolution in Data Warehouses », *Knowledge and Information Systems*, vol. 4, n° 3, 2002, p. 283–304.

[BEL 05] BELLATRECHE L., GIACOMETTI A., MARCEL P., MOULOUDI H., LAURENT D., « A Personalization Framework for OLAP Queries », *VIIIth ACM International Workshop on Data Warehousing and OLAP (DOLAP 05), Bremen, Germany*, ACM Press, 2005, p. 9–18.

[BLA 99] BLASCHKA M., SAPIA C., HÖFLING G., « On Schema Evolution in Multidimensional Databases », *Ist International Conference on Data Warehousing and Knowledge Discovery (DaWaK 99), Florence, Italy*, vol. 1676 de *LNCS*, Springer, 1999, p. 153–164.

[BLI 98] BLIUJUTE R., SALTENIS S., SLIVINSKAS G., JENSEN C., « Systematic Change Management in Dimensional Data Warehousing », *IIIrd International Baltic Workshop on Databases and Information Systems, Riga, Latvia*, 1998, p. 27–41.

[BOD 03] BODY M., MIQUEL M., BÉDARD Y., TCHOUNIKINE A., « Handling Evolutions in Multidimensional Structures », *XIXth International Conference on Data Engineering (ICDE 03), Bangalore, India*, IEEE Computer Society, 2003, p. 581–591.

[EDE 01] EDER J., KONCILIA C., « Changes of Dimension Data in Temporal Data Warehouses », *IIIrd International Conference on Data Warehousing and Knowledge Discovery (DaWaK 01)*, vol. 2114 de *LNCS*, Springer, 2001, p. 284–293.

[FAV 07] FAVRE C., BENTAYEB F., BOUSSAÏD O., « Intégration des connaissances utilisateurs pour des analyses personnalisées dans les entrepôts de données évolutifs », *VIIèmes Journées Francophones Extraction et Gestion des Connaissances (EGC 07), Namur, Belgique*, vol. E-9 de *Revue des Nouvelles Technologies de l'Information*, Cépaduès Editions, 2007, p. 217–222.

[GOL 06] GOLFARELLI M., LECHTENBORGER J., RIZZI S., VOSSEN G., « Schema Versioning in Data Warehouses : Enabling Cross-Version Querying via Schema Augmentation », *Data and Knowledge Engineering*, vol. 59, n° 2, 2006, p. 435–459, Elsevier Science Publishers B. V.

[HUR 99] HURTADO C. A., MENDELZON A. O., VAISMAN A. A., « Updating OLAP Dimensions », *IInd ACM International Workshop on Data Warehousing and OLAP (DOLAP 99), Kansas City, Missouri, USA*, ACM Press, 1999, p. 60–66.

[MAL 04] MALINOWSKI E., ZIMÁNYI E., « OLAP Hierarchies : A Conceptual Perspective », *XVIth International Conference on Advanced Information Systems Engineering (CAiSE 04), Riga, Latvia*, vol. 3084 de *LNCS*, Springer, 2004, p. 477-491.

[MAZ 06] MAZÓN J.-N., TRUJILLO J., « Enriching Data Warehouse Dimension Hierarchies by Using Semantic Relations », *XXIIIrd British National Conference on Databases (BNCOD 2006), Belfast, Northern Ireland*, vol. 4042 de *LNCS*, Springer, 2006, p. 278–281.

[MEN 00] MENDELZON A. O., VAISMAN A. A., « Temporal Queries in OLAP », *XXVIth International Conference on Very Large Data Bases (VLDB 00), Cairo, Egypt*, Morgan Kaufmann, 2000, p. 242–253.

[MOR 04] MORZY T., WREMBEL R., « On Querying Versions of Multiversion Data Warehouse », *VIIth ACM International Workshop on Data Warehousing and OLAP (DOLAP 04), Washington, Columbia, USA*, ACM Press, 2004, p. 92–101.

[NAB 05] NABLI A., SOUSSI A., FEKI J., BEN-ABDALLAH H., GARGOURI F., « Towards an Automatic Data Mart Design », *VIIth International Conference on Enterprise Information Systems (ICEIS 05), Miami, Florida, USA*, 2005, p. 226-231.

[RAV 06] RAVAT F., TESTE O., ZURFLUH G., « A Multiversion-Based Multidimensional Model », *VIIIth International Conference on Data Warehousing and Knowledge Discovery (DaWaK06), Krakow, Poland*, vol. 4081 de *LNCS*, Springer, 2006, p. 65-74.