Laboratoire ERIC
Université Lumière Lyon 2
Rapport de recherche interne

Une approche orientée utilisateur pour l'évolution de schéma dans les entrepôts de données

Laboratoire ERIC
Université Lyon2
5 Avenue Pierre Mendès-France
69676 BRON Cedex - France

http ://eric.univ-lyon2.fr

Cécile FAVRE
cecile.favre@univ-lyon2.fr

28 Septembre 2006

Résumé

Les entrepôts de données centralisent des données agrégées provenant de différentes sources pour répondre aux besoins d'analyse des utilisateurs. Les utilisateurs sont souvent impliqués dans la définition initiale du schéma, mais ils ne le sont pas dans le processus d'évolution de celui-ci. Dans cet article, nous proposons alors une approche qui permet aux utilisateurs d'intégrer leur propre connaissance du domaine afin d'enrichir le schéma de l'entrepôt, en leur laissant exprimer la façon dont ils souhaitent agréger les données. Nous représentons cette connaissance sous la forme de règles de type "si-alors". Ces règles sont donc exploitées pour créer de nouveaux axes d'analyse sous forme de niveaux de granularité qui enrichissent les hiérarchies de dimensions. Pour valider le cadre formel que nous proposons, nous avons développé un prototype et avons appliqué notre approche sur les données bancaires de LCL-Le Crédit Lyonnais.

Abstract

Data warehouses store aggregated data from different heterogeneous sources to answer users' analysis needs for decision support. The analysis possibilities of a data warehouse are determined by its schema. Thus changes which occur on data sources or on analysis needs necessitate a schema evolution of the data warehouse. In the literature, different works refer to schema evolution to tackle the problem of data sources changes. In this paper, to take into account analysis needs evolution, we propose an approach to involve users in the schema evolution process by allowing them to express the way to aggregate data. We choose to express that with "if-then" rules. Thus, these rules are used to enrich and personalize analysis possibilities by creating new granularity levels in dimension hierarchies of the data warehouse. To validate the formal framework we propose, we developed a prototype and applied our approach on banking data of LCL-Le Crédit Lyonnais.

1 Introduction

Pour analyser une masse de données de plus en plus conséquente, provenant de sources hétérogènes, la structuration et le stockage de ces données dans un entrepôt constituent un support efficace. Une des étapes clé de la réussite du processus d'entreposage réside dans la définition du schéma multidimensionnel de cet entrepôt. En effet, c'est ce schéma qui détermine les analyses pouvant être effectuées à partir de l'entrepôt : ce dernier va permettre d'analyser des *mesures* qui caractérisent des *faits*, en fonction de *dimensions* qui peuvent être organisées selon des *hiérarchies*, déterminant la manière selon laquelle sont agrégées les données.

Les modèles multidimensionnels classiques tels que celui proposé par [CT98] considèrent les faits comme la partie dynamique des entrepôts de données et les dimensions comme des entités statiques. Cependant, en pratique, des changements peuvent se produire dans les dimensions, soit parce que les sources de données elles-mêmes sont modifiées, soit parce que les besoins d'analyse des utilisateurs évoluent. Ces changements doivent être répercutés sur le schéma de l'entrepôt qui doit par conséquent évoluer.

Différents travaux se sont intéressés à l'évolution de schéma des entrepôts en proposant des stratégies soit de mise à jour, assurant le transfert des données d'un ancien schéma vers un nouveau comme suggéré par [HMV99], soit de modélisation temporelle pour garder la trace des évolutions en ayant recours à des labels temporels, comme proposé par [BSSJ98], [MV00], ou encore [MW04]. Ces approches constituent des alternatives intéressantes pour prendre en compte l'évolution de schéma induite par une modification dans les sources de données, en proposant des structures qui permettent de prendre en compte ces changements. Cependant, elles n'apportent pas de solution à l'émergence de nouveaux besoins d'analyse exprimés par les utilisateurs. En effet, ces approches n'impliquent pas directement les utilisateurs dans le processus d'évolution, alors qu'ils le sont souvent dans la phase de définition du schéma de l'entrepôt [Kim96].

Notre objectif consiste alors à impliquer les utilisateurs dans le processus d'évolution de l'entrepôt, en permettant des analyses personnalisées en fonction de leur propre connaissance du domaine et de leurs besoins. Il s'agit, plus précisément de générer de nouveaux axes d'analyse en enrichissant les hiérarchies de dimension avec de nouveaux niveaux de granularité basés sur cette connaissance. Pour atteindre cet objectif, nous proposons dans cet article une approche globale composée de quatre étapes : (1) une phase d'acquisition des connaissances de l'utilisateur, (2) une phase d'intégration de ces connaissances ; (3) une phase de mise à jour de la hiérarchie de dimension en fonction des connaissances intégrées, et (4) une phase d'analyse prenant en compte le nouveau schéma. Dans notre approche, nous définissons la connaissance des utilisateurs sous la forme de règles d'agrégation de type "si-alors". Ce type de règles permet de modéliser les connaissances de façon simple et explicite, telle que l'a précisé [HHNT86], et se prête bien à la connaissance que l'on veut représenter, en l'occurrence la façon d'agréger les données. Pour mettre en œuvre notre approche, nous proposons un modèle dynamique d'entrepôt de données basé sur ces règles d'agrégation, baptisé *R-DW* (*Rule-based Data Warehouse*). Notre modèle *R-DW* est composé d'une partie "fixe", correspondant à un schéma qui répond à des besoins d'analyse globaux ; et d'une partie "évolutive" définie par les règles d'agrégation

répondant aux besoins d'analyse individuels.

Notre principale contribution est alors d'impliquer l'utilisateur dans l'évolution de schéma de l'entrepôt de données pour répondre à des besoins d'analyse individuels, et d'augmenter ainsi l'intéraction entre celui-ci et le système décisionnel. En effet, en offrant la possibilité à l'utilisateur d'intégrer sa propre connaissance dans l'entrepôt pour créer dynamiquement des niveaux de granularité, celui-ci devient un réel acteur du processus décisionnel, dont le rôle va au-delà de la navigation dans les analyses prévues par le modèle. De plus, nous définissons un cadre formel qui permet de décrire notre modèle. Pour valider notre modèle, nous avons développé un prototype baptisé WEDriK (data Warehouse Evolution Driven by Knowledge) et avons appliqué notre approche aux données bancaires de LCL[1] (Le Crédit Lyonnais).

Le reste de cet article est organisé de la façon suivante. Nous introduisons un exemple motivant l'utilisation du modèle *R-DW* dans la Section 2. Puis, nous discutons l'état de l'art dans la Section 3. Nous présentons ensuite, dans la Section 4, notre approche globale ainsi que le modèle *R-DW*, les règles qui le constituent et les contraintes sur ces règles. Puis nous définissons le cadre formel du modèle *R-DW* dans la Section 5, avant d'exposer la mise en œuvre de ce modèle dans la Section 6. Enfin, nous concluons et indiquons les perspectives de ce travail dans la Section 7.

2 Exemple introductif

Pour illustrer notre approche de modélisation d'entrepôts de données dynamiques et évolutifs à base de règles, nous utilisons, tout au long de cet article, le cas réel de la banque LCL. [2] (Le Crédit Lyonnais). Le PNB annuel (Produit Net Bancaire) correspond à ce que rapporte un client à l'établissement bancaire. Cette mesure est analysée selon les dimensions `CLIENT`, `AGENCE` et `ANNEE`. Le modèle multidimensionnel présenté dans la Figure 1 répond à ce besoin d'analyse.

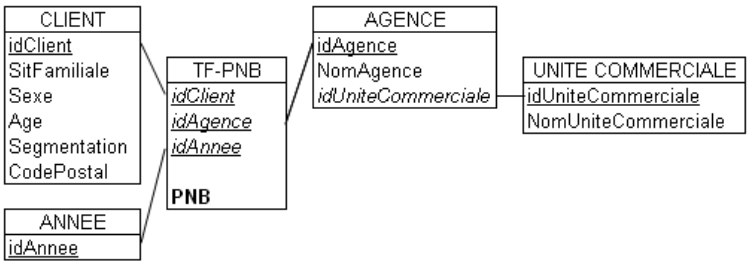

FIG. 1 – Modèle d'entrepôt pour l'analyse du PNB du LCL

La Figure 2(a) montre le schéma de la dimension `AGENCE`, qui représente la façon dont peuvent être agrégées les données. Initialement, il est donc possible d'agréger les données selon le niveau `UNITE_COMMERCIALE`, qui correspond à

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

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

un regroupement d'agences. Supposons qu'un utilisateur veuille analyser les données selon le type d'agence. Si cette information n'est pas présente dans l'entrepôt de données, il est impossible pour l'utilisateur d'obtenir une telle analyse.

Nous proposons alors à l'utilisateur d'intégrer sa propre connaissance sur les types d'agence pour générer le niveau de granularité `TYPE_AGENCE`, à travers la définition de l'attribut `NomTypeAgence` caractérisant ce niveau, enrichissant ainsi le schéma de la dimension selon la Figure 2(b). Pour définir le niveau `TYPE_AGENCE`, et par conséquent les valeurs de l'attribut `NomTypeAgence`, ainsi que le lien d'agrégation par rapport au niveau `AGENCE`, les règles suivantes sont utilisées :
$W = \{$'01903','01905','02256'$\}$
$(R1)$ *si* $idAgence \in W$ *alors* $NomTypeAgence =$ 'étudiant'
$(R2)$ *si* $idAgence \notin W$ *alors* $NomTypeAgence =$ 'classique'

De cette façon, 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 ($R1$) ou au contraire d'une agence classique ($R2$).

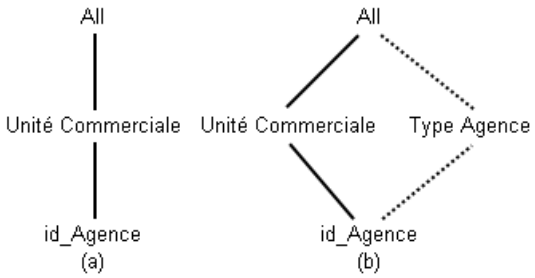

FIG. 2 – (a) Schéma de la dimension `AGENCE` initial, (b) Schéma de la dimension `AGENCE` enrichi par l'utilisateur

3 État de l'art

L'entreposage de données est une technologie dite "centrée utilisateur". Pour parvenir à un tel qualificatif, nous pensons que l'implication de l'utilisateur est nécessaire, et pas seulement du point de vue de l'utilisation "finale" de l'entrepôt. Or, cette implication est très variable selon les étapes du processus d'entreposage, et selon les approches qui ont été proposées. L'implication de l'utilisateur au niveau de la conception du schéma de l'entrepôt a déjà été évoquée, mais elle ne fait pas l'objet, à notre connaissance, de proposition au niveau de la prise en compte de l'évolution du schéma.

Du point de vue de la conception du schéma de l'entrepôt, nous distinguons dans la littérature trois types d'approche : 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 de schéma. En effet, il s'agit de construire

le schéma de l'entrepôt à partir de ceux des sources de données, en supposant que le modèle qui sera construit pourra répondre aux besoins d'analyse, ce qui n'est pas forcément le cas. Par exemple, [GMR98] 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. Les approches guidées par les besoins d'analyse, quant à elles, proposent de définir le schéma de l'entrepôt en partant des besoins décisionnels comme l'a proposé [Kim96]. Ainsi, les utilisateurs sont impliqués dans le processus de définition du schéma en étant consultés afin de collecter l'ensemble de leurs 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 encore moins possible de déterminer leurs besoins à venir. Les approches mixtes permettent, quant à elles, de combiner les deux types d'approches précédentes. 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. De telles approches ont été proposées par [BCC+01], [PD02] ou encore [NSF+05].

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 migrer les données d'un ancien schéma vers le plus récent comme l'ont proposé [BSH99] ou [HMV99]. Dans ce cas, un seul schéma est supporté. L'avantage de cette approche est qu'elle fournit une comparaison des données dans le temps. Cependant, travailler avec la version la plus récente du schéma masque l'évolution des dimensions. Une analyse peut alors fournir des conclusions erronées. La deuxième alternative consiste, elle, à garder la trace de ces évolutions, en utilisant des labels de validité temporelle. Ces labels sont apposés soit au niveau des instances tel que l'ont proposé [BSSJ98], soit au niveau des liens d'agrégation comme proposé par [MV00], ou encore au niveau des versions du schéma comme l'ont proposé [BMBT02, BEK+04] ou encore [MW04]. Les évolutions du schéma sont donc bien conservées et assure la cohérence des analyses. Toutefois, l'inconvénient de cette approche est de ne pas disposer de comparaison des données dans le temps. Par ailleurs, ce type de solutions nécessite une réimplémentation des outils d'analyse, de chargement de données,... afin de gérer les particularités de ces modèles.

Un troisième type de travail sur l'évolution de schéma dans les entrepôts de données a émergé, se focalisant particulièrement sur l'aspect 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, ce courant permet de prendre en compte l'évolution des sources de données en propageant ces modifications sur les vues de l'entrepôt, tel que le propose [Bel02].

Ces trois courants constituent des alternatives intéressantes pour répondre

au problème de l'évolution de schéma induite par une modification dans les sources de données. Cependant, elles n'apportent pas de solution à l'émergence de nouveaux besoins d'analyse exprimés par les utilisateurs. En effet, ces approches n'impliquent pas directement les utilisateurs dans le processus d'évolution. Elles constituent des solutions techniques devant être mises en œuvre par l'administrateur pour faire évoluer l'entrepôt de données, mais n'envisageant pas l'évolution des besoins d'analyses exprimée par les utilisateurs eux-mêmes.

Dans l'approche que nous proposons dans ce papier, nous nous focalisons sur l'évolution de schéma induite par la prise en compte de connaissances utilisateurs. Notre approche s'inspire des approches de conception présentées précédemment. En effet, nous voulons assurer à la fois la validité du modèle du point de vue des sources de données disponibles, et la conformité de celui-ci avec les besoins d'analyse des utilisateurs. C'est pourquoi, nous préconisons un modèle avec une partie fixe dont la définition est guidée par des besoins d'analyse globaux recueillis lors de la conception de l'entrepôt et les données qui permettent d'y répondre. Notre approche s'inspire également des travaux d'évolution de schéma, en particulier ceux proposant une mise à jour des dimensions. C'est pourquoi, la partie fixe de notre modèle est reliée à une partie évolutive qui traduit les besoins utilisateurs, leur permettant ainsi d'accéder à des analyses personnalisées. Grâce au modèle R-DW (Rule-based Data Warehouse) que nous présentons plus en détail dans la section suivante, nous parvenons à une architecture répondant au qualificatif de "centrée utilisateur".

4 Une approche orientée utilisateur pour mettre à jour les hiérarchies de dimension

4.1 Modélisation à base de règles

L'architecture globale de notre approche est présentée dans la Figure 3. Elle consiste donc à ajouter au schéma de base de l'entrepôt de données un ensemble de règles d'agrégation qui représentent la façon selon laquelle les utilisateurs veulent agréger les données, nécessitant la phase d'acquisition de ces connaissances, puis d'intégration afin d'être stockées. Ces règles permettent la mise à jour du schéma de base via la création ou l'enrichissement de hiérarchies de dimension lors de la phase de mise à jour. Le modèle induit par les règles évolue ainsi de façon incrémentale et dynamique, selon les besoins exprimés par les utilisateurs, rendant possible la phase d'analyse sur le modèle enrichi.

Cette approche se base sur le modèle *R-DW*, qui est composé de deux parties : une partie fixe et une partie évolutive (Figure 4). La partie fixe est composée d'une table de faits reliée à des tables de dimensions qui peuvent être hiérarchisées (schéma en étoile ou en flocon de neige). La partie évolutive est définie par un ensemble de règles d'agrégation, qui définissent le lien d'agrégat entre deux niveaux de granularité dans une hiérarchie de dimension.

Pour gérer l'évolution du schéma de l'entrepôt selon notre approche, nous proposons le méta-modèle présenté dans la Figure 4. Il considère différents concepts classiques du domaine tels que les notions de dimension, fait, mesure, niveau... La particularité de notre méta-modèle est qu'un niveau ne correspond pas seulement à un ensemble d'attributs explicites, mais aussi à des attributs générés avec des règles.

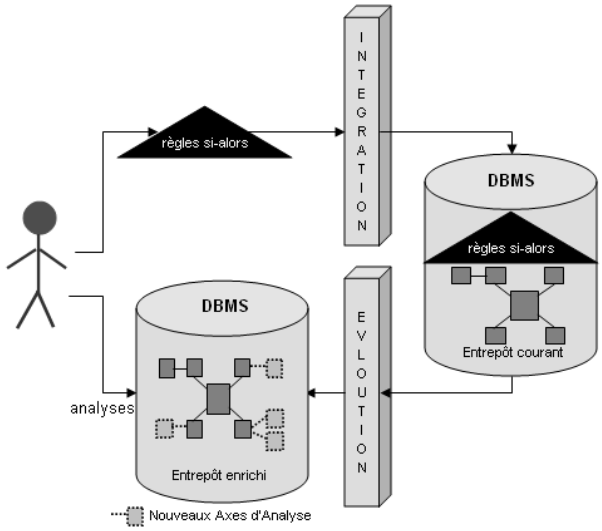

FIG. 3 – Architecture globale pour l'évolution de schéma

Ces règles sont des règles de type "*si-alors*". La clause "*si*" permet d'exprimer les conditions sur les attributs caractérisant le niveau de granularité dit "inférieur", c'est à dire le niveau existant sur lequel va se baser le nouveau niveau. 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 supérieur, 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. Ce mécanisme est représenté dans la Figure 5. Ainsi, les données concernant chaque instance du niveau inférieur pourront être agrégées selon une instance du niveau créé. En effet, nous avons choisi de représenter les hiérarchies classiques, dans lesquelles une instance d'un niveau donné correspond à une instance du niveau supérieur, même s'il existe différents types d'agrégation pour modéliser des situation réelles telles qu'on pu les décrire [PJD01] ou [MZ04].

Par exemple, les règles suivantes déterminent le niveau hiérarchique `TYPE_AGENCE` en définissant les valeurs de l'attribut `NomTypeAgence`, en se basant sur le niveau hiérarchique `AGENCE` (attribut `idAgence`) :
$W = \{$'01903','01905','02256'$\}$
$(R1)$ *si* $idAgence \in W$
alors $NomTypeAgence =$ 'étudiant'
$(R2)$ *si* $idAgence \notin W$
alors $NomTypeAgence =$ 'classique'

Pour générer la partition des instances, les conditions exprimées peuvent porter indifféremment sur des attributs discrets ou continus. Dans le premier cas, cela nécessite de lister les valeurs. Dans le second cas, il est alors possible d'avoir recours à des opérateurs de comparaison. Il est également possible de considérer des compositions de conditions portant sur plusieurs attributs. Le tableau de la Figure 6 récapitule ces traitements et les exemples les illustrant.

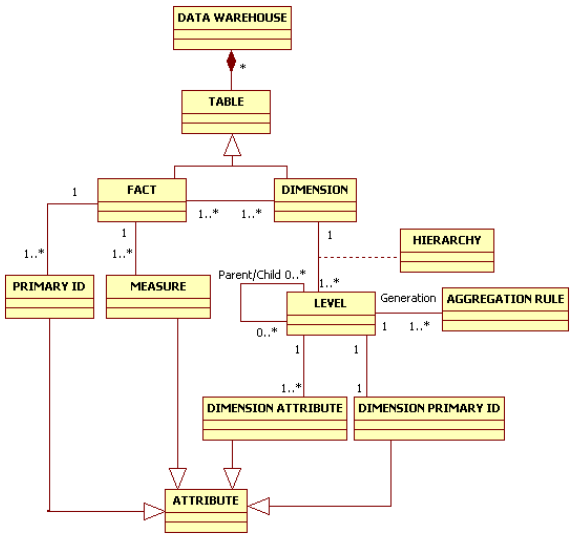

FIG. 4 – *R-DW* meta model

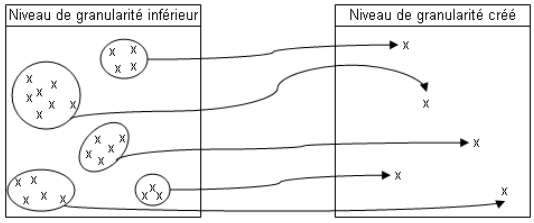

FIG. 5 – Partition induite par les règles d'agrégation

4.2 Ajout ou insertion de niveaux

Le niveau de granularité créé peut être ajouté à la fin de la 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 avant d'établir ce lien, il faut savoir si ce lien peut être effectivement généré. En effet, il peut être généré 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 **??**(a). Suppososns que nous ajoutons un nouveau niveau qui agrège les données des agences selon leur taille : petite, moyenne et grande. Il est impossible de créer un lien d'agrégation sémantique entre la taille d'agence et le niveau existant qui représente l'unité commerciale. Dans ce cas le niveau créé doit être ajouté à la fin d'une hiérarchie, et ne peut être inséré entre les niveaux Agence et Unité Commerciale.

Gestion	Moyen	Application	Exemples
Attributs discrets	Ensemble de valeurs	regroupement des identifiants des agences étudiantes dans un ensemble de valeurs et définition des règles en se basant sur cet ensemble	I={idAgence} : tous les identifiants d'agence
			E={'01903','01905','02256'} : identifiants des agences étudiantes
			(R1) si idAgence∈E alors attr_type_agence='étudiant'
			(R2) si idAgence ∈I-E alors attr_type_agence='classique'
Attributs continus	Discrétisation	définition de classes d'âge de clients	(R3) si Age<60 alors attr_classe_age='moins de 60 ans'
			(R4) si Age≥60 alors attr_classe_age='plus de 60 ans'
Combinaison d'attributs	Combinaison de conditions	définition de l'appartenance du client aux groupes 'femmes mariées', 'hommes mariés'..., constitués à partir de la combinaison des attributs SitFamiliale et Sexe	(R5) si Sexe='F' et SitFamiliale='M' alors attr_groupe_personne='femmes mariées'
			(R6) si Sexe='F' et SitFamiliale≠'M' alors attr_groupe_personne='femmes non mariées'
			(R7) si Sexe='H' et SitFamiliale='M' alors attr_groupe_personne='hommes mariés'
			(R8) si Sexe='H' et SitFamiliale≠'M' alors attr_groupe_personne='hommes non mariés'

FIG. 6 – Gestion des attributs dans les règles

4.3 Contraintes sur les règles

Les règles permettent donc 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 ne doit pas être apportée au modèle au détriment de sa cohérence. En effet, c'est la cohérence du modèle qui assure celle des analyses. Pour valider les règles exprimées par les utilisateurs, nous définissons alors trois types de contraintes :

- contrainte d'intégrité
- contrainte de cohérence
- 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.

Prenons l'exemple de la définition du niveau de granularité `REGION` dans la hiérarchie de dimension concernant la localisation géographique des clients selon la règle suivante :

$si\ attr\_departement\ \in\ \{'01','07','26','38','42','69','73','74'\}$
$alors\ attr\_region = 'Rhône - Alpes'$

Cette règle, qui définit le niveau `REGION`, satisfait la contrainte d'intégrité si :

- l'attribut `attr_departement` est présent dans la partie fixe ou dans la partie évolutive (i.e. défini lui-même grâce à des règles)
- les valeurs de l'ensemble $\{'01','07','26','38','42','69','73','74'\}$ appartiennent bien au domaine de définition de `attr_departement`
- $'Rhône - Alpes'$ est une valeur possible du domaine de définition de l'attribut `attr_region`

La contrainte de cohérence est directement liée au fait que l'ensemble de règles qui définissent un niveau donné doivent former une partition du domaine de définition de l'attribut qui représente ce niveau de granularité. Cela implique non seulement que tout le domaine de définition doit être couvert par les règles, mais aussi qu'il ne doit pas y avoir de recouvrement entre les conditions expri-

mées dans les règles. En d'autres termes, deux prémisses de règles basées sur des attributs similaires, mais avec des conditions différentes sur ceux-ci, ne peuvent pas donner une conclusion identique.

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`. Un même utilisateur ne peut pas définir les règles suivantes :

$si\ Age < 60\ alors\ attr\_classe\_age$ = 'moins de 60 ans'

$si\ Age < 80\ alors\ attr\_classe\_age$ = 'moins de 80 ans'

Dans ce cas, il y a un recouvrement des règles qui rend impossible le calcul de la valeur de l'attribut `attr_classe_age` pour une personne âgée de 40 ans par exemple.

La contrainte de complétude a pour but d'assurer que lorsqu'un niveau de granularité est construit, chaque instance 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 la règle définissant ce qu'est une agence étudiante. En effet, dans ce cas, différents indicatifs d'agence (tous ceux correspondant à des agences classiques) ne seraient pas repris au niveau `TYPE_AGENCE`.

4.4 Discussion

Notre modèle *R-DW* présente plusieurs avantages par rapport aux modèles d'entrepôt existants. Il permet de :

- faire évoluer les contextes d'analyse en permettant de créer ou d'enrichir des hiérarchies de dimensions de façon dynamique ;
- prendre en compte des connaissances du domaine exprimées par l'utilisateur ;
- renforcer l'interaction entre l'utilisateur et le système d'aide à la décision en permettant à celui-ci d'intégrer ses propres connaissances.

Notre approche a pour but d'enrichir les possibilités d'analyse d'un entrepôt de données. Notre approche étant centrée utilisateur, il est primordial que l'implication des utilisateurs ne compromette pas les données stockées initialement dans l'entrepôt. Or, ajouter un niveau de granularité au dessus de la table des faits impliquerait un recalcul de la table des faits. Ainsi, pour permettre une approche centrée utilisateur d'évolution de schéma de l'entrepôt de données, nous supposons dans ce papier que les nouveaux besoins d'analyse ne modifient ni la table des faits, ni les niveaux de granularité directement liés à celle-ci. Pour ces raisons, nous avons choisi de baser notre modèle sur une partie fixe qui correspond à un schéma classique et qui permet de répondre à des besoins d'analyse globaux.

Par ailleurs, notre proposition consiste à impliquer l'utilisateur dans la création d'un nouveau niveau de granularité, et ainsi de préciser comment les données doivent être agrégées d'un niveau à un autre. Comme les utilisateurs doivent exprimer cela au moyen de règles, cette approche est plus intéressante dans certains cas. D'une manière générale, on peut dire qu'elle est plus intéressante lorsque le nombre de tuples dans le niveau créé n'est pas trop important. En effet, ce nombre correspond au nombre de règles nécessaire pour construire le niveau. Ainsi la faisabilité de notre approche dépend particulièrement de la facilité d'identifier les instances dans le niveau inférieur sur lequel va se baser le

niveau créé, et ainsi de la simplicité des règles exprimées.

La difficulté d'exprimer les règles dépend surtout du type des attributs utilisés dans les règles. En effet, l'objectif des règles "si-alors" est de regrouper des instances du niveau inférieur. Si des attributs continus sont utilisés, il sera plus simple de définir par exemple des intervalles de valeurs en exprimant des conditions (tel que sur l'attribut âge par exemple). Lorsque des attributs discrets sont utilisés, l'utilisateur doit lister chaque valeur de l'attribut de façon explicite. Si la cardinalité de l'attribut concerné est importante, la tâche peut devenir réellement fastidieuse (dans le cas d'un regoupement par rapport à l'identifiant de client par exemple). Ainsi, nous pensons qu'il est important de fournir à l'utilisateur une interface qui permette d'exprimer des règles de façon simple.

5 Cadre formel

Nous présentons dans cette section le cadre formel de notre approche.

5.1 Le modèle *R-DW*

Nous désignons le modèle d'entrepôt à base de règles *R-DW* par le triplet suivant :

$$R\text{-}DW = (\mathcal{F}, \mathcal{E}, \mathcal{U})$$

où $\mathcal{F}$ est la partie fixe, $\mathcal{E}$ la partie évolutive et $\mathcal{U}$ l'univers de *R-DW*.

Définition 1. *Univers de l'entrepôt*
Soit $R\text{-}DW = (\mathcal{F}, \mathcal{E}, \mathcal{U})$.
L'*univers* $\mathcal{U}$ *de l'entrepôt R-DW* est un ensemble d'attributs, tel que :

$$\mathcal{U} = \mathcal{U}_1 \cup \mathcal{U}_2$$

où $\mathcal{U}_1 = \{B_\alpha, 1 \leq \alpha \leq z\}$ est l'ensemble des z attributs prédéfinis (définis dans la partie fixe $\mathcal{F}$ de *R-DW*) et $\mathcal{U}_2 = \{C_\beta, \beta \geq 1\}$ est l'ensemble des attributs générés (définis par la partie évolutive $\mathcal{E}$ de *R-DW*).

Définition 2. *Partie fixe de R-DW*
Soit $R\text{-}DW = (\mathcal{F}, \mathcal{E}, \mathcal{U})$.
La *partie fixe de R-DW* est définie par :

$$\mathcal{F} = < F, \mathcal{D} >$$

où F est une table de faits, et $\mathcal{D} = \{D_s, 1 \leq s \leq t\}$ est l'ensemble des t dimensions de premier niveau qui ont un lien direct avec F. Nous supposons que ces dimensions sont indépendantes.

Une table de dimension D_s contient une clé primaire $D_s.PK$ et un ensemble de n_s attributs $\{D_s.Z_n, 1 \leq n \leq n_s\}$.

F contient une clé composée d'un ensemble de t ($t \geq 1$) clés étrangères $\{F.K_s, 1 \leq s \leq t\}$ où $F.K_s = D_s.PK$, et un ensemble de m mesures notées $\{F.M_q, 1 \leq q \leq m\}$.

Notre objectif est d'ajouter de nouveaux axes d'analyse à la partie fixe. Ces nouveaux axes sont exprimés grâce à la définition de nouveaux niveaux de granularité dans les hiérarchies de dimension.

Définition 3 *Hiérarchie de dimension et niveau de granularité*
Soit *R-DW* $= (< F, \mathcal{D} >, \mathcal{E}, \mathcal{U})$.
$D_s.H_k, D_s \in \mathcal{D}, k \geq 1$ est une *hiérarchie de la dimension* D_s.
La hiérarchie de dimension $D_s.H_k$ est composée d'un ensemble de w niveaux de granularité ordonnés notés L_i :
$D_s.H_k = \{L_1, L_2, \ldots, L_i, \ldots, L_w, w \geq 1\}$, avec $L_1 \prec L_2 \prec \ldots \prec L_i \prec \ldots \prec L_w$ où $\prec$ exprime l'ordre partiel sur les L_i.
Le *niveau de granularité* L_i de la hiérarchie H_k de la dimension D_s est noté $D_s.H_k.L_i$ ou plus simplement L_i^{sk}. Les niveaux de granularité sont définis par des attributs générés.

Définition 4 *Attribut généré*
Soit *R-DW* $= (< F, \mathcal{D} >, \mathcal{E}, \mathcal{U})$.
Soit L_i^{sk} le niveau de granularité L_i de la hiérarchie H_k de la dimension D_s.
Soit $\mathcal{U} = \mathcal{U}_1 \cup \mathcal{U}_2$ où $\mathcal{U}_2$ est l'ensemble des attributs définis par la partie évolutive $\mathcal{E}$ de *R-DW*.
Un *attribut généré* $A \in \mathcal{U}_2$ caractérise le niveau de granularité L_i^{sk} et est noté $L_i^{sk}.A$.

Remarque : pour des raisons de simplification, nous supposons que chaque niveau d'une hiérarchie de dimension ne contient qu'un seul attribut généré, même s'il est possible d'en générer plusieurs par niveau de granularité.

Définition 5. *Partie évolutive de R-DW*
Soit *R-DW* $= (\mathcal{F}, \mathcal{E}, \mathcal{U})$.
La *partie évolutive de R-DW* est définie par :

$$\mathcal{E} = \{< \mathcal{R}_{i.}, L_i^{sk}.A >\}$$

où $\mathcal{R}_{i.} = \{r_{ij}, 1 \leq j \leq v, 1 \leq i \leq w\}$ est un ensemble de v règles d'agrégation définissant les valeurs de l'attribut généré $L_i^{sk}.A$ qui représente le niveau de granularité L_i de la hiérarchie H_k de la dimension D_s. $\mathcal{E}$ représente donc l'ensemble des niveaux de granularité créés et les règles associées qui permettent de les définir.

Définition 6. *Règle d'agrégation*

Une *règle d'agrégation* permet de définir le lien d'agrégation qui existe entre deux niveaux de granularité dans une hiérarchie de dimension.

Elle est basée sur un ensemble $\mathcal{T}$ de n termes de règles, notés RT_p, tel que :

$$\mathcal{T} = \{RT_p, 1 \leq p \leq n\} = \{U\ op\ \{ens|val\}\}$$

où U est un attribut appartenant à l'univers $\mathcal{U}$ de l'entrepôt ; op est un opérateur soit relationnel ($=$, $<$, $>$, $\leq$, $\geq$, $\neq$, ...), soit ensembliste ($\in$, $\notin$, ...) ; ens est un ensemble de valeurs et val est une valeur finie.

Exemple 6a.
RT_1 : $idAgence \in$ {'01903','01905','02256'}
RT_2 : $idAnnee < 2001$
RT_3 : $Sexe =$ 'F'

Une règle d'agrégation est une règle de type "*si-alors*". La conclusion de la règle (clause "alors") définit la valeur de l'attribut généré. La prémisse de la règle (clause "si") est basée sur une composition de *(conjonctions|disjonctions)* des termes de règles :

$$r_{ij}\ :$$
$$si\ RT_1\ (et|ou)\ RT_2\ ...\ (et|ou)\ RT_n$$
$$alors\ L_i^{sk}.A = val_{ij}$$

où $val_{ij} \in Dom_{L_i^{sk}.A}$ le domaine de définition de l'attribut $L_i^{sk}.A$.

Nous notons $body(r_{ij})$ la clause "si" de la règle et $head(r_{ij})$ la clause "alors".

Exemple 6b. Les règles suivantes déterminent les différentes valeurs de l'attribut `NomTypeAgence` qui définit le niveau hiérarchique `TYPE_AGENCE` :
r_{11} : $si\ idAgence \in$ {'01903','01905','02256'}
$alors\ NomTypeAgence =$ 'étudiant'
r_{12} : $si\ idAgence \notin$ {'01903','01905','02256'}
$alors\ NomTypeAgence =$ 'classique'

Exemple 6c. La règle suivante permet de définir la valeur *'femmes mariées'* de l'attribut `attr_groupe_personnes` du niveau hiérarchique `GROUPE_PERSONNES` à partir des attributs `Sit_Familiale` et `Sexe` de la table `CLIENT` :
$si\ Sit\_Familiale =$ 'Marié' $et\ Sexe =$ 'F'
$alors\ attr\_groupe\_personne =$ 'femmes mariées'

Ainsi, les règles d'agrégation créent ou enrichissent une hiérarchie de dimension en définissant les valeurs de l'attribut en fonction d'une condition (Exemple 6b.) ou de composition de conditions (Exemple 6c.) basées sur les attributs des niveaux inférieurs de la dimension.

5.2 Les contraintes

Les règles doivent définir une partition du domaine de définition de l'attribut généré.

D'une part, les conditions exprimées dans un ensemble de règles qui définissent un attribut généré donné prises deux à deux doivent être disjointes :

$$\forall i, \forall j, q \; tels \; que \; j < q, j \in [1, v-1], q \in [2, v],$$
$$body(r_{ij}) \bigcap body(r_{iq}) =$$

D'autre part, l'ensemble du domaine de définition de l'attribut généré noté $Dom_{L_i^{s^k}.A}$ doit être couvert par les règles :

$$\forall i, \bigcup_{j=1}^{v} val_{ij} = Dom_{L_i^{s^k}.A}$$

Dans la section suivante, nous reviendrons sur comment valider ces contraintes.

6 Mise en œuvre

Pour valider notre approche, nous avons développé une plateforme nommée WEDriK (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 tout d'abord les fonctionnalités et l'implémentation des contraintes, avant de présenter un cas d'application.

6.1 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 7), selon quatre étapes :

1) Ils choisissent le niveau sur lequel sera basé le nouveau niveau ;

2) Ils expriment les conditions qui permettent le regroupement des instances du niveau de base ;

3) Ils définissent les modalités des attributs qui composent le nouveau niveau ;

4) Ils répètent les étapes 2) et 3) jusqu'à ce que toutes les instances du niveau de base aient une instance correspondante dans le nouveau niveau.

Deuxièmement, pour le stockage des connaissances, nous pouvons préciser que les règles sont stockées à l'aide d'une table relationnelle. Les différents attributs de la table sont : (1) niveau de base, (2) clause *si*, (3) clause *alors*. Ainsi, les règles sont stockées dans le SGBD. Cela évite d'avoir à développer un système complexe devant gérer simultanément les règles et l'entrepôts 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, concernant la mise à jour de la hiérarchie, notre plateforme permet la création d'un nouveau niveau dans une hiérarchie qui détermine les nouvelles possibilités d'analyse 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 (dans le cas où nous supposons

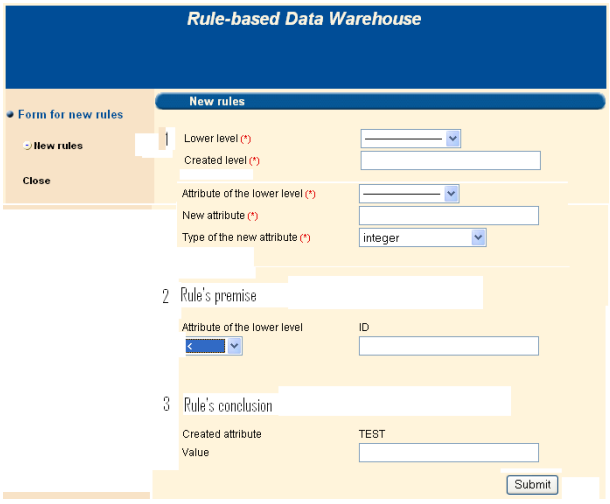

FIG. 7 – Formulaire pour de nouvelles règles

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 avons ensuite à mettre à jour dan 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.

Algorithme 1 Création 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**

Quatrièmement, pour l'exploitation du nouveau modèle, notre prototype fournit le schéma d'analyse. 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 lorsqu'un utilisateur interroge le prototype. 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 utilise donc le standard qu'est XML, évitant le recours à

un outil de visualisation sous un format propriétaire.

6.2 Mise en œuvre de l'insertion d'un niveau

Lorsqu'un niveau doit être inséré entre deux niveaux existants, l'utilisateur ne doit formuler que le lien d'agrégation 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, dans le cas bien sûr où cela est sémantiquement possible. Pour cela, nous proposons 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 $L2$, et de supprimer le lien entre $L1$ 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, nous modifions la structure de la table $L1$, en supprimant 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 $L1.A$=$L'.A$) ;

9: ALTER TABLE $L1$ DROP COLUMN B ;

6.3 Mise en œuvre des contraintes

Définir des contraintes pour tester la validité des règles exprimées par les utilisateurs est primordial et celles-ci doivent être implémentées. Tout d'abord, la contrainte d'intégrité est mise en œuvre 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, une validation sémantique des règles devrait être réalisée, ce qui constitue une tâche pas forcément réalisable. Nous avons alors choisi de tester la validité des règles en exploitant les données. Pour atteindre ce but, comme nous avons fait le choix de rester dans le SGBD, nous proposons de baser notre vérification sur des requêtes.

Considérons l'ensemble des règles R qui détermine un nouveau niveau de granularité L' à partir du niveau inférieur L. Cet ensemble de règles doit satisfaire deux contraintes pour respecter la définition d'une partition. Premièrement, les conditions exprimées dans la clause "si" des différentes règles prises deux à deux doivent être mutuellement disjointes. Deuxièmement, chaque instance du niveau inférieur doit avoir une instance correspondante dans le niveau créé.

Pour permettre cette vérification, nous proposons l'Algorithme 3. 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. Par exemple, considérons la règle qui définit une valeur de l'attribut *attr_groupe_personne* à partir des attributs *SitFamiliale* et *Sexe* du niveau $CLIENT$:

r : *si* $SitFamiliale =$ 'M' *et* $Sexe =$ 'F'
alors $attr\_groupe\_personne =$ 'femmes mariées'
q : SELECT * FROM CLIENT WHERE $SitFamiliale =$ 'M' $AND\ Sexe =$ 'F'

La requête nous fournit un ensemble d'instances du niveau L. Premièrement, nous vérifions que l'intersection de tous ces ensembles considérés pris deux à deux est vide. Deuxièmement, nous vérifions que l'union de ces ensembles correspond à l'ensemble des instances du niveau L.

Notons que ce choix de validation implique que les règles sont validées par les données disponibles dans l'entrepôt de données. De ce fait, cette validation doit être effectuée de nouveau si ces données sont mises à jour dans l'entrepôt.

6.4 Cas d'application

Nous avons appliqué notre modèle aux données bancaires qui concernent l'analyse du PNB. Les règles exprimées par les utilisateurs et le schéma induit par ces règles sont présentés dans la Figure 8. Ainsi, à partir de ce nouveau modèle, l'utilisateur pourra effectuer des analyses sur le PNB, non seulement en utilisant les dimensions de premier niveau, mais également en faisant intervenir des niveaux de granularité générés par les règles comme le type d'agence, la période, les classes d'âge...

règles, créer une vue regroupant les données

7 Conclusion

Dans cet article, nous avons proposé un modèle d'entrepôt de données dynamique et évolutif, à base de règles, nommé *R-DW*, dans lequel les règles expriment de nouvelles connaissances définies par l'utilisateur lui-même. Notre

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

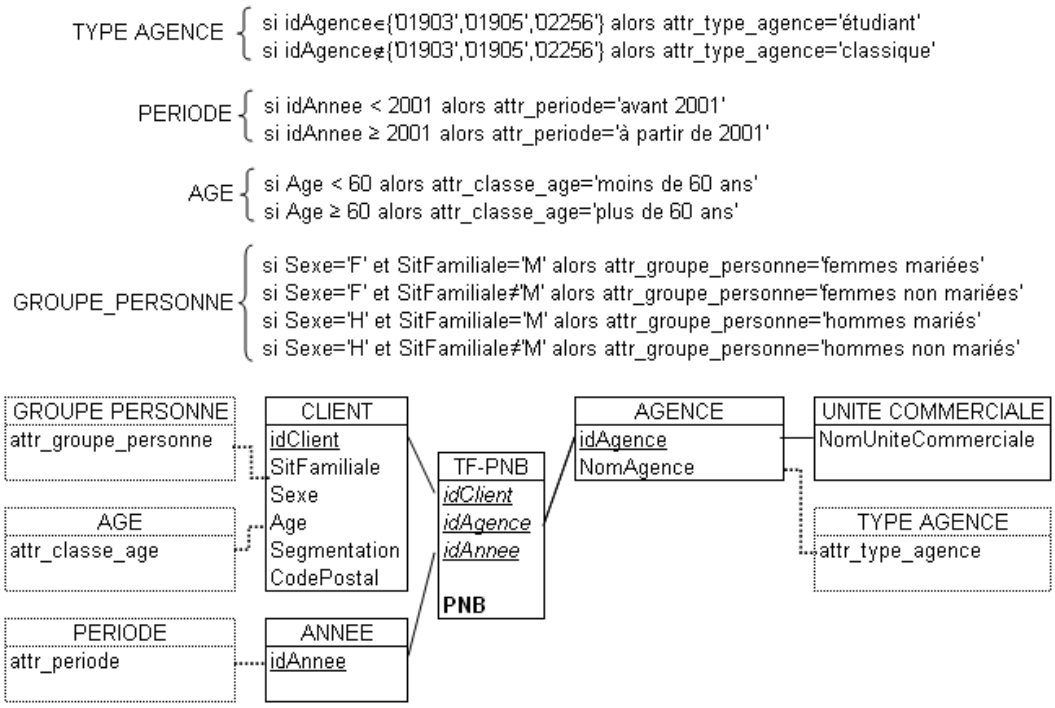

FIG. 8 – Règles et schéma induit pour l'analyse du PNB

modèle *R-DW* est composé de deux parties : une partie fixe correspondant à un schéma en étoile ou en flocon de neige ; une partie évolutive, définie par les règles, qui génèrent ou enrichissent les hiérarchies de dimensions. Notre modèle *R-DW* permet d'impliquer l'utilisateur dans l'évolution du schéma de l'entrepôt pour répondre aux besoins d'analyse émergents qui dépendent de sa propre connaissance du domaine. La plateforme que nous avons réalisée pour valider notre formalisation a permis d'appliquer notre modèle aux données de la banque LCL en prenant en compte des connaissances exprimées par un utilisateur. Afin d'assurer la cohérence du modèle induit par les règles, nous avons introduit différents types de contraintes (intégrité, cohérence et complétude).

Ce travail ouvre différentes perspectives. Tout d'abord, il nous semble primordial de pouvoir prendre en compte l'évolution des connaissances elles-mêmes, en réfléchissant aux stratégies de mise à jour et de versionnement au niveau des règles elles-mêmes. Par ailleurs, nous nous sommes placés dans le cas des hiéarchies classiques. Il serait donc intéressant de pouvoir traduire d'autres types de liens d'agrégation, tels que ceux évoqués par [PJD01], en adaptant les contraintes posées sur les règles. D'autre part, les règles d'agrégation exprimées se basent sur les instances dans le modèle. Pour faciliter la tâche d'acquisition des connaissances, il nous semble alors pertinent 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 soit instanciée. Enfin, concernant la définition des règles, l'atout de notre approche est de pouvoir faire intervenir l'utilisateur en lui laissant la possibilité d'introduire ses connaissances dans le système.

Mais il nous semble intéressant, en parallèle, de pouvoir l'aider à découvrir de nouveaux axes d'analyse. Pour ce faire, nous pensons que des méthodes d'apprentissage non supervisé peuvent être utilisées pour permettre la génération de la partition des instances du niveau inférieur.

Références

[BCC+01] Angela Bonifati, Fabiano Cattaneo, Stefano Ceri, Alfonso Fuggetta, and Stefano Paraboschi. Designing data marts for data warehouses. *ACM Trans. Softw. Eng. Methodol.*, 10(4) :452–483, 2001.

[BEK+04] Bartosz Bébel, Johann Eder, Christian Koncilia, Tadeusz Morzy, and Robert Wrembel. Creation and management of versions in multiversion data warehouse. In *SAC'04 : Proceedings of the 2004 ACM symposium on Applied computing*, pages 717–723, New York, NY, USA, 2004. ACM Press.

[Bel02] Zohra Bellahsene. Schema evolution in data warehouses. *Knowledge and Information Systems*, 4(3) :283–304, 2002.

[BMBT02] Mathurin Body, Maryvonne Miquel, Yvan Bédard, and Anne Tchounikine. A Multidimensional and Multiversion Structure for OLAP Applications. In *DOLAP'02 : 5th ACM International Workshop on Data Warehousing and OLAP*, 2002.

[BSH99] Markus Blaschka, Carsten Sapia, and Gabriele Höfling. On Schema Evolution in Multidimensional Databases. In *DaWaK'99 : 1st International Conference on Data Warehousing and Knowledge Discovery*, pages 153–164, 1999.

[BSSJ98] R. Bliujute, S. Saltenis, G. Slivinskas, and C. Jensen. Systematic Change Management in Dimensional Data Warehousing. In *3rd International Baltic Workshop on DB and IS*, pages 27–41, 1998.

[CT98] Luca Cabibbo and Riccardo Torlone. A Logical Approach to Multidimensional Databases. In *EDBT'98 : 6th International Conference on Extending Database Technology*, pages 183–197, 1998.

[GMR98] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. Conceptual Design of Data Warehouses from E/R Schemes. In *Hawaii Int. Conf. on System Sciences, vol. VII*, pages 334–343, 1998.

[HHNT86] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard. *Induction : processes of inference, learning, and discovery.* MIT Press, Cambridge, MA, USA, 1986.

[HMV99] Carlos A. Hurtado, Alberto O. Mendelzon, and Alejandro A. Vaisman. Updating OLAP Dimensions. In *DOLAP'99 : 2nd ACM International Workshop on Data Warehousing and OLAP*, pages 60–66, 1999.

[Kim96] R. Kimball. *The Data Warehouse Toolkit.* John Wiley & Sons, 1996.

[MV00] Alberto O. Mendelzon and Alejandro A. Vaisman. Temporal Queries in OLAP. In *VLDB'00 : 26th International Conference on Very Large Data Bases*, pages 242–253, 2000.

[MW04] Tadeusz Morzy and Robert Wrembel. On querying versions of multiversion data warehouse. In *DOLAP'04 : Proceedings of the 7th ACM international workshop on Data warehousing and OLAP*, pages 92–101, New York, NY, USA, 2004. ACM Press.

[MZ04] Elzbieta Malinowski and Esteban Zimányi. OLAP Hierarchies : A Conceptual Perspective. In *CAiSE*, pages 477–491, 2004.

[NSF+05] Ahlem Nabli, Ahlem Soussi, Jamel Feki, Hanêne Ben-Abdallah, and Faïez Gargouri. Towards an automatic data mart design. In *VIIth International Conference on Enterprise Information Systems (ICEIS 05), Miami, Florida, USA*, pages 226–231, 2005.

[PD02] Cassandra Phipps and Karen C. Davis. Automating Data Warehouse Conceptual Schema Design and Evaluation. In *DMDW*, pages 23–32, 2002.

[PJD01] Torben Bach Pedersen, Christian S. Jensen, and Curtis E. Dyreson. A Foundation for Capturing and Querying Complex Multidimensional Data. *Information Systems*, 26(5) :383–423, 2001.