Entrepôts de données pour l'aide à la décision médicale: conception et expérimentation

Serna Encinas María Trinidad

UNIVERSITÉ JOSEPH FOURIER

THÈSE

pour obtenir le grade de

Docteur de l'Université Joseph Fourier

Spécialité : Informatique

préparée au laboratoire LSR – IMAG
dans le cadre de l'École Doctorale
« Mathématiques, Sciences et Technologies de l'Information »

présentée et soutenue publiquement par

María Trinidad Serna Encinas

Le 27 Juin 2005

Entrepôts de données pour l'aide à la décision médicale : conception et expérimentation

Composition du jury

Mme. Christine Collet,	Président
Mme. Anne Doucet,	Rapporteur
M. Mokrane Bouzeghoub,	Rapporteur
M. Jacques Demongeot,	Examinateur
M. René Ecochard,	Examinateur
M. Michel Adiba,	Directeur de thèse

Remerciements

Je tiens à remercier Anne Doucet, professeur à l'Université Pierre et Marie Curie de Paris (LIP6), et Mokrane Bouzeghoub, professeur à l'Université de Versailles (PRiSM), pour avoir accepté d'être rapporteur et pour leurs commentaires et critiques constructives qui m'ont apporté un regard éclairant sur les contributions de cette thèse. Je remercie aussi Jacques Demongeot, professeur à l'Institut d'Ingénierie de l'Information de Santé de Grenoble (TIMC), et René Ecochard, professeur au Service de Biostatistique de Lyon (LBBE), d'avoir accepté de faire partie de mon jury.

Je remercie très sincèrement Christine Collet, professeur à l'Ecole Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble (ENSIMAG) et responsable du projet NODS, pour m'avoir accueilli dans son équipe et pour l'honneur qu'elle m'a fait en présidant le jury de thèse.

J'exprime ma profonde gratitude à Michel Adiba, professeur à l'Université Joseph Fourier de Grenoble (UJF), pour la confiance qu'il m'a toujours témoigné, ainsi que pour ses conseils, ses encouragements et sa patience. J'ose dire que vous devenez mon ami.

Je remercie également Claudia Lucia Roncancio, professeur à l'Ecole Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble (ENSIMAG), pour son aide, ses suggestions et principalement, pour son amitié.

Je tiens aussi à remercier aux membres et collaborateurs de l'équipe STORM, Genoveva Vargas-Solar, Cyril Labbe, Fabrice Jouanot, Christophe Bobineau, Khalid Belhajjame, Edgar Benitez, Tran-Kien Cao, Laurent d'Orazio, Levent Gurgen, Nagapraveen Jayaprakash, Thi-Huong-Giang Vu et Hanh Tan, pour la disponibilité qu'ils ont eue pendant la réalisation de ce travail.

Merci à tous les membres du Laboratoire Logiciels, Systèmes, Réseaux de l'IMAG, en particulier aux équipes administratives (Liliane Di Giacomo, Martine Pernice, Pascale Poulet et Carol Pasanisi) qui m'ont toujours aidée dans les démarches à suivre.

J'adresse mes plus vifs remerciements à Gennaro Bruno, Héctor Manuel Pérez

Urbina, Maria del Pilar Villamil, Sonia Mendoza-Chapa, Edicarsia Barbiero, Julio Moreno, Irma Ramirez, Patricia Quintero, Norma Pacheco, Sonia Meneses, Carmen Villarreal, Daniel Pérez et Francisca Lorena Zepeda, pour avoir partagé cette aventure avec moi et m'avoir toujours soutenue pendant les moments les plus difficiles de cette thèse. Pour moi, vous êtes déjà une partie de ma famille.

Je remercie ma mère, qui toujours m'a soutenue, mes soeurs et mes frères, pour leur amour inconditionnel et sans limites, pour leurs encouragements et pour leurs conseils. Je tiens à remercier spécialement à Migdia, Carmen, Rosa, et Juan, qui ont toujours cru en moi et qui m'ont soutenue tout ou long de ce travail.

A mes filles, dont l'existence donne un sens à ma vie. Tous mes remerciements sont à vous. Vous êtes la raison pour laquelle je me lève tous les jours de ma vie. *Pour ustedes, yo me levanto todos los dias de mi vida.*

A mon père, où qu'il soit. *A mi padre, donde quiera que esté.*

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Les entrepôts de données intègrent les informations en provenance de différentes sources, souvent réparties et hétérogènes et qui ont pour objectif de fournir une vue globale de l'information aux analystes et aux décideurs. Ces applications d'aide à la décision sont de type OLAP (*On-Line Analytical Processing* ou Analyse en ligne).

La construction et la mise en oeuvre d'un entrepôt de données représentent une tâche complexe qui se compose de plusieurs étapes. La première consiste à l'analyse des sources de données et à l'identification des besoins des utilisateurs. La deuxième correspond à l'organisation des données à l'intérieur de l'entrepôt. Finalement, la troisième consiste à établir divers outils d'interrogation (d'analyse, de fouille de données ou d'interrogation). Chaque étape présente des problématiques spécifiques. Ainsi, par exemple, lors de la première étape, la difficulté principale consiste en l'intégration des données, de manière à qu'elles soient de qualité pour leur stockage. Pour l'organisation, ils existent plusieurs problèmes comme : la sélection des vues à matérialiser, le rafraîchissement de l'entrepôt, la gestion de l'ensemble de données (courantes et historisées), entre autres. En ce qui concerne le processus d'interrogation, nous avons besoin des outils performants et conviviaux pour l'accès et l'analyse de l'information.

Notre travail se focalise principalement sur les deux dernières étapes, ainsi, pour le processus d'organisation, nous proposons la définition d'un modèle multidimensionnel, un algorithme pour la sélection optimale des vues à matérialiser et l'utilisation des versions de schémas bitemporels pour la gestion des aspects temporels des entrepôts. Pour l'interrogation, nous avons développé une interface graphique qui permet la génération semi-automatique des indicateurs. Dans les chapitres suivants, nous déployons chacune de nos propositions.

Cette thèse a eu pour cadre un projet médical. Ainsi, dans ce chapitre nous présentons d'abord une introduction des entrepôts dans le milieu médical et un résumé du projet ADELEM (Aide à la Décision Logistique et Médical). Ensuite, nous présentons la problématique et l'objectif de la thèse et pour finir, nous exposons la démarche suivie et notre contribution.

1.1 Les entrepôts de données pour l'aide à la décision

La définition classique d'un entrepôt donnée par [IH94] :

"Un Entrepôt de Données est une collection de données orientées sujet, intégrées, non volatiles et historisées, organisées pour le support d'un processus d'aide à la décision".

Nous détaillons ces caractéristiques [Fra97, DG01] :

orientées sujet : Les données des entrepôts sont organisées par sujet plutôt que par application. Par exemple, une chaîne de magasins d'alimentation organise les données de son entrepôt par rapport aux ventes qui ont été réalisées par produit et par magasin, au cours d'un certain temps.

intégrées : Les données provenant des différentes sources doivent être intégrées, avant leur stockage dans l'entrepôt de données. L'intégration (mise en correspondance des formats, par exemple), permet d'avoir une cohérence de l'information.

non volatiles : A la différence des données opérationnelles, celles de l'entrepôt sont permanentes et ne peuvent pas être modifiées. Le rafraîchissement de l'entrepôt, consiste à ajouter de nouvelles données, sans modifier ou perdre celles qui existent.

historisées : La prise en compte de l'évolution des données est essentielle pour la prise de décision qui, par exemple, utilise des techniques de prédiction en s'appuyant sur les évolutions passées pour prévoir les évolutions futures.

La construction d'un entrepôt revient à faire correspondre les besoins des utilisateurs avec la réalité des informations disponibles. Nous devons d'abord identifier et analyser les sources de données, ce qui nous permet de proposer les mécanismes adaptés selon les caractéristiques des informations. Ensuite, nous devons organiser l'ensemble de données à l'intérieur de l'entrepôt. Pour cela, nous devons d'abord structurer ces informations en considérant leur granularité. Ceci nous permet d'aboutir à la conception d'un schéma multidimensionnel qui permet de répondre aux besoins des utilisateurs.

La figure 1.1 montre une architecture générale d'un système décisionnel qui se compose de trois processus : extraction-intégration, organisation et interrogation.

Nous trouvons le processus d'extraction-intégration entre les sources de données et l'entrepôt. Ce processus est responsable de l'identification des données dans les diverses sources internes et externes ; de l'extraction de l'information qui nous intéresse et de la préparation et de la transformation (nettoyage, filtrage,...) des données.

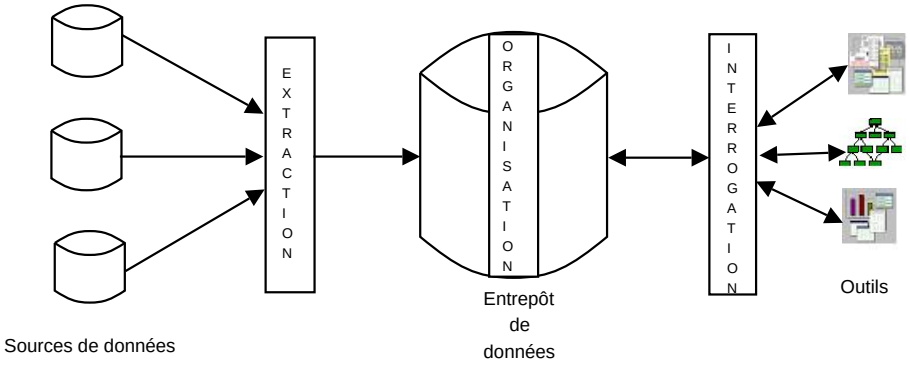

FIG. 1.1 – Architecture générale d'un système décisionnel

A l'intérieur de l'entrepôt, nous trouvons le processus d'organisation, il est responsable de structurer les données par rapport à leur niveau de granularité (agrégats).

Finalement, le troisième processus correspond à l'interrogation qui se place entre l'entrepôt et les différents outils pour arriver à l'analyse des données, pour les différents utilisateurs de l'entreprise.

Malgré une conception attentive et soigneuse, la structure d'une base de données est sujette à de nombreux changements. Bien que les estimations diffèrent, la plupart s'accordent sur le fait que plus de 50% du travail des développeurs se focalise sur les modifications du système après leur implantation [Rod95]. Ces changements peuvent affecter aussi bien les données que leur structure. En effet, l'évolution d'un schéma concerne tout le cycle de vie d'un entrepôt de données. Un de premiers travaux dans ce contexte est [HMV99a, HMV99b], où les auteurs proposent des opérateurs pour l'évolution des dimensions.

1.2 Le projet ADELEM : Aide à la Décision Logistique et Médicale

Cette thèse a eu pour cadre le projet ADELEM[1] qui consiste en la mise au point d'outils logiciels nécessaires à l'aide à la décision logistique et médicale, dans le cadre des SROS (Schémas d'Organisation Sanitaire et Sociale) gérés par les ARH (Agences Régionales d'Hospitalisation). Dans le système de soins d'une région, ces agences sont chargées de répartir de manière optimale les moyens médicaux (l'"offre") en fonction des besoins sanitaires (la "demande"). A partir des observations de santé constituant l'information primaire, essentiellement contenue dans les données PMSI (Programme de Médicalisation du Système d'Information des hôpitaux) [PMS04], il

[1]Action Concertée Incitative "Technologies pour la Santé". Projet ADELEM. 2001

est nécessaire de mettre au point des outils logiciels d'analyse et de visualisation.

Ce projet est traité par une équipe interdisciplinaire composée de :

- Le Laboratoire TIMC IMAG, UMR CNRS 5525.

- Le Laboratoire de Biométrie et Biologie Évolutive (UMR CNRS 5558) qui fournit l'analyse statistique des données médicales.

- L'Organisation Mondiale de la Santé (basé en Suisse) qui a développé le logiciel HealthMapper pour la création et la manipulation des données géographiques.

- Notre équipe du Laboratoire LSR IMAG, UMR CNRS 5526 qui fournit le support pour les bases de données et les entrepôts.

Objectif du projet :

La répartition de l'offre de soins (nombre de lits par spécialité, plateaux techniques) doit s'adapter régulièrement à l'évolution de la demande (évolution des pathologies nécessitant une prise en charge hospitalière, de la structure d'âge et de l'implantation géographique de la population), et de l'offre de soins (évolution des techniques et de l'organisation du système de santé, telles que réseau et télé-médecine). Le projet consiste donc en la mise au point d'un outil de modélisation-simulation à partir des données du PMSI et des données de l'INSEE (Institut National de la Statistique et des Etudes Economiques) [INS05] pour aider les gestionnaires du système de santé dans leur démarche de prise de décision.

Sources de données réelles :

Nous avons trois types de sources : géographiques, démographiques et les données publiques concernant la santé. Elles sont réparties, hétérogènes et certaines d'entre elles sont externes au domaine médical proprement dit. En ce qui concerne les données géographiques, l'équipe de Service de Biostatistique à Lyon s'est focalisé sur la visualisation (représentation cartographique) de l'offre et de la consommation de soins. Ainsi, dans la suite, nous décrivons ce travail.

Description de l'activité de l'équipe de Lyon (Service de Biostatistique)

Ils ont abouti au développement d'un logiciel de cartographie de l'offre et de la demande de soins hospitaliers en France. Dans cette version initiale, ils se sont intéressés à l'offre et à la demande de soins, aux besoins et aux flux mais pas à la comparaison sur une même carte de l'offre et de la consommation (ratios) qui nécessite de définir un maillage géographique commun. Ainsi les représentations graphiques de l'offre et de la demande se feront sur des fonds de cartes différents. En effet,

les renseignements sur l'origine géographique des patients pour la consommation de soin sont basés sur les codes postaux et non pas le code INSEE de la commune de résidence du patient [Bel02].

La carte de France se décompose en quatre niveaux administratifs : la commune, le canton, le département et la région. Il existe une hiérarchie entre ces niveaux : la commune est incluse dans le canton, lui-même est inclus dans le département, et celui-ci appartient à une région. Néanmoins, nous ne disposons pas encore de la correspondance entre les communes et les cantons au niveau national (fichier de l'Intitut de Géographie National). Ainsi, dans cette version 0, la France se compose de trois niveaux administratifs :

- niveau 1 correspondant à Admin1 : région
- niveau 2 correspondant à Admin2 : département
- niveau 3 correspondant à Admin3 : commune

Pour la création du logiciel, ils ont utilisé le système HealthMapper (logiciel de cartographie). Il se base sur un gestionnaire de données qui permet de combiner les informations géographiques et épidémiologiques. Le nombre d'indicateurs prévus pour cette version 0 est limité à 2 pour la visualisation de l'offre et de la consommation de soins. Cette version permet de représenter tous les établissements de courts séjours au niveau national ainsi que le nombre de lits de chaque établissement hospitalier. Pour ce dernier, la figure 1.2 montre la carte obtenue.

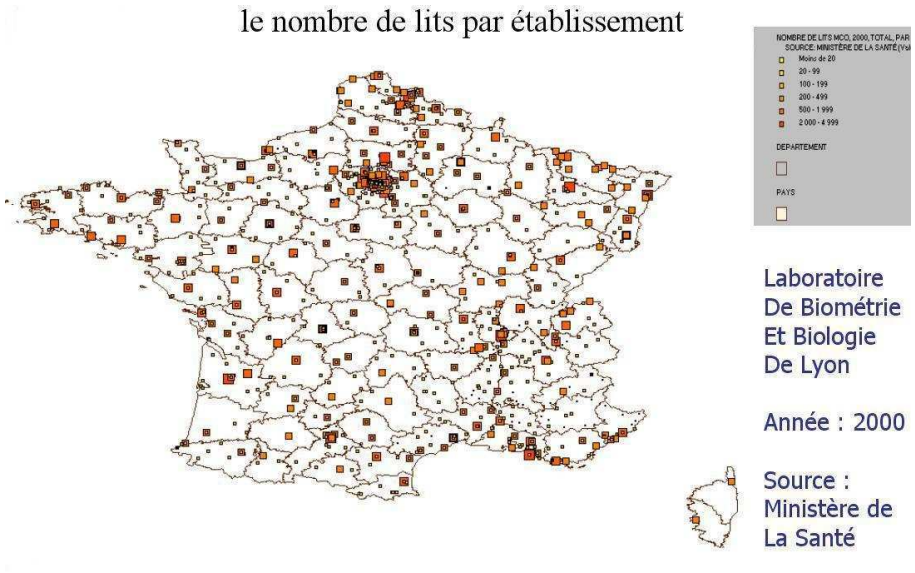

FIG. 1.2 – Tous les établissements au niveau national

1.3 Problématique et objectif de la thèse

Les entrepôts de données ont été conçus pour l'aide à la décision. Ils intègrent les informations en provenance des différents systèmes transactionnels de l'entreprise. L'ensemble des données, y compris leur historique, est utilisé pour faire des calculs prévisionnels, des statistiques ou pour établir des stratégies de développement et d'analyses des tendances.

Dans le cadre du projet ADELEM, nous nous proposons d'adapter notre savoir-faire au problème de la gestion de données médicales qui constituent un cadre applicatif particulièrement intéressant. En effet, ces données se trouvent réparties dans plusieurs sources qu'il faut, dans un premier temps, fédérer pour constituer un entrepôt de données pertinentes pour l'application visée. Cette étape est importante car elle doit non seulement identifier les sources, mais aussi déterminer comment extraire de ces sources les données désirées. Nous devons déterminer si les données doivent être extraites telles quelles ou bien s'il faut les traiter au préalable en leur appliquant des fonctions spécifiques comme des agrégats, des sommes, ... En plus, nous devons établir un mécanisme pour la gestion de l'évolution. Dans ce cas, il faut déterminer l'adaptation au niveau : de l'application d'extraction, des agrégats et de l'application d'analyse.

Cette problématique est générale à la constitution de tout entrepôt mais nous devons ici tenir compte de la nature particulière des données sur lesquelles portent l'étude : type, format, sémantique, confidentialité, degré de fiabilité et de confiance, informations manquantes ou incomplètes, ... En résumé, il s'agit de constituer un entrepôt qui contient des données pertinentes et de qualité sur lesquelles sera basé l'outil d'interrogation et le processus d'aide à la décision.

Précedemment, nous avons dit que notre travail s'encadre principalement aux étapes d'organisation et d'interrogation. Pour cela, nous nous sommes focalisés sur les problématiques suivantes : la conception d'un entrepôt de données, la sélection des vues à matérialiser et la gestion de l'évolution. Ainsi, nous traitons d'abord ces sujets et nous terminons avec l'objectif de notre thèse.

1.3.1 Conception d'un système pour le décisionnel

La conception et la construction d'un entrepôt de données est une tâche complexe et délicate. Dans [Kim96, KR03], nous trouvons une méthodologie descendante pour la conception d'un entrepôt. Elle se compose de neuf décisions majeures qui sont à la base d'une conception complète. Les cinq premières portent sur la structure logique, tandis que les autres quatre traitent sur la structure physique.

1) L'identification des tables de faits.
2) Le niveau de granularité de chaque table de faits.
3) Les dimensions appartenant à la table de faits.

4) L'ensemble de mesures.
5) Les attributs des dimensions avec des descriptions complètes.
6) La gestion de l'évolution.
7) La définition des agrégats, des dimensions dégénérées et des dimensions hétérogènes.
8) L'étendue historique des données.
9) La périodicité des mises à jour.

Nous donnons une description de ces décisions :

A partir de l'analyse des sources de données existantes et réellement utilisées, nous pouvons aboutir aussi bien à l'identification des tables de faits, qui représentent le sujet d'analyse, qu'à son niveau de granularité. Lorsque le grain de la table de faits est connu, les dimensions peuvent être identifiées. Le choix des dimensions est le point clé de la conception, car elles modélisent les perspectives de l'analyse. Une fois les dimensions choisies, nous définissons l'ensemble de mesures, qui représentent les différentes valeurs de l'activité analysée. Nous devons aussi enregistrer pour chaque dimension l'ensemble des attributs textuels. Les attributs peuvent être utilisés comme source de restrictions dans une requête.

Pour la gestion de l'évolution à l'intérieur des dimensions, Kimball [Kim96] propose trois solutions qui assurent un suivi des modifications dans le temps :

1) Remplacer des valeurs anciennes par des nouvelles dans l'enregistrement de la dimension, néanmoins, nous perdons la possibilité de suivre les événements passés.

2) Créer des nouveaux enregistrements de dimension lors du changement qui contiennent les nouvelles valeurs de l'attribut. Ceci équivaut à segmenter l'historique selon l'ancienne et la nouvelle description.

3) Créer des nouveaux champs "actuels" à l'intérieur de l'enregistrement d'origine de la dimension, tout en conservant en même temps les premières valeurs enregistrées. Dans ce cas, nous pouvons garder seulement la valeur originale et la courante de l'attribut modifié. Les valeurs intermédiaires sont perdues.

Une dimension qui contient un seul attribut représente une dimension dégénérée. De cette manière, nous gardons l'attribut (la clé de la dimension) à l'intérieur de la table de faits.

Nous décrirons une dimension hétérogène en utilisant un exemple. Une compagnie d'assurances a en général des domaines d'activité très différenciés. Il y a par exemple : les risques habitation, les risques automobiles, les risques objets personnels, la responsabilité civile, ..., où chaque domaine est représenté par des attributs particuliers. Nous identifions deux dimensions : une pour le bien assuré et une autre

pour le risque couvert. Néanmoins, la dimension `Bien_assure` contient l'ensemble d'attributs des produits hétérogènes (habitation, automobile, ...) ce qui fait d'elle une dimension hétérogène. Kimball propose de faire des dimensions particularisées, ce qui signifie de faire des "copies" de la dimension `Bien_assure` pour chaque domaine d'activité, ainsi, pour chaque type de risque couvert nous créons des dimensions particularisées pour `Bien_assure` et pour `Risque_couvert`.

Finalement, nous devons établir le période des données historisées ainsi que la fréquence des mises à jour. Cette dernière peut être journalière, hebdomadaire, mensuelle,...

1.3.2 Sélection des vues à matérialiser

La sélection de l'ensemble optimal des vues à matérialiser fait partie du processus d'organisation et elle représente une tâche essentielle dans la conception d'un entrepôt. La matérialisation nous permet d'optimiser l'exécution d'une requête, pour cela, nous avons les possibilités suivantes :

La matérialisation complète : Cette approche donne le meilleur temps de réponse de la requête. Néanmoins, la matérialisation complète n'est pas faisable, due principalement au coût de stockage élevé.

Pas de matérialisation : Dans ce cas, nous n'avons pas besoin de stockage supplémentaire. Cependant, pour répondre à chaque requête, nous devons chercher l'information au niveau plus bas de granularité (i.e., *raw data*). Ainsi, nous avons le problème d'un coût de calcul très élevé pour chaque requête.

La matérialisation partielle : Dans cette approche, le problème se réduit à déterminer le nombre optimal de vues à matérialiser, en considérant leur coût de stockage, de maintenance, ...

Il est évident que la dernière approche est la plus intéressante. Elle fait l'objet de plusieurs recherches qui ont pour objectif d'aboutir à la définition d'un mécanisme pour la sélection d'un ensemble optimal des vues à matérialiser. La plupart des travaux utilisent un modèle de coût mais qui a été adapté avec l'inclusion de certains paramètres, tels que : la fréquence de la requête, la fréquence des mises à jour sur les relations de base, le coût de maintenance, entre autres.

Pour finir, nous voulons faire une remarque. Quand nous parlons de la possibilité de rien matérialiser, nous nous référons au fait que nous ne considérons pas les données du niveau plus bas comme des données matérialisées. En effet, certains d'entre nous visualisent les données de détail de l'entrepôt comme une sorte de matérialisation de l'information qui se trouve dans les sources de données. Néanmoins, pour nous, ces données une fois intégrées et stockées dans l'entrepôt, représentent

l'ensemble de relations de base à partir desquelles nous sélectionnerons les vues à matérialiser.

1.3.3 Gestion de l'évolution des entrepôts

Le problème d'évolution de schéma apparaît quand un changement au schéma d'une base de données modifie l'ensemble de vues définies sur ce schéma. Par exemple : Supposons que S_1 représente un schéma et V_1 l'ensemble de vues sur S_1. Si nous avons une nouvelle version de schéma S_2 à partir de S_1, le problème est la définition de la nouvelle version V_2 qui soit cohérente avec S_2 [Ber03].

L'évolution d'un schéma a des conséquences sur l'application chargée de l'extraction et l'intégration de données des sources, car elle peut devenir incomplète ou incohérente vis-à-vis du nouveau schéma de l'entrepôt. Cette évolution entraîne aussi l'adaptation des agrégats pré-calculés et l'adaptation du processus de maintenance.

La figure 1.3 montre les adaptations nécessaires après qu'une évolution du schéma a eu lieu dans un entrepôt de données.

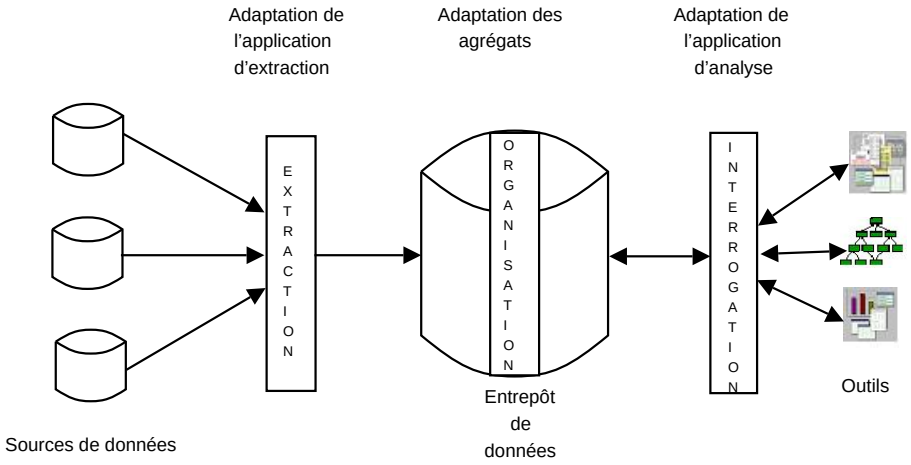

FIG. 1.3 – Gestion de l'évolution du schéma dans un entrepôt

Pour résoudre le problème de perte de données après des changements du schéma, le concept d'évolution de schéma à été introduit pour récupérer les données existantes par le biais de leur adaptation au nouveau schéma. Néanmoins, dans les systèmes qui doivent gérer des données historiques, l'évolution de schéma n'est pas suffisante et la maintenance de plusieurs schémas est requise [GM02]. Ainsi, pour gérer ces changements, nous pouvons utiliser soit l'évolution de schémas soit les versions de schémas. Nous devons choisir la technique à utiliser par rapport aux caractéristiques de l'application cible.

1.3.4 Objectif de la thèse

Notre objectif consiste à la conception et mise en oeuvre d'un entrepôt de données pour l'aide à la décision. Pour faire cela, nous proposons :

- La définition d'un métamodèle multidimensionnel qui se compose de trois classes : Cube, Dimension et Hiérarchie.

- Un algorithme pour la sélection de l'ensemble optimal des vues à matérialiser.

- Une interface graphique pour la génération (semi-automatique) des requêtes.

- Les versions de schémas bitemporels pour la gestion de l'évolution d'un entrepôt.

1.4 Démarche et contribution

Notre travail se focalise sur les processus d'organisation et d'interrogation. Nous reprenons les propositions énumérées précédemment et nous donnons une description de chacune d'elles.

1.4.1 Métamodèle multidimensionnel

Pour le métamodèle, nous avons spécifié trois métaclasses : Cube, Dimension et Hiérarchie. Nous avons proposé les définitions pour chaque métaclasse et leurs instances, ainsi qu'une définition d'une base de données multidimensionnelle et de son instance. Ceci nous a permis d'aboutir à la conception d'un modèle multidimensionnel qui se compose d'un ensemble de classes contenant des propriétés, des relations et des contraintes entre les classes. Nous distinguons les termes de dimension et de hiérarchie avec les caractéristiques suivantes :

- Une dimension peut contenir zéro, une ou plusieurs hiérarchies. Une instance de hiérarchie ne peut pas relier l'instance du cube auquel est associée l'instance de la dimension contenant cette hiérarchie. Néanmoins, elle peut relier une instance d'un autre cube et ceci en raison de la granularité associée à chaque hiérarchie.

- Nous pouvons insérer un niveau de hiérarchie soit à côté de la hiérarchie déjà définie soit à l'intérieur.

- Nous considérons le cas où nous avons seulement deux niveaux de hiérarchie.

Nous avons conçu un schéma en constellation pour le projet ADELEM et nous avons donné une description de ce schéma en utilisant notre métamodèle.

1.4.2 Algorithme pour la sélection des vues à matérialiser

Nous avons dit précédemment que pour la matérialisation d'un hypercube, nous avions les possibilités suivantes : la matérialisation complète, pas de matérialisation ou la matérialisation partielle. Dans notre cas, nous considérons la dernière approche.

Notre algorithme utilise un modèle de coût, néanmoins, nous considérons aussi les paramètres suivants : fréquence d'utilisation, coût de calcul et fréquence des mises à jour des relations de base.

Nous avons utilisé notre algorithme et celui proposé dans [HRU95] sur des données médicales réelles et nous avons constaté que notre proposition donne de meilleurs résultats pour notre cas expérimental.

1.4.3 Interface pour la génération (semi-automatique) des indicateurs

L'objectif de notre interface est de faciliter la tâche de création d'indicateurs pour les utilisateurs. Nous avons développé les composants suivants :

- Un module pour la création et/ou la modification du schéma.

- Un module pour la génération de requêtes de manière quasi-automatique.

Nous avons construit le schéma en étoile `Adelem_MCO` et nous avons crée la base de données correspondante en utilisant un échantillon de 10% des données réelles du projet ADELEM. Ceci nous a permis de vérifier et de prouver notre interface graphique, de cette manière, la requête SQL générée s'exécute sur les données médicales et le résultat est présenté sur l'interface.

1.4.4 Versions de schémas bitemporels

Nous proposons l'utilisation des versions de schémas bitemporels pour la gestion, le stockage et la visualisation des données historisées intensionnelles et extensionnelles. Dans ce cas, nous devons gérer la combinaison du temps de transaction et du temps de validité dans notre modèle de versions.

Néanmoins, la gestion des versions est une tâche complexe et elle présente plusieurs problèmes. Le principal, à notre connaissance, est l'explosion des versions, ainsi, nous devons établir un mécanisme pour contrôler cette explosion. Notre proposition considère :

- Un opérateur appelé *SetVersion* qui permet d'appliquer un ensemble de changements sur une version et de générer une nouvelle version.

- Un ensemble de 19 primitives nécessaires pour la gestion des cubes, des dimensions et des hiérarchies.

- Un ensemble de 5 primitives qui agissent sur les versions de schémas.

- Un gestionnaire d'évolution responsable de la manipulation des différentes versions de schémas historisées.

1.5 Plan de la thèse

Nous présentons ici l'organisation du document.

Le chapitre 2 fait un état de l'art des entrepôts de données dans le contexte où nous nous plaçons. Nous présentons l'architecture d'un entrepôt et ses composants, les différents modèles multidimensionnels pour la construction d'un système décisionnel, ainsi que l'ensemble des opérations pour la manipulation des données multidimensionnelles. Nous décrivons les serveurs décisionnels : ROLAP, MOLAP et HOLAP. Dans la dernière partie, nous présentons l'approche des versions de schéma. Nous présentons d'abord l'état courant des versions dans les modèles de données existants. Ensuite nous décrivons l'espace de versions, leurs types, la gestion de données extensionnelles et intensionnelles, ainsi que les opérations primitives à utiliser.

Le chapitre 3 présente l'architecture conçue pour un système décisionnel dans le domaine médical. Elle est basée sur trois composants, une interface graphique pour la génération semi-automatique des indicateurs, un entrepôt multidimensionnel et les versions de schémas bitemporels. Nous décrivons notre métamodèle et ses métaclasses. Nous présentons le schéma en constellation conçu composé de trois sous-schémas : `Prise_MCO, Prise_SSR` et `Population`, où chacun est composé soit de ses propres dimensions soit des dimensions partagées.

Le chapitre 4 traite le sujet des vues matérialisées, aussi bien leur sélection que leur création. Nous présentons d'abord l'hypercube pour le schéma `Adelem_MCO`. Ensuite, nous présentons un algorithme pour la sélection de l'ensemble optimal des vues à sélectionner. Nous terminons ce chapitre avec une description du fonctionnement de l'interface pour la génération semi-automatique d'indicateurs.

Le chapitre 5 présente notre proposition pour la gestion, le stockage et la visualisation de l'ensemble de données historisées. Pour la création et la gestion des versions de schémas, nous donnons une liste d'opérations primitives adaptées aux entrepôts de données et nous avons aussi défini un ensemble de contraintes pour la manipulation des versions. Nous donnons aussi pour chaque primitive sa définition formelle. La dernière partie traite de la description du gestionnaire d'évolution de schémas. Nous présentons aussi bien la gestion de données historisées que l'activité

du gestionnaire d'évolution par niveau.

Finalement, le chapitre 6 présente nos conclusions et quelques perspectives.

Chapitre 2

Entrepôts de données multidimensionnelles et aspects temporels

Les entrepôts de données sont apparus vers les années 1990 en réponse à la nécessité de rassembler toutes les informations de l'entreprise en une base de données unique destinée aux analystes et aux gestionnaires [Cod93, DG01]. L'ensemble des données, y compris leur historique, est utilisé dans de nombreux domaines, tels que : l'analyse de données et l'aide à la décision (gestion et analyse de marché, gestion et analyse du risque, gestion et détection des fraudes,...) ; dans autres applications (recherches dans des textes, dans les documents web, dans l'astronomie,...) [Adi02, BCA01].

Dans ce chapitre, nous analysons aussi bien les caractéristiques des entrepôts que leurs aspects temporels.

2.1 Entrepôts de données

Nous présentons d'abord l'architecture d'un système décisionnel qui se compose de trois composants : les sources, l'entrepôt et les outils pour l'interrogation de l'ensemble de données. Nous décrivons aussi les caractéristiques des entrepôts et les bases de données.

2.1.1 Architecture d'un entrepôt de données

L'architecture des entrepôts de données repose souvent sur un SGBD séparé du système de production de l'entreprise qui contient les données de l'entrepôt. Le processus d'extraction des données permet d'alimenter périodiquement ce SGBD. Néanmoins avant d'exécuter ce processus, une phase de transformation est appliquée aux données opérationnelles. Celle-ci consiste à les préparer (mise en correspondance des formats de données), les nettoyer, les filtrer,..., pour finalement aboutir à leur

stockage dans l'entrepôt.

Dans la Figure 2.1, nous présentons une architecture simplifiée d'un entrepôt [DG01]. Les différents composants ont été intégrés dans trois parties : les sources de données, l'entrepôt et les outils existants dans le marché.

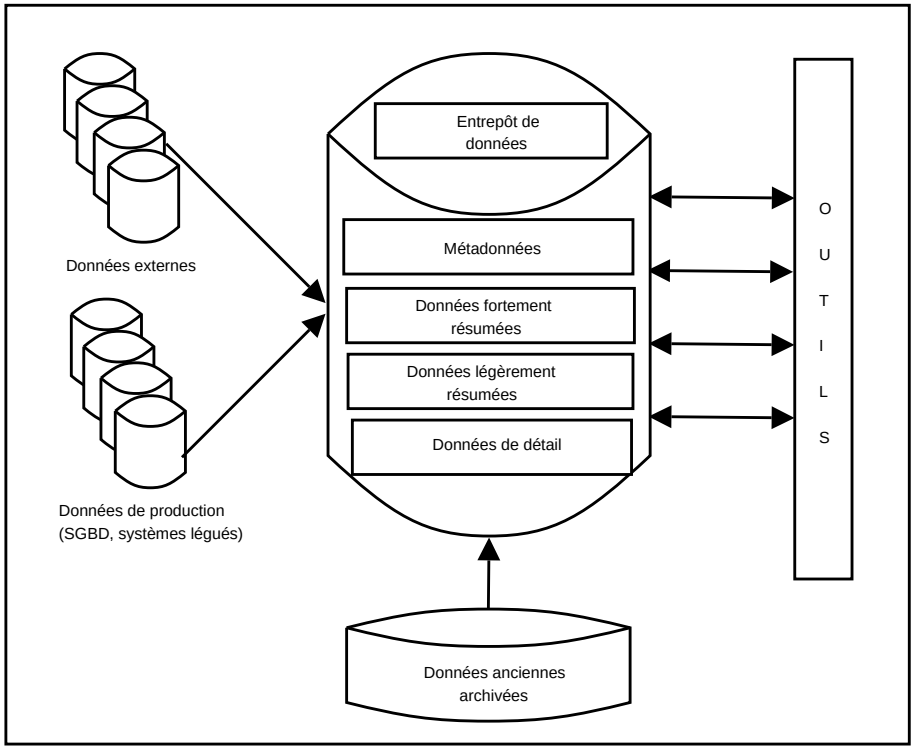

FIG. 2.1 – Architecture d'un entrepôt de données

a) Les sources : Les données de l'entrepôt sont extraites de diverses sources souvent réparties et hétérogènes, et qui doivent être transformées avant leur stockage dans l'entrepôt. Nous avons deux types de sources des données : internes et externes à l'organisation.

Internes : La plupart des données sont saisies à partir des différents systèmes de production qui rassemblent les divers SGBD opérationnels, ainsi que des anciens systèmes de production qui contiennent des données encore exploitées par l'entreprise.

Externes : Ils représentent des données externes à l'entreprise et qui sont souvent achétées. Par exemple, les sources de données démographiques.

b) L'entrepôt de données : Il existe plusieurs types de données dans un entrepôt, qui correspondent à diverses utilisations, comme :

Données de détail courantes : Ce sont l'ensemble des données quotidiennes et plus couramment utilisées. Ces données sont généralement stockées sur le disque pour avoir un accès rapide. Par exemple, le détail des ventes de l'année en cours, dans les différents magasins.

Données de détail anciennes : Ce sont des données quotidiennes concernant des événements passés, comme par exemple le détail des ventes des deux dernières années. Nous les utilisons pour arriver à l'analyse des tendances ou des requêtes prévisionnelles. Néanmoins ces données sont plus rarement utilisées que les précédentes, et elles sont souvent stockées sur des mémoires d'archives.

Données résumées ou agrégées : Ce sont des données moins détaillées que les deux premières et elles permettent de réduire le volume des données à stocker. Le type de données, en fonction de leur niveau de détail, permet de les classifier comme des données légèrement ou fortement résumées. Par exemple, les ventes mensuelles par magasin des dix dernières années sont des données faiblement résumées, tandis que les ventes semestrielles, par région, des dix dernières années sont fortement résumées.

Les métadonnées : Ce sont des données essentielles pour parvenir à une exploitation efficace du contenu d'un entrepôt. Elles représentent des informations nécessaires à l'accès et l'exploitation des données dans l'entrepôt comme : la sémantique (leur signification), l'origine (leur provenance), les règles d'agrégation (leur périmètre), le stockage (leur format, par exemple : francs, euro,...) et finalement l'utilisation (par quels programmes sont-elles utilisées).

c) Outils : Il existe sur le marché différents outils pour l'aide à la décision, comme les outils de fouille de données ou datamining (pour découvrir des liens sémantiques), outils d'analyse en ligne (pour la synthèse et l'analyse des données multidimensionnelles), outils d'interrogation (pour faciliter l'accès aux données en fournissant une interface conviviale au langage de requêtes),... [CD97, Fra97, DG01].

2.1.2 Entrepôts et les bases de données

Dans l'environnement des entrepôts de données, les opérations, l'organisation des données, les critères de performance, la gestion des métadonnées, la gestion des transactions et le processus de requêtes sont très différents des systèmes de bases de données opérationnels. Par conséquent, les SGBD relationnels orientés vers l'environnement opérationnel, ne peuvent pas être directement transplantés dans un système d'entrepôt de données [WB97].

Les SGBD ont été crées pour les applications de gestion de systèmes transactionnels. Par contre, les entrepôts de données ont été conçus pour l'aide à la prise

de décision. Ils intègrent les informations qui ont pour objectif de fournir une vue globale de l'information aux analystes et aux décideurs.

Le tableau 2.1 résume ces différences entre les systèmes de gestion de bases de données et les entrepôts de données [DG01].

	SGBD	**Entrepôts de données**
Objectifs	Gestion et production	Consultation et analyse
Utilisateurs	Gestionnaires de production	Décideurs, analystes
Taille de la base	Plusieurs gigaoctets	Plusieurs teraoctets
Organisation des données	Par traitement	Par métier
Type de données	Données de gestion (courantes)	Données d'analyse (résumées, historisées)
Requêtes	Simples, prédéterminées, données détaillées	Complexes, spécifiques, agrégations et *group by*
Transactions	Courtes et nombreuses, temps réel	Longues, peu nombreuses

TAB. 2.1 – Différences entre SGBD et entrepôts de données

Systèmes transactionnels et systèmes décisionnels Les SGBD ont été créés pour gérer de grands volumes d'information contenus dans les différents systèmes opérationnels qui appartiennent à l'entreprise. Ces données sont manipulées en utilisant des processus transactionnels en ligne [Cod93].

Parallèlement à l'exploitation de l'information contenue dans ces systèmes opérationnels, les dirigeants des entreprises ont besoin d'avoir une vision globale concernant toute cette information pour faire des calculs prévisionnels, des statistiques ou pour établir des stratégies de développement et d'analyses des tendances.

Le tableau 2.2 compare les caractéristiques des systèmes transactionnels et décisionnels par rapport aux données et aux utilisateurs [BCA01, CT98, GM00, SMKK98, Tes00, ZS99].

2.2 Modélisation multidimensionnelle

Pour arriver à construire un modèle approprié pour un entrepôt de données, nous pouvons choisir, soit un schéma relationnel (le schéma en étoile, en flocon de neige ou en constellation) ; soit un schéma multidimensionnel. Avant de décrire les différents schémas, nous commençons par quelques concepts de base.

	S. Transactionnel	**S. Décisionnel**
Données	Exhaustives Courantes Dynamiques Orientées applications	Résumées Historiques Statiques Orientées sujets (d'analyse)
Utilisateurs	Nombreux Variés (employés, directeurs,...) Concurrents Mises à jour et interrogations Requêtes prédéfinies Réponses immédiates Accès à peu d'information	Peu nombreux Uniquement les décideurs Non concurrents Interrogations Requêtes imprévisibles et complexes Réponses moins rapides Accès à de nombreuses informations

TAB. 2.2 – Comparaison des systèmes

La modélisation multidimensionnelle consiste à considérer un sujet analysé comme un point dans un espace à plusieurs dimensions. Les données sont organisées de manière à mettre en évidence le sujet (le **fait**) et les différentes perspectives de l'analyse (les **dimensions**).

En partant de cette définition, nous remarquons les concepts de fait et de dimension. Le **fait** représente le sujet d'analyse. Il est composé d'un ensemble de **mesures** qui représentent les différentes valeurs de l'activité analysée. Par exemple, dans le fait Ventes (*cf.* Figure 2.2), nous pouvons avoir la mesure "Quantité de produits vendus par magasin". Les mesures doivent être valorisées de manière continue [AGS97, CT97, Inm92, Inm95, Kim96, KR03, KS95] et elles peuvent être additives (pour résumer une grande quantité d'enregistrements) ; semi-additives (si elles peuvent seulement être additionnées pour certaines dimensions) et non additives.

Une dimension modélise une perspective de l'analyse. Elle se compose de **paramètres** (ou attributs) qui servent à enregistrer les descriptions textuelles. Nous pouvons utiliser les paramètres textuels comme source des restrictions dans une requête. Par exemple, dans la requête "Quantité de produits vendus dans la région Rhône Alpes durant le mois de Janvier 2000". Nous trouvons les paramètres : région "Rhône Alpes" et mois "Janvier 2000".

Une **hiérarchie** représente les paramètres d'une dimension selon leur niveau de granularité ou de détail. Les paramètres sont ordonnés par une relation "**est _plus_ fin**" et notée **P1** $\longrightarrow$ **P2**. Dans notre exemple sur les ventes (*cf.* figure 2.2) nous pouvons avoir une hiérarchie pour la dimension `Magasin` de la forme suivante :

Commune ⟶ Département ⟶ Région ⟶ Pays

2.2.1 Schémas relationnels

Dans les schémas relationnels nous trouvons deux types de schémas. Les premiers sont des schémas qui répondent fort bien aux processus de type OLTP qui ont été décrits précédemment, alors que les deuxièmes, que nous appelons des schémas pour le décisionnel, ont pour but de proposer des schémas adaptés pour des applications de type OLAP.

Nous décrivons les différents types des schémas relationnels pour le décisionnel.

2.2.1.1 Le schéma en étoile

Il se compose du fait central et de leurs dimensions. Dans ce schéma il existe une relation pour les faits et plusieurs pour les différentes dimensions autour de la relation centrale. La relation de faits contient les différentes mesures et une clé étrangère pour faire référence à chacune de leurs dimensions.

La figure 2.2 montre le schéma en étoile en décrivant les ventes réalisées dans les différents magasins de l'entreprise au cours d'un jour. Dans ce cas, nous avons une étoile centrale avec une table de faits appelée `Ventes` et autour leurs diverses dimensions : `Temps`, `Produit` et `Magasin`.

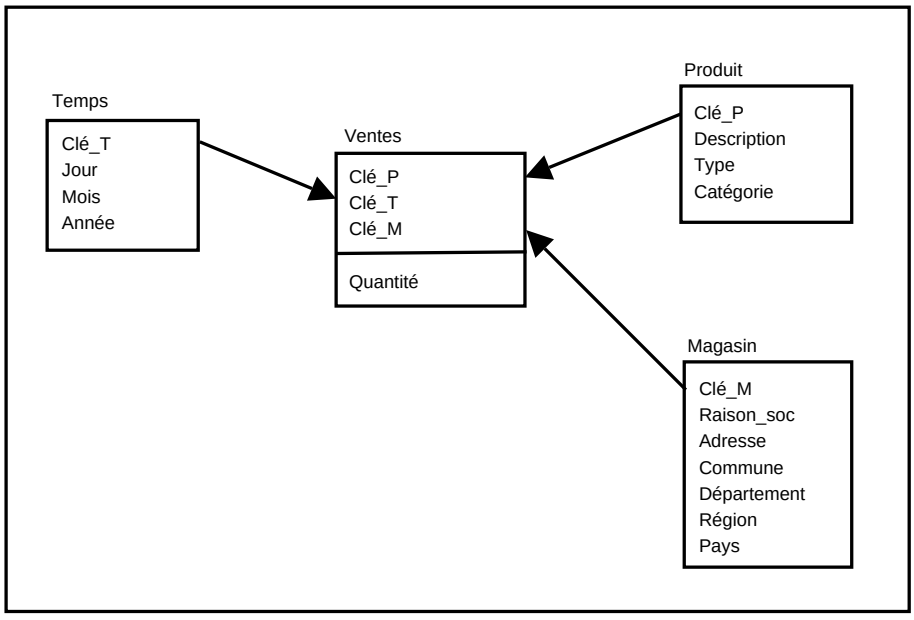

FIG. 2.2 – Exemple de modélisation en étoile.

2.2.1.2 Le schéma en flocon de neige (Snowflake)

Il dérive du schéma précédent avec une relation centrale et autour d'elle les différentes dimensions, qui sont éclatées ou décomposées en sous hiérarchies. L'avantage du schéma en flocon de neige est de formaliser une hiérarchie au sein d'une dimension, ce qui peut faciliter l'analyse. Un autre avantage est représenté par la normalisation des dimensions, car nous réduisons leur taille. Néanmoins dans [Kim96], l'auteur démontre que c'est une perte de temps de normaliser les relations des dimensions dans le but d'économiser l'espace disque. Par contre, cette normalisation rend plus complexe la lisibilité et la gestion dans ce type de schéma. En effet, ce type de schéma augmente le nombre de jointures à réaliser dans l'exécution d'une requête.

Les hiérarchies pour le schéma en flocon de neige de l'exemple de la figure 2.2 sont :

Dimension `Temps` = Jour $\longrightarrow$ Mois $\longrightarrow$ Année

Dimension `Magasin` = Commune $\longrightarrow$ Département $\longrightarrow$ Région $\longrightarrow$ Pays

La figure 2.3 montre le schéma en flocon de neige avec les dimensiones `Temps` et `Magasin` éclatées en sous hiérarchies.

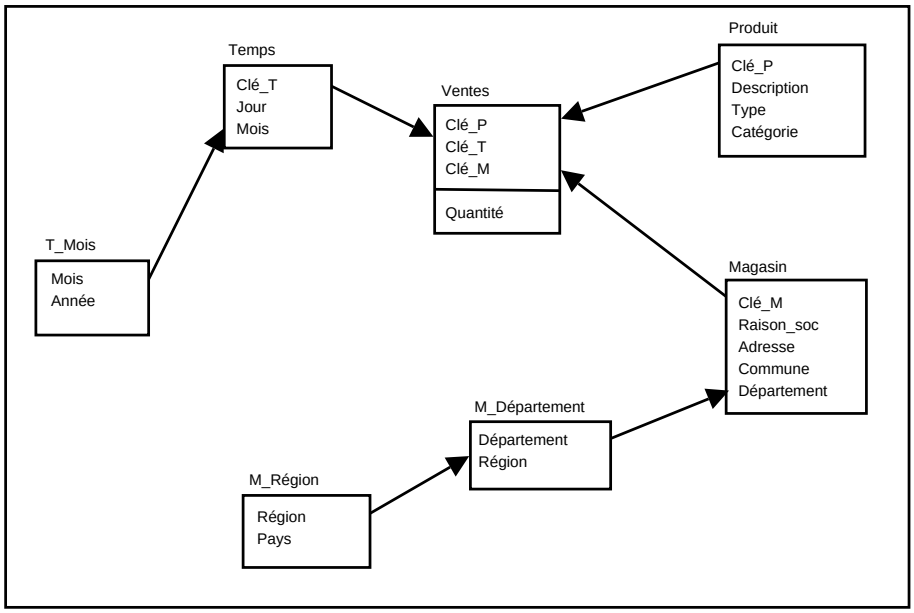

FIG. 2.3 – Exemple de modélisation en flocon de neige.

Dans l'exemple ci-dessus, la dimension `Temps` a été éclatée en deux, `Temps` et `T_Mois`. La deuxième dimension `Magasin`, à été décomposée en trois : `Magasin`, `M_Département` et `M_Région`.

2.2.1.3 Le schéma en constellation

Le schéma en constellation représente plusieurs relations de faits qui partagent des dimensions communes. Ces différentes relations de faits composent une famille qui partage les dimensions mais où chaque relation de faits a ses propres dimensions [BCA01].

La figure 2.4 montre le schéma en constellation qui est composé de deux relations de faits. La première s'appelle `Ventes` et enregistre les quantités de produits qui ont été vendus dans les différents magasins pendant un certain jour. La deuxième relation gère les différents produits achetés aux fournisseurs pendant un certain temps.

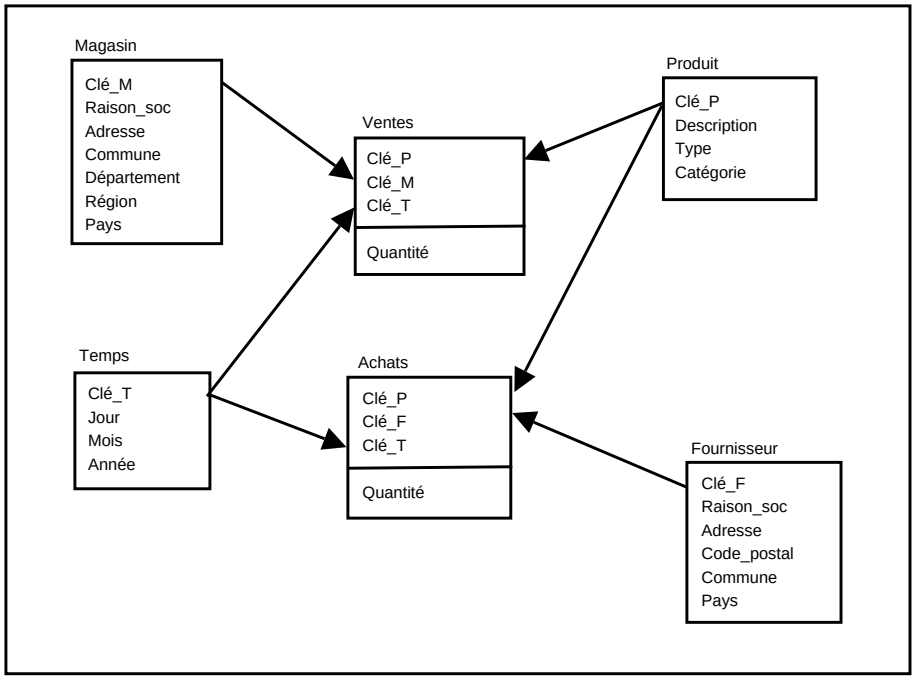

FIG. 2.4 – Exemple de modélisation en constellation.

La relation de faits `Ventes` partage leurs dimensions `Temps` et `Produits` avec la table `Achats`. Néanmoins, la dimension `Magasin` appartient seulement à `Ventes`. Egalement, la dimension `Fournisseur` est liée seulement à la relation `Achats`.

2.2.2 Schéma multidimensionnel (Cube)

Dans le modèle multidimensionnel, le concept central est le **cube**, lequel est constitué des éléments appelés **cellules** qui peuvent contenir une ou plusieurs **mesures**. La localisation de la cellule est faite à travers les **axes**, qui correspondent chacun à une **dimension**. La dimension est composée de **membres** qui représentent les différentes valeurs.

En reprenant une partie du schéma en étoile de la fig. 2.2, nous pouvons construire le schéma multidimensionnel suivant.

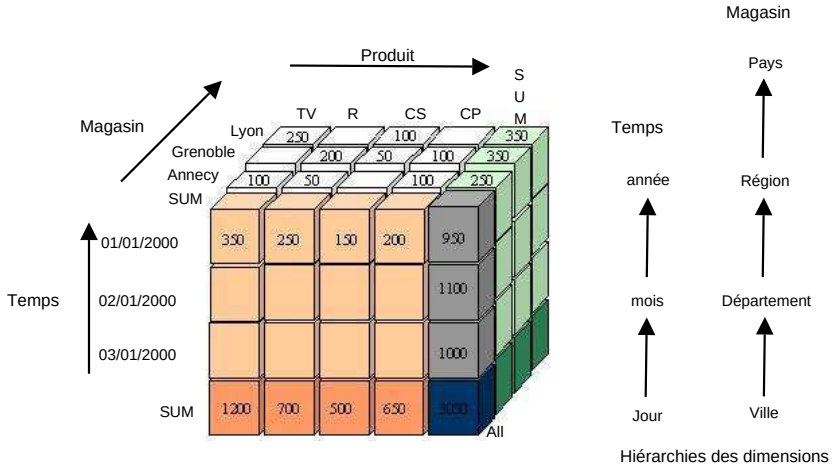

FIG. 2.5 – Exemple de schéma multidimensionnel.

La figure 2.5, présente un schéma multidimensionnel pour les ventes qui ont été réalisées dans les magasins pour les différents produits au cours d'un temps donné (jour). Par exemple, nous avons la quantité de 100 Téléviseurs vendus dans le magasin d'Annecy pendant le 1er janvier 2000.

2.3 Manipulation des données multidimensionnelles

Pour visualiser les données multidimensionnelles, nous pouvons utiliser la représentation sous forme d'une table de données, qui est la plus courante [Mar98, Vas98]. Dans une table, nous représentons les différentes combinaisons des valeurs choisies pour constituer les noms de lignes et de colonnes. Néanmoins, quand le nombre de dimensions est supérieur à deux, l'utilisateur a des problèmes pour visualiser simultanément l'ensemble de l'information. Pour résoudre ce problème, nous devons disposer d'opérations pour manipuler les données et rendre possible la visualisation.

Nous présentons les opérations pour la manipulation des données multidimensionnelles, en les divisant selon leur impact sur la façon de présenter les différentes vues des données analysées [Mar98, MQM97, MS90, Tes00].

2.3.1 Opérations classiques

Ces opérations correspondent aux opérations relationnelles de manipulation des données :

La sélection : Résulte en un sous-ensemble de données qui respecte certains conditions d'appartenance. Nous pouvons avoir une sélection avec des conditions soit sur

les mesures, soit sur les membres. Par exemple, une sélection des ventes supérieur à 350 est une sélection sur les mesures, tandis que une séléction des ventes réalisés dans la région "Rhône Alpes" de l'année "2000" est une sélection sur les membres d'une dimension.

La projection : Résulte en un sous-ensemble des attributs d'une relation, qui sont soit des dimensions, soit des niveaux de granularité. Dans les systèmes décisionnels, les opérations de sélection et de projection sont appelées souvent "slice-and-dice".

La jointure : Permet d'associer les données de relations différentes. Par exemple, en utilisant les tables 2.3 et 2.4, nous faisons une jointure sur la dimension `Produit`. L'objectif est de représenter sur une même table la quantité de produits vendue et leur prix (*cf.* Figure 2.6).

Ventes (01/01/2000)	**Mag1**	**Mag2**	**Mag3**
Téléviseur	100		250
Radio	50	200	
Caméscope		50	100
Camera photo	100	100	

TAB. 2.3 – Ventes par magasin

Produit	**Prix (01/01/2000)**
Téléviseur	1
Radio	2
Caméscope	3
Camera photo	4

TAB. 2.4 – Prix des produits

Les opérations ensemblistes : D'union, d'intersection et de différence sont des opérations qui agissent sur des relations qui ont le même schéma. Par exemple, l'union des tables 2.5 et 2.6 donne comme résultat la table 2.7.

2.3.2 Opérations agissant sur la structure

Les opérations agissant sur la structure visent à présenter une vue (face du cube) différente en fonction de leur analyse, citons :

La rotation (*rotate*) : Consiste à pivoter ou à effectuer une rotation du cube, de manière à présenter une vue différente des données à analyser.

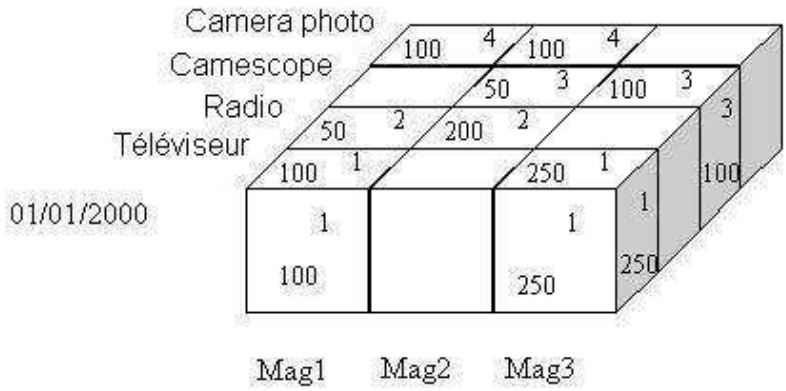

FIG. 2.6 – Exemple de l'opération Jointure.

Ville	Produit	Mag1	Mag2	Mag3
Grenoble	**Téléviseur**	50	100	
Grenoble	**Radio**	50		100
Grenoble	**Caméscope**	100		
Grenoble	**Camera photo**		50	100

TAB. 2.5 – Ventes de produits par ville 1

Ville	Produit	Mag1	Mag2	Mag3
Fontaine	**Téléviseur**	100	100	50
Fontaine	**Radio**	50		100
Fontaine	**Caméscope**	100	50	100
Fontaine	**Camera photo**	100		100

TAB. 2.6 – Ventes de produits par ville 2

Ville	Produit	Mag1	Mag2	Mag3
Grenoble	**Téléviseur**	50	100	
Grenoble	**Radio**	50		100
Grenoble	**Caméscope**	100		
Grenoble	**Camera photo**		50	100
Fontaine	**Téléviseur**	100	100	50
Fontaine	**Radio**	50		100
Fontaine	**Caméscope**	100	50	100
Fontaine	**Camera photo**	100		100

TAB. 2.7 – Ventes du Département Isère

La permutation (*switch*) : Consiste à inverser des membres d'une dimension, de manière à permuter deux tranches du cube.

La division (*split*) : Consiste à présenter chaque tranche du cube en passant d'une représentation tridimensionnelle à une présentation tabulaire. Par exemple, si nous découpons par magasin le cube de ventes de la figure 2.5, nous avons les tables 2.8, 2.9 et 2.10.

Ventes Magasin 1	**01/01/2000**	**02/01/2000**	**03/01/2000**
Téléviseur	100	100	50
Radio	50	200	100
Caméscope			100
Camera photo	100	100	100

TAB. 2.8 – Table de ventes du Magasin 1

Ventes Magasin 2	**01/01/2000**	**02/01/2000**	**03/01/2000**
Téléviseur		100	150
Radio	200	200	100
Camescope	50	100	
Camera photo	100		100

TAB. 2.9 – Table de ventes du Magasin 2

Ventes Magasin 3	**01/01/2000**	**02/01/2000**	**03/01/2000**
Téléviseur	250	100	50
Radio			100
Caméscope	100	50	50
Camera photo		100	100

TAB. 2.10 – Table de ventes du Magasin 3

Nous remarquons que le nombre de tables résultantes de cette opération dépend du nombre de valeurs à l'intérieur de la dimension.

L'emboîtement (*nest*) : Permet d'imbriquer les membres d'une dimension. En utilisant cette opération, nous représentons dans une table bidimensionnelle toutes les données d'un cube quel que soit le nombre de dimensions.

L'enfoncement (*push*) : Consiste à combiner les membres d'une dimension aux mesures du cube et donc de représenter un membre comme une mesure.

L'opération inverse de **retrait (*pull*) :** Permet de changer le statut de certaines mesures, pour transformer une mesure en membre d'une dimension.

La factualisation (*fold*) : Consiste à transformer une dimension en mesure(s) ; cette opération permet de transformer en mesure l'ensemble des paramètres d'une dimension.

La paramétrisation (*unfold*) : Permet de transformer une mesure en paramètre dans une nouvelle dimension.

L'opération Cube : Permet de calculer des sous-totaux et un total final dans le cube (*cf.* Fig 2.5).

2.3.3 Opérations agissant sur la granularité

Les opérations agissant sur la granularité des données analysées, permettent de hiérarchiser la navigation entre les différents niveaux de détail d'une dimension. Dans la suite nous traitons les deux opérations de ce type :

Le forage vers le haut (*drill-up ou roll-up*) : Permet de représenter les données du cube à un niveau plus haut de granularité en respectant la hiérarchie de la dimension. Nous utilisons une fonction d'agrégation (somme, moyenne,...), qui est paramétrée, pour indiquer la façon de calculer les données du niveau supérieur à partir de celles du niveau inférieur.

Le forage vers le bas (*drill-down ou roll-down ou scale-down*) : Consiste à représenter les données du cube à un niveau de granularité inférieur, donc sous une forme plus détaillée.

Ces types d'opérations ont besoin d'informations non représentées dans un cube, pour augmenter ou affiner des données, à partir d'une représentation initiale vers une représentation de granularité différente. Le forage vers le haut a besoin de connaître la fonction d'agrégation utilisée tandis que le forage vers le bas nécessite de connaître les données au niveau inférieur.

2.4 Serveurs OLAP (*On-Line Analytical Processing*)

Les données opérationnelles constituent la source principale d'un système d'information décisionnel. Les systèmes décisionnels complets reposent sur la technologie OLAP, conçue pour répondre aux besoins d'analyse des applications de gestion.

L'acronyme **FASMI** (*Fast Analysis of Shared Multidimensional Information*) permet de résumer la définition des produits OLAP. Cette définition fut utilisée pour

la première fois en 1995 et depuis aucune autre définition n'est plus proche pour résumer le terme OLAP [Ola04].

Fast **:** Le temps de réponse aux demandes des utilisateurs oscille entre 1 et 20 secondes. Les constructeurs utilisent des pré-calculs pour réduire les durées des requêtes.

Analysis **:** Le système doit pouvoir faire face à toutes les logiques d'affaires et de statistiques, ainsi que fournir la possibilité aux utilisateurs de construire leurs calculs et leurs analyses sans avoir à programmer. Pour cela, il y a des outils qui seront fournis par le constructeur.

Shared **:** Le système doit créer un contexte où la confidentialité est préservée et doit gérer les cas où plusieurs utilisateurs ont des droits en écritures. Ce point constitue la plus grosse faiblesse des produits actuels.

Multidimensional **:** C'est la caractéristique clé. Le système doit fournir des vues conceptuelles multidimensionnelles des données. Il doit supporter aussi les hiérarchies.

Informations **:** L'ensemble des données et les informations nécessaires pour un produit OLAP.

Nous exposons dans la suite les divers types de stockage des informations dans les systèmes décisionnels.

2.4.1 ROLAP (*Relational OLAP*)

Dans les systèmes relationnels OLAP, l'entrepôt de données utilise une base de données relationnelle. Le stockage et la gestion de données sont relationnels. Toutefois, le modèle relationnel requiert des extensions pour supporter les requêtes d'analyses multidimensionnelles du niveau d'application. Le moteur ROLAP traduit dynamiquement le modèle logique de données multidimensionnel M en modèle de stockage relationnel R (la plupart des outils requièrent que la donnée soit structurée en utilisant un schéma en étoile ou un schéma en flocon de neige). Techniquement, le moteur ROLAP exécute une transformation à partir d'une requête multidimensionnelle m contre M vers une requête relationnelle r contre R. L'efficacité du résultat de la requête est le facteur dominant pour la performance et le passage à l'échelle global du système. Ainsi, les stratégies d'optimisation représentent le point principal qui distingue les systèmes ROLAP existants [DSBH99, Sat03].

La figure 2.7 montre une architecture pour le serveur ROLAP.

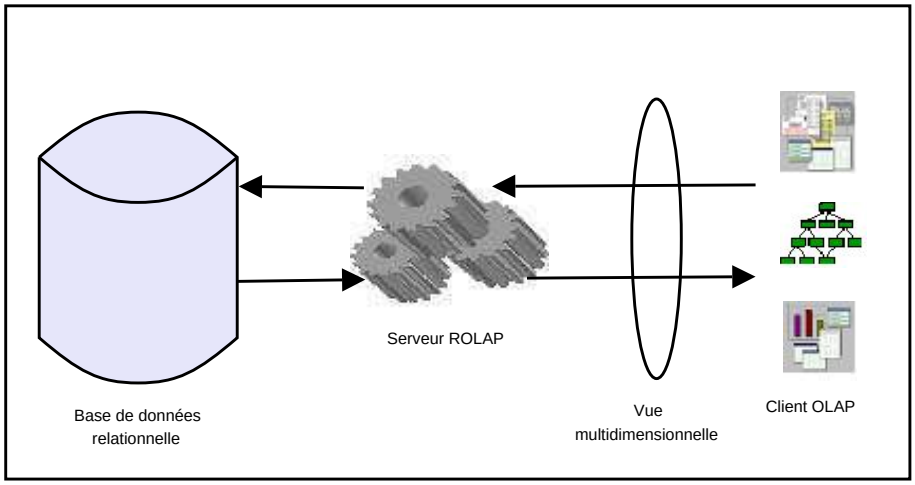

FIG. 2.7 – Architecture ROLAP.

La technologie ROLAP a deux avantages principaux : (1) elle permet la définition de données complexes et multidimensionnelles en utilisant un modèle relativement simple , et (2) elle réduit le nombre de jointures à réaliser dans l'exécution d'une requête. Le désavantage est que le langage de requêtes tel qu'il existe, n'est pas assez puisant ou n'est pas assez flexible pour supporter de vraies capacités d'OLAP [VS99, BSH98].

Nous décrivons quelques produits commerciaux existants [How03, CHJM02] :

IBM DB2 UDB : C'est un SGBD tournant sur une variété de plate-formes. IBM est une *shared nothing database* et elle supporte le concept de fédération. Elle dispose d'une large gamme d'outils, par exemple :

DB2 Performance Expert **:** Cet outil pour multiplate-formes permet la création des rapports, des analyses et recommande des changements par rapport à la performance.

DB2 DataJoiner **:** Outil pour l'optimisation des requêtes SQL.

DB2 Integrated Cluster Environnement **:** Il permet le passage à l'échelle.

Oracle 9*i* : C'est un SGBD tournant aussi sur une variété de plate-formes. Oracle supporte des partitions de *hash, range* et *list*.

Oracle 9i : C'est une *shared disk database* et elle supporte la consolidation (focalisée sur une base de données centralisée).

Real Application Clusters : Il permet de désigner certains processeurs comme processeurs OLAP et d'autres comme processeurs de requêtes.

L'optimiser : Basé sur les coûts ou sur les règles.

SQL Server 2000 : C'est une *shared nothing database* et elle supporte le concept de fédération (liaison entre bases de données distribuées et hétérogènes via le logiciel).

L'optimiser de SQL Server : Basé sur les coûts avec création automatique de statistiques et leur rafraîchissement.

Le Query Processor : Il supporte des requêtes multidimensionnelles, ainsi que les index composites et semi-jointures.

Le SQL Query Analyzer : Il peut faire des suggestions par rapport à l'implantation des index additionnels et des statistiques complémentaires.

Microsoft DTS (Data Transformation Services) : L'outil ETL intégré dans Microsoft SQL Server.

2.4.2 MOLAP (*Multidimensional OLAP*)

Les systèmes multidimensionnels OLAP utilisent une base de données multidimensionnelle pour stocker les données de l'entrepôt et les applications analytiques sont construites directement sur elle. Dans cette architecture, le système de base de données multidimensionnel sert tant au niveau de stockage qu'au niveau de gestion des données. Les données des sources sont conformes au modèle multidimensionnel, et dans toutes les dimensions, les différentes agrégations sont précalculées pour des raisons de performance [WB97].

La figure 2.8 montre une architecture pour les systèmes MOLAP.

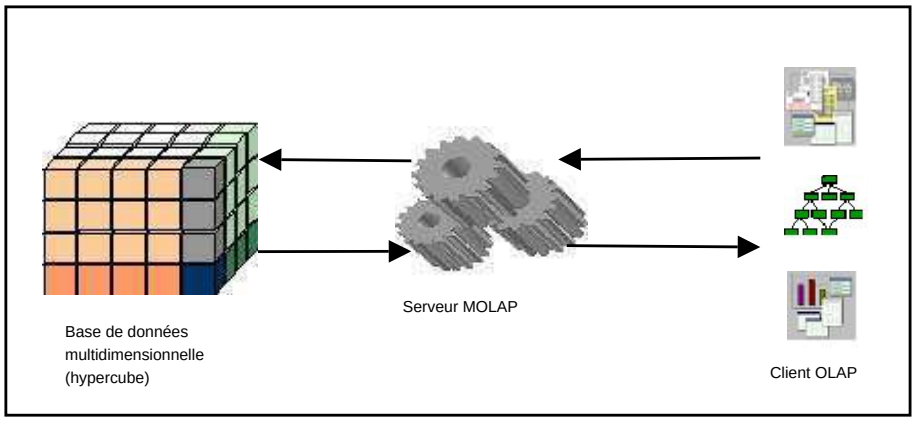

FIG. 2.8 – Architecture MOLAP.

Les systèmes MOLAP doivent gérer le problème de données clairsemées, quand seulement un nombre réduit de cellules d'un cube contiennent une valeur de mesure associée. Par exemple, dans le cube Ventes (*cf.* Fig 2.5), il peut arriver que certains produits ne se soient pas vendus dans une ville spécifique et qu'en conséquent il n'y ait pas de valeurs de mesure pour cette combinaison. La façon de résoudre cette problématique, par les produits commerciaux, représente un des plus importants critères pour les systèmes MOLAP. La plupart des systèmes existants utilisent l'approche du vecteur multidimensionnel pour le stockage de données, ainsi que la compression de l'espace de stockage pour les vecteurs de plus de trois dimensions [Ben02, DSBH99].

Les avantages des systèmes MOLAP sont basés sur les désavantages des systèmes ROLAP et elles représentent la raison de leur création. D'un côté, les requêtes MOLAP sont très puissantes et flexibles en termes du processus OLAP, tandis que, d'un autre côté, le modèle physique correspond plus étroitement au modèle multidimensionnel. Néanmoins, il existe des désavantages au modèle physique MOLAP. Le plus important, à notre avis, c'est qu'il n'existe pas de standard du modèle physique.

Les produits commerciaux existants sont :

Hyperion Essbase OLAP Server : C'est une base de données multi-utilisateurs, multi-thread et multidimensionnelle [Blo99]. Les composants d'Essbase OLAP sont :

Hyperion Essbase Application Manager : Il fournit des outils graphiques. Il inclut des modules pour la construction et le chargement des structures OLAP, pour le chargement des données, pour la définition des processus de calcul, pour la gestion des partitions de la base de données,...

Hyperion Essbase Query Designer : Ce composant remplace le *Query Wizard* d'Essbase 6.0. Il a été conçu pour faciliter la navigation des utilisateurs finaux.

Hyperion Essbase Partition Option : Il permet aux développeurs des applications la création logique et physique de sous-ensembles des bases de données. Les deux types de partitions qui sont supportés sont : transparente et liée. Le premier cherche à montrer à l'utilisateur final une base de données centralisée. Avec le deuxième type, nous pouvons définir deux partitions liées si elles partagent au moins une partie d'une dimension. Ainsi, nous pouvons naviguer entre applications et non entre partitions. Par exemple, nous pouvons naviguer entre l'application de ventes et celle d'achats, car elles relient le même ensemble de produits.

SAS OLAP Server : C'est une base multidimensionnelle qui fournit un accès rapide aux données agrégées [SAS04]. Leurs composants incluent :

SAS OLAP Cube Studio : C'est un outil pour la construction des cubes et qui peut être facilement utilisé pour la définition des mesures, des dimensions et des agrégations.

SAS Metadata Server : C'est un conteneur des métadonnées.

Ce serveur fournit un processus **ETL** (*Extract, Transform and Load*) qui est transparent et intégré.

Informix MetaCube : Il fournit un mécanisme pour analyser l'entrepôt de données et leurs *data marts* intégrant des outils OLAP avec la technologie de bases de données relationnelles [Inf02]. Les composants du logiciel MetaCube sont :

MetaCube Analysis Engine : Il fournit une interface multidimensionnel. Cet outil étend la fonctionnalité d'analyse de la base de données avec l'incorporation des opérations OLAP, par exemple l'opération rotation (*rotate*).

MetaCube Explorer : C'est une interface graphique qui permet aux utilisateurs d'exécuter des requêtes au travers du *MetaCube Analysis Engine*.

MetaCube Warehouse Manager : C'est une interface graphique pour créer une description multidimensionnelle des tables et des colonnes dans l'entrepôt de données. Le *MetaCube Analysis Engine* stocke cette description multidimensionnelle comme un ensemble de tables.

2.4.3 HOLAP (*Hybrid OLAP*)

Un système HOLAP est un système qui supporte et intègre un stockage des données multidimensionnel et relationnel d'une manière équivalente pour profiter des caractéristiques de correspondance et des techniques d'optimisation.

Ci-dessous, nous traitons une liste des caractéristiques principales qu'un système HOLAP doit fournir [DSBH99] :

La transparence du système : Pour la localisation et l'accès aux données, sans connaître si elles sont stockées dans un SGBD relationnel ou dimensionnel. Pour la transparence de la fragmentation,...

Un modèle de données général et un schéma multidimensionnel global : Pour aboutir à la transparence du premier point, tant le modèle de données général que le langage de requête uniforme doivent être fournis. Etant donné qu'il n'existe pas un modèle standard, cette condition est difficile à réaliser.

Une allocation optimale dans le système de stockage : Le système HOLAP

doit bénéficier des stratégies d'allocation qui existent dans les systèmes distribués tels que : le profil de requêtes, le temps d'accès, l'équilibrage de chargement,...

Une re-allocation automatique : Toutes les caractéristiques traitées ci-dessus changent dans le temps. Ces changements peuvent provoquer la réorganisation de la distribution des données dans le système de stockage multidimensionnel et relationnel, pour assurer des performances optimales.

Actuellement, la plupart des systèmes commerciaux utilisent une approche hybride. Cette approche permet de manipuler des informations de l'entrepôt de données avec un moteur ROLAP, tandis que pour la gestion des *data marts*, ils utilisent l'approche multidimensionnelle. Dans la figure 2.9, nous montrons une architecture en utilisant les types de serveurs ROLAP et MOLAP pour le stockage de données.

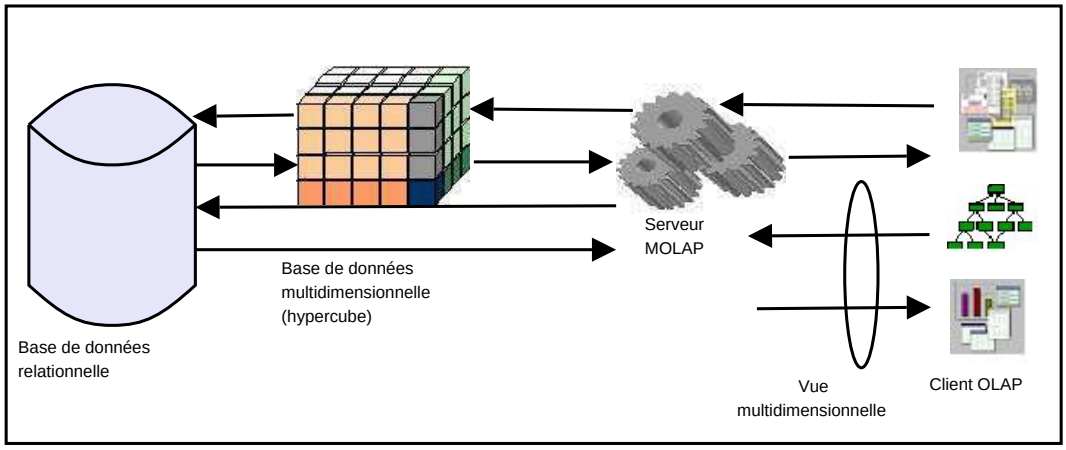

FIG. 2.9 – Architecture HOLAP.

Les produits commerciaux existants sont :

DB2 OLAP Server : Il permet de calculer, de consolider et d'accéser l'information à partir des bases de données multidimensionnelles, relationnelles ou les deux [DB204]. Certains de ses composants sont :

DB2 OLAP Integration Server : Il utilise des outils graphiques et des services pour l'intégration des données qui réduisent le coût pour créer, déployer et gérer les applications de DB2 OLAP Server.

DB2 OLAP Server Administration Services : Il fournit des outils pour améliorer et faciliter des tâches d'administration.

Oracle Express Server : Ce serveur exploite un modèle de données multidimensionnel. Il gère un ensemble d'indicateurs à n dimensions, dont les valeurs sont

stockées ou calculées dynamiquement [ORA96].

Le stockage des données peut se faire dans une base de données multidimensionnelle ou relationnelle.

La base *Oracle Express Server :* Stocke les agrégats multidimensionnels, tandis que les données de détail sont stockées dans la base relationnelle.

En utilisant un **4GL (Langage de 4ème génération)**, *Express Server* propose des fonctions avancées pour la présentation et l'analyse des résultats, tels que : traitement de séries chronologiques, consolidation, statistiques, prévisions,...

2.5 Les projets de recherche

Dans cette partie, nous décrivons les projets de recherche sur les entrepôts de données. Les deux premiers (WHIPS et SIRIUS) sont centrés sur des problèmatiques liées à l'extraction et à la maintenance des données. Le troisième DWQ traite de la qualité des entrepôts, tandis que le dernier, le projet EVOLUTION propose le développement d'une méthodologie et d'outils pour l'aide à la conception et l'évolution des entrepôts.

2.5.1 **WHIPS** *Warehouse Information Prototype at Stanford*

L'objectif du projet est de développer des algorithmes pour collecter, intégrer et maintenir des informations de sources hétérogènes, distribuées et autonomes. Le modèle relationnel est utilisé comme modèle unificateur.

Pour l'initialisation de l'entrepôt, l'intégrateur crée un gestionnaire de vues pour chaque vue dans l'entrepôt. Le gestionnaire de vues envoie les requêtes issues de la définition d'une vue au processeur de requêtes. Le processeur de requête contacte les traducteurs de la source d'information adéquate pour obtenir les résultats, les assemble et passe la réponse au gestionnaire de vue. Le gestionnaire de vue envoie la réponse à l'adaptateur de l'entrepôt pour initialiser la vue.

Chaque source contienne un moniteur et un adaptateur. L'adaptateur transforme les données de la source en une représentation relationnelle. Le moniteur détecte les changements qui se produisent au niveau des sources. Les mises à jour sont transmises à l'intégrateur, qui les transmet aux gestionnaires de vues qui sont concernés par les modifications [LZW+97, HGMW+95].

Description des composants :

Adaptateur/Moniteur : Il est associé à chaque source de données et il a deux fonctionnes : transformer le schéma et ses instances en une représentation intermé-

diaire et détecter automatiquement les changements dans la source pour les propager vers l'intégrateur.

Intégrateur : Il est responsable de la réception des représentations intermédiaires des données sources, en provenance des adaptateurs, et de l'intégration de celles-ci. L'intégration entraîne aussi une phase de transformation, celle-ci consiste à les préparer (mise en correspondance des formats de données), les nettoyer, les filtrer,..., pour les rendre conforme au schéma de l'entrepôt et aux critères de qualité choisis.

2.5.2 SIRIUS *Supporting the Incremental Refreshment of Information warehouses*

Ce projet développé à l'Université de Zurich, est un système d'entrepôt de données qui a pour objectif d'étudier des techniques permettant le rafraîchissement incrémental de l'entrepôt en réduisant les temps de mise à jour. Le schéma de l'entrepôt est défini sous la forme d'un *schéma global UML*.

Chaque source de données est munie d'un adaptateur et d'un moniteur. Le moniteur détecte les évolutions de la source à laquelle il est associé. Le module de gestion du rafraîchissement est le module central (DWRM *Data Warehouse Refresh Manager*). L'adaptateur traduit les données de la source dans un modèle commun, interne au système et les transmet au module central DWRM [VGD00].

Objectif du projet :

L'objectif principal de cette approche est d'introduire une architecture d'intergiciel flexible, pour :

- Fournir des solutions pour le rafraîchissement de données.

- Pouvoir être utilisée de façon indépendante par rapport au stockage de données de l'entrepôt.

- Pouvoir être utilisée par des entrepôts de données composés des sources de données hétérogènes.

2.5.3 DWQ *Foundations of Data Warehouse Quality*

DWQ est un projet de la communauté Européenne (France, Allemagne, Italie et Grèce) pour le développement des fondements sémantiques qui permettront d'aider les concepteurs d'entrepôts de données dans le choix des modèles, des structures de données avancées et des techniques d'implantation efficaces en s'appuyant sur des facteurs de qualité de service.

Les résultats comportent des méta-modèles formels de données destinés à la description de l'architecture statique d'un entrepôt de données. Les outils associés comportent des facilités de modélisation incluant des caractéristiques spécifiques aux entrepôts comme la résolution de sources multiples, la gestion de données multidimensionnelles agrégées et des techniques pour l'optimisation de requêtes et la propagation incrémentale des mises à jour.

Ce projet permet aussi l'évolution de l'entrepôt. Les outils incluent un ensemble d'opérateurs, l'analyse et l'optimisation des définitions des vues, la sélection optimisée de sources de données, des stratégies d'intégration et des vues matérialisées [JV97, JQJ98, JJQV99, TS99].

Objectifs du projet :

Les objectifs de recherche visent trois domaines, où les facteurs de qualité représentent le point central :

- Enrichir la sémantique de la méta base avec des models formels de qualité de l'information qui permettent l'optimisation de la conception de façon adaptative et quantitative.

- Enrichir la sémantique des models d'information qui permettent aussi bien le rafraîchissement incrémental de données et leur propagation vers l'entrepôt, que la résolution des conflits.

- Enrichir la sémantique des models de schéma de l'entrepôt qui permettent aux concepteurs de prendre les avantages explicites par rapport à la nature temporelle, spatiale et des agrégats des données.

Composants de l'entrepôt de données :

Le projet DWQ fournit un modèle architectural qui considère la conception, la mise en oeuvre, la maintenance et l'évolution des entrepôts. Les composants basics sont :

- Sources : les données stockées dont le contenu peut être matérialisé dans l'entrepôt.

- Adaptateur : responsable du chargement des données sources dans l'entrepôt.

- Base de données finale : l'entrepôt de données et les *data marts*.

- Méta base : conteneur d'information des autres composants, par exemple, le schéma des données sources.

- Agents d'administration : concepteurs de l'entrepôt.

- Clients : utilisateurs de l'information.

2.5.4 Projet EVOLUTION

Le projet propose le développement d'une méthodologie et d'un atelier d'outils de type CASE pour l'aide à la conception et l'évolution des entrepôts de données. L'objectif de ce projet est la spécification d'un ensemble d'outils d'aide à la conception de l'entrepôt et à sa maintenance. Deux objectifs sont intégrés dans la création de ces outils : prévoir l'évolutivité de schéma et gérer la traçabilité des évolutions successives.

Pour atteindre ce but, trois objectifs, correspondant aux problèmes de recherches encore non résolus sont à atteindre :

- La définition d'un formalisme de modélisation des données et des métadonnées de l'entrepôt, ce formalisme doit permettre à la fois de le décrire et de le manipuler (pour le faire évoluer ou pour vérifier ses propriétés).

- Des algorithmes de traitement sémantique de la triple hétérogénéité de conceptualisation, de temps et de granularité.

- Des algorithmes d'optimisation des performances des ressources matérielles et logicielles de l'entrepôt pour fournir efficacement les outils de *Data Mining*.

L'approche envisagée propose : l'intégration sémantique des différents schémas sources, la constitution d'un schéma de l'entrepôt, la définition et la mise à jour des vues, la prise en compte de l'imprécision (des données mal connues ou incertaines dans l'entrepôt), la prise en compte du temps et de la granularité, la vérification de la cohérence et de la consistance et l'optimisation des ressources (parallélisme) [Evo97].

2.6 Aspects temporels des entrepôts

Malgré une conception attentive et soigneuse, la structure d'une base de données est sujette à de nombreux changements. Bien que les estimations diffèrent, la plupart s'accordent sur le fait que plus de 50% du travail des développeurs se focalise sur les modifications du système après leur implantation [Rod95]. Pour résoudre le problème de perte de données après des changements du schéma, le concept d'évolution de schéma à été introduit pour récupérer les données existantes. Cette récupération se réalise par le biais de l'adaptation des données au nouveau schéma. Néanmoins, dans les systèmes qui doivent gérer des données historiques, l'évolution

de schéma n'est pas suffisante et la maintenance de plusieurs schémas est requise [GM02]. Cette problématique a donnée naissance à la notion de versions de schémas.

Nous traitons les différentes nuances entre les définitions de modification, d'évolution et de version de schéma. Nous commençons avec l'état actuel des versions dans les différents modèles existants. Ensuite nous traitons le sujet des versions de schémas, la gestion des données intensionnelles et extensionnelles, les changements et leur représentation. Pour finir, la dernière section donne une brève description des opérations primitives.

2.6.1 Etat courant des versions

Nous étudions ce paradigme dans les divers modèles de données existants. Ceux que nous abordons sont le modèle relationnel, le modèle à objets, les bases de données temporelles et nous terminons avec les entrepôts de données.

2.6.1.1 Modèle relationnel

La gestion des évolutions dans la structure des bases de données a donné naissance aux concepts de modification, d'évolution et de versions de schémas. La distinction entre les différents concepts reste un peu imprécise. Dans la suite nous donnons chaque définition [DGW+98, Rod92, Rod95, TS93] :

La modification de schéma d'un système de bases de données peut se faire même si la base de données est déjà peuplée.

L'évolution de schéma d'un système de bases de données facilite la modification du schéma de la base sans perdre les données existantes.

Les versions de schéma permettent l'accès à toutes les données, de façon rétrospective et prospective, à travers des interfaces de versions définies par l'utilisateur.

A partir de ces définitions, nous constatons que même si l'évolution de schéma permet leur actualisation, elle n'implique pas un support historique complète du schéma.

Versions et bases de données : Dans la suite nous traitons les conditions pour qu'une base de données supporte la gestion de versions [ABCM99, CW98, KU95, Mun93].

La fonctionnalité des systèmes qui supportent les versions est représentée par l'ensemble des opérations à utiliser. Les opérations fondamentales sont la création de nouvelles versions et l'accès aux données des différentes versions. Néanmoins, le système doit fournir d'autres opérations sur les versions : définition des noms ou des identifications, effacement, modification, transformer une version dans une version

immuable[1] , combinaison de versions, etc.

La transparence des versions représente la capacité d'accéder aux données sans avoir la nécessité de manipuler le système de versions. Il existe trois manières d'aborder les relations entre le modèle de données et le modèle de versions. La première consiste à placer le modèle de version **au-dessus** du modèle de données. De cette façon, le modèle de version est représenté par un schéma dont le modèle de données sous-jacent est indépendant du modèle de versions. L'avantage de cette solution est la simplicité du modèle de données. Par contre, cette approche ne supporte pas le stockage des versions de façon efficace. De plus, la gestion des transactions du SGBD ne considère pas le système de versions. PCTE (*Version Management in the PACT Integrated Software Engineering*) et CoMa [CW98, EJWG03] sont deux exemples de systèmes reposant sur cette approche.

La deuxième solution construit un modèle de version **à l'intérieur** du modèle de données. Cette approche requiert l'extension du modèle de données pour supporter les versions, ainsi que la modification du langage de requêtes pour fournir un mécanisme d'écriture de requêtes à une base de données supportant des versions. Ce processus d'extension du modèle complique la structure des données ainsi que la formulation des requêtes. De plus, le modèle de version dépend fortement du modèle de données. Quelques systèmes qui suivent cette voie sont DAMOKLES [CW98, EJWG03] et Adele [Mun93, EJWG03].

La dernière approche représente le modèle de version **en-dessous** du modèle de données. Dans cette alternative, le modèle de version est complètement orthogonal au modèle de données. Cette approche peut être atteinte au travers d'un moteur de version qui fournit un stockage des deltas[2] et des règles de configuration utilisées dans la formulation des modèles de version spécifiques. Le moteur de version est indépendant du modèle de données et peut être associé à n'importe quel modèle de données. Les systèmes qui utilisent cette approche sont par exemple ICE et COV [CW98].

Pour finir, dans [CGJM01], les auteurs utilisent les versions d'entités persistantes et le partage entre versions des parties ayant même valeur (qu'ils appellent *versions locales*) par plusieurs versions (*versions globales*), pour présenter deux approches : ascendante et descendante. La première utilise l'identification ascendante des versions locales, le partage incassable (qui permet seulement l'insertion de nouvelles versions et la lecture des versions stockées) et le stockage par entité. L'approche descendante utilise l'identification de versions globales, le partage cassable (qui permet en plus les mises à jour des versions) et aussi le stockage par entité. Ils présentent aussi les conclusions de chaque approche.

[1]Une version immuable ne permet pas de changements.
[2]Les deltas représentent les différences entre versions.

2.6.1.2 Modèle à objets

Dans ce paradigme, deux types de changements sont significatifs : les changements dans la définition d'une classe (le contenu d'un noeud) et les changements dans la structure (les arêtes et les noeuds). Ces changements représentent l'ensemble des opérations primitives qui permettent de manipuler l'évolution de la classe. Les changements dans la définition de la classe incluent les opérations de création et de suppression des attributs et des méthodes qui appartiennent à une classe. Les changements dans la structure, correspondent aux opérations de création et de suppression d'une classe, ainsi que la modification des relations entre les classes [MBJ+02, GM99, GMS00, FGM00, Lau96, RR97].

Versions de schémas, de classes et de vues : Dans l'environnement de bases de données multi-utilisateurs, une fonctionnalité importante est la capacité des utilisateurs à visualiser et manipuler différentes collections d'objets via différentes versions de schéma. Il existe trois approches pour représenter les évolutions : la version de schéma, la version de classes et les schémas de vues [Lau96, KC88, FGM00, MS93, Odb92].

Dans l'approche de **versions de schéma**, chaque version représente une configuration cohérente de versions au niveau de la classe. Le système fournit un ensemble d'opérations primitives pour la gestion de l'évolution. Quand nous dérivons une nouvelle version de schéma SV_j à partir de SV_i, les classes existantes peuvent être modifiées ou effacées et des classes nouvelles peuvent être crées pour SV_j. Si une classe C est modifiée, on peut dire que SV_i et SV_j contiennent différentes versions de la class C, dénotée par $SV_i.C$ et $SV_j.C$ respectivement. Toutes les versions d'un schéma sont stockées dans une structure qui garde les relations de dérivation entre toutes les versions de schémas.

La deuxième approche traite l'évolution au **niveau de la classe**. A la différence des versions de schéma, si une version nouvelle V_j de la classe C est dérivée de la version V_i, toutes les sous-classes de la classe C doivent faire référence à la nouvelle version de C. Ceci signifie qu'une nouvelle version doit être créée pour chacune des sous-classes de C.

Le **schéma de vues** peut être utilisé pour simuler des modifications de schéma. L'idée générale est de manipuler la base de données seulement à travers des vues. Dans ce cas, les changements peuvent être simulés en changeant les vues et les applications peuvent utiliser différentes vues (comme différents schémas) sans changer la base de données sous-jacente.

De nombreux travaux [MBJ+02, GM99, GMS00, KC88, Lau96] se sont focalisés sur la première approche, car elle manipule la granularité de versions au niveau du schéma. Ceci permet une vue globale de l'ensemble des classes à l'intérieur d'une version de schéma.

2.6.1.3 Bases de données temporelles

Les bases de données temporelles permettent de représenter et de manipuler l'histoire des données (données extensionnelles). Le support de versions de schémas dans les bases de données temporelles présente deux problèmes fondamentaux. Le premier est le stockage et la manipulation de l'ensemble de versions de schéma appelée la sémantique du changement. Ce problème implique la vérification et la maintenance de la consistance du schéma après des changements. Le second problème représente la gestion et la manipulation des versions de données, nommée aussi la propagation des changements. Cette problématique concerne la consistance des données existantes avec la nouvelle version du schéma [MBJ+02, GM02, GMS00, ME97, RGMS01].

L'aspect temporel est associé aux objets, aux versions, aux attributs et aux relations. Dans les bases de donnés temporelles, le temps de transaction correspond à l'instant où un fait est enregistré dans la base. Alors que, le temps de validité d'un fait correspond aux instants auxquels ce fait est vrai dans le monde modélisé. Dans la suite, nous traitons chaque définition.

Temps de transaction, temps de validité et bitemporel : La manipulation de la dimension temps dans les versions de schéma correspond soit à la version de schéma au temps de transaction, soit au temps de validité, soit les deux. Cette dernière appelée bitemporel, représente la combinaison des deux premières [Dum00, GM02, ME97, Sno99].

Pour le **temps de transaction**, les modifications appliquées dans une version de schéma sont effectives au moment de leur définition. Dans cette approche, la gestion du temps est transparente à l'utilisateur. Le schéma courant peut être modifié et les changements sont appliqués sans aucune référence au temps. Cette approche ne permet aucun changement sur les versions successives du schéma. Néanmoins, toutes les versions sont disponibles pour l'accès et la manipulation des données.

La version de schéma en **temps de validité** est nécessaire quand les modifications des versions de schémas rétroactives ou proactives doivent être supportées. Dans ce cas, à chaque version de schéma est assignée une validité temporelle. Cette approche permet des versions de schémas multiples valides à différents moments. Toutes les versions de schémas sont disponibles aussi bien pour l'accès et la manipulation des données, que pour la réalisation des modifications futures.

La version de schéma **bitemporelle** représente une combinaison du temps de transaction et du temps valide. Elle supporte les modifications des versions de schémas courantes, comme dans la première approche et aussi les modifications des versions de schémas de façon rétroactive et proactive. Par rapport à la version de schéma en temps valide, l'histoire complète des changements du schéma est maintenue, car aucune version de schéma n'est abandonnée.

Le tableau 2.11 décrit un exemple de relation bitemporelle en TSQL2 [Sno95].

Nom	**Salaire**	**Temps de transaction**	**Temps de validité**
SERNA	1000	1-07-2000 - 31-12-2000	1-07-2000 - 31-12-2001
SERNA	1500	1-01-2002 - 31-01-2002	1-01-2002 - 31-12-2002
SERNA	1600	1-01-2003 - UC[3]	01-01-2003 - 31-12-2004

TAB. 2.11 – Exemple de relation bitemporelle

2.6.1.4 Le temps et les entrepôts de données

Les actualisations réalisées dans les sources des données doivent être reportées sur l'entrepôt. Ce processus requiert une transaction de maintenance. Dans les entrepôts, une problématique très importante est la façon d'exécuter les transactions de maintenance sans perturber les requêtes des utilisateurs. La plupart des systèmes commerciaux garantissent la consistance sans blocage, en exécutant une seule et unique transaction de maintenance, normalement pendant la nuit, quand l'entrepôt est hors de service [QW97].

Les changements effectués dans les sources peuvent être appliqués aussi bien au contenu des données qu'à leur structure. Dans [Ben02], l'auteur propose une infrastructure adaptable pour l'évolution des entrepôts de données. Il présente un ensemble d'opérateurs d'évolution de schémas multidimensionnels, ainsi qu'un langage de description de données multidimensionnelles. Néanmoins, aucun support historique n'est présenté dans ce travail.

En ce qui concerne l'approche des versions de schémas, la plupart des recherches se sont focalisées dans la propagation des changements sur les données [KC02, KM99, QW97, TU98]. Dans [KC02], par exemple, les auteurs proposent un contrôle de concurrence multi-version adapté pour les entrepôts de données (MVCC-DW) en utilisant un serveur multidimensionnel OLAP (MOLAP). MVCC-DW permet des actualisations de l'entrepôt pendant que des requêtes sont exécutées sur les données.

Dans [QW97], les auteurs proposent un mécanisme appelé 2VNL (Two-Version No-Locking). Ce mécanisme permet qu'une requête puisse continuer à utiliser une vue, alors qu'une transaction de maintenance écrit la nouvelle version de cette vue.

Dans [ZR98, RKZ+99], les auteurs proposent une approche pour la maintenance de la consistance dans un environnement concurrent supportant des changements sur les données et sur leur schéma. Ils ont développé un système appelé EVE (Evolvable

[3]Le symbole UC (*Until Changed*) indique que l'intervalle de transaction de la version va jusqu'à la date d'aujourd'hui.

View Environment), qui utilise un langage nommé Evolvable-SQL pour le support du processus d'évolution des vues, et un processus de réécriture de vues pour leur adaptation quand le schéma évolue.

Cependant, dans l'environnement des entrepôts nous considérons comme essentielles la gestion et la manipulation des données historiques, aussi bien que des données courantes. L'approche des versions de schéma a été utilisée principalement pour la cohérence des données et de leurs schémas courants. Nous proposons l'utilisation des versions de schéma, comme une alternative pour la gestion et la manipulation des données courantes et historiques.

2.6.2 Espace de versions

Dans cette partie, nous traitons de l'espace de versions. D'abord, nous raffinons la définition de version de schéma. Ensuite, nous expliquons leurs types ainsi que les graphes de version, puis, nous terminons avec une description de l'ensemble des opérations à utiliser lors des changements à effectuer.

Versions de schémas : Il est important dans la gestion des versions de distinguer la récupération et la mise à jour des données. Pour répondre à ce besoin, le concept de versions de schéma a été divisé en version partielle et complète [CW98, Rod95, WMC01] :

Version partielle de schéma : Les données stockées sous n'importe quel schéma historique peuvent être visualisées sous tout autre schéma. Néanmoins les données peuvent seulement être actualisées au travers du schéma courant.

Version complète de schéma : Les données stockées sous n'importe quel schéma historique peuvent être visualisées et actualisées sous tout autre schéma.

Un modèle de version définit les éléments qui ont évolués, les propriétés communes partagées et les deltas. Chaque version doit être identifiée de façon unique, à travers un identificateur de version VID. Les deltas représentent les différences entre deux versions. Un delta peut être symétrique ou direct.

Un **delta symétrique** entre deux versions v_1 et v_2, représente l'ensemble de différences entre elles, c'est à dire :

$\Delta(v_1, v_2) = (v_1 - v_2) \cup (v_2 - v_1)$

Où le symbole Δ correspond à la définition du delta.

Un **delta direct** représente une séquence d'opérations de changements op_1, ... ,op_m. Quand ces changements sont appliqués sur une version v_1, ils produisent une

nouvelle version v_2.

$\Delta\ (v_1,v_2) = op_1, \ldots, op_m$

C'est-à-dire, v_1 génère v_2 : $v_1 \longrightarrow v_2$

2.6.3 Versions basées sur l'état et sur les changements

Une version est définie comme un état d'un objet variant dans le temps. Les modèles de versions peuvent considérer les différents états des objets. Dans cette approche, les versions sont décrites par rapport aux révisions et aux variantes. Une révision est une version qui remplace la précédente. L'ensemble des versions qui coexistent sont appelées des variantes.

Dans les modèles basés sur les changements, une version est décrite par rapport aux modifications. Dans cette approche, à chaque changement est affecté un identificateur (CID) ainsi que des attributs pour caractériser les causes et/ou la nature du changement. Ainsi, une version est décrite par rapport aux changements qu'elle implante.

L'espace des changements peut se représenter de deux façons (Figure 2.10).

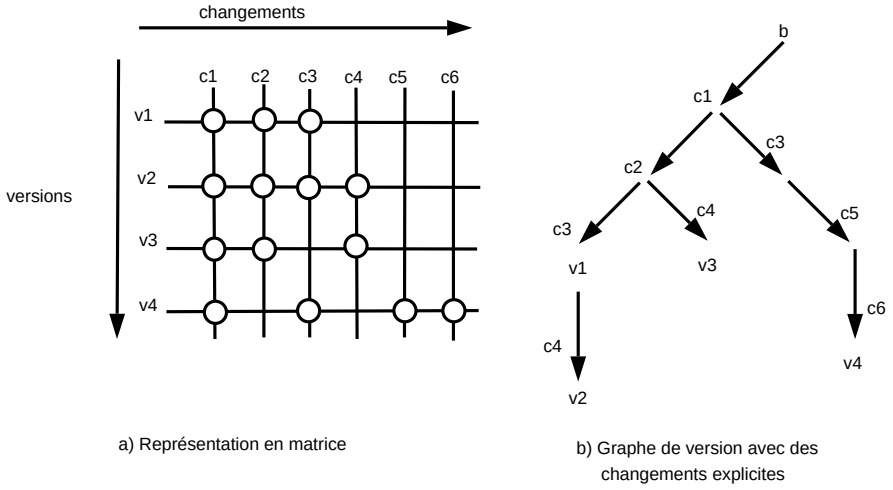

FIG. 2.10 – Espace des changements

La partie a) de la figure montre la représentation en matrice utilisée dans le système Aide de camp [CW98, Mun93]. Les lignes et colonnes correspondent à des versions et à des changements. Par exemple, l'ensemble des changements c_1, c_2, c_3 et c_4 construisent la version v_2. La partie b) représente un graphe de versions où b dénote la racine à partir de laquelle tous les changements d'une version sont décrits de manière explicite. Au contraire, dans l'approche basée sur l'état, les changements

sont anonymes de façon à ce que nous puissions les déduire à partir de la topologie du graphe de version.

2.6.4 Graphes de version

Un graphe de version consiste en un ensemble de noeuds et d'arêtes qui correspondent à une version et à leur relations [Mun93, CW98]. La plupart des systèmes utilisent une organisation à un ou deux niveaux que nous exposons dans la suite.

Types d'organisations Un graphe de version d'un niveau consiste en un ensemble de versions qui sont connectées par leurs relations successeurs. Ce type de graphes représente l'historique de l'évolution d'un élément. Par exemple, si la version v_2 est un successeur de v_1, cela signifie que la version v_2 a été dérivée à partir de v_1. La figure 2.11 représente différents types de graphes de version.

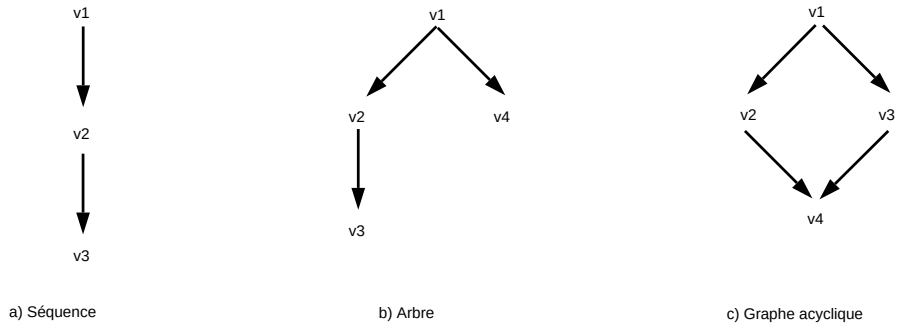

FIG. 2.11 – Graphes de version à un niveau

La partie a) de la figure 2.11 montre le cas le plus restrictif qui représente les versions comme une séquence de révisions. Dans le cas d'un arbre b), les noeuds qui ne sont pas des feuilles peuvent créer des successeurs. Finalement, dans un graphe acyclique c), nous pouvons avoir des versions avec de multiples prédécesseurs. Dans l'ensemble des systèmes qui utilisent le type d'organisation à un niveau, nous pouvons citer PCTE et DAMOKLES [CW98, EJWG03].

Dans le cas d'organisation à deux niveaux, un graphe de version est constitué d'un ensemble de branches, dont chacune est composée d'une séquence de révisions. Cette approche gère deux types de relations : successeurs (à l'intérieur d'une branche) et descendants (entre les branches). La figure 2.12 montre ce type d'organisation.

Les changements exécutés dans une branche peuvent être propagés aux autres branches. Nous pouvons avoir comme résultat un Graphe Direct Acyclique. Néanmoins, les branches ne sont pas reliées et elles continuent d'exister. Le système ClearCase [CW98, EJWG03] utilise cette approche.

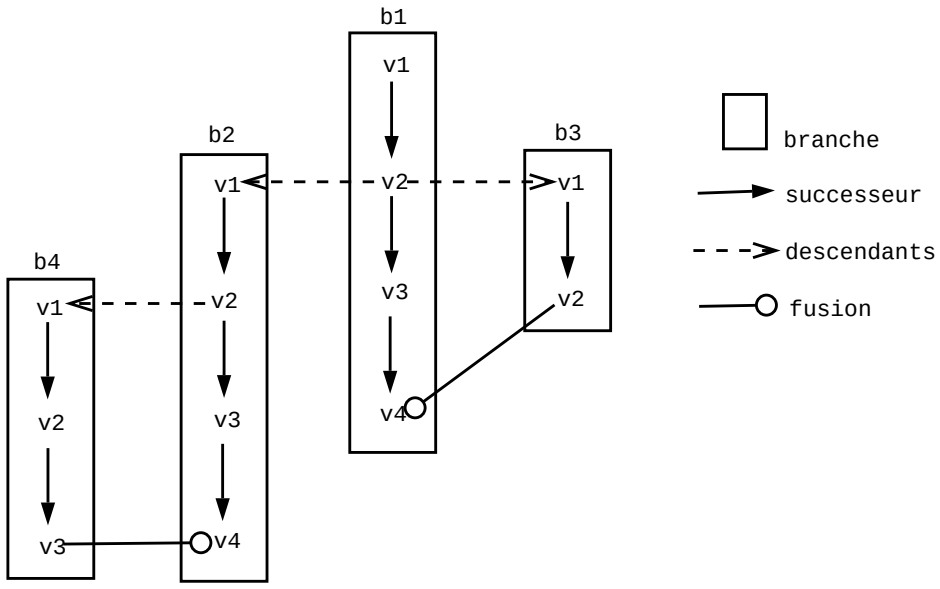

FIG. 2.12 – Graphe de version à deux niveaux

2.6.5 Gestion de versions extensionnelles et intensionnelles

La gestion de versions extensionnelles ou de versions de données, concerne la reconstruction de versions créées à priori. Elle requiert l'identification de la version (VID) et l'immutabilité de la version. Une version est immuable si elle n'autorise pas de changements, ce qui permet d'assurer une récupération sûre des données. Finalement, la reconstruction de versions demande aussi un système de stockage efficace [CW98, ME97, MBJ+02].

En ce qui concerne la gestion de versions intensionnelles (ou versions de schémas), elle traite de la construction de nouvelles versions à partir des descriptions basées sur des propriétés. Cette approche est très utilisée quand un logiciel évolue en plusieurs révisions et variantes où beaucoup de changements doivent être combinés. Les systèmes qui supportent des versions intensionnelles doivent fournir un mécanisme pour résoudre le problème d'explosion des combinaisons et de contrôle de cohérence. Ce mécanisme permet seulement la construction de versions pouvant satisfaire certaines contraintes [CW98, ABCM99].

Dans [ABCM99], les auteurs présentent un modèle unifié de versions extensionnelles. Ce modèle utilise un mécanisme appelé *version concentration* pour réduire le nombre de combinaisons à prendre en compte. Dans [MBJ+02], les auteurs utilisent un modèle temporel de version (TVM) pour permettre le stockage de versions d'objets, ainsi que toute l'histoire des changements pour chaque version. Finalement, dans [GM02], les auteurs présentent un modèle formel pour des versions temporelles de schémas dans les bases de données orientées objets. Ils proposent un schéma évolué qui consiste en une collection de versions de schémas définies sur un ensemble de noms de classes et d'attributs.

2.6.6 Opérations primitives

La possibilité de supporter des changements de schémas est représentée par un ensemble de primitives qui permettent aussi bien des changements au niveau de la structure du schéma, que la mise à jour des données extensionnelles. La liste des opérations primitives définies par un modèle de données particulier représente l'ensemble de changements possibles à exécuter sur le modèle. Par exemple, dans le contexte des bases de données orientées sujets [Lau96, GM99], cette liste a été groupée en : changements de schéma sur un noeud et changements de schéma sur des arêtes. La première catégorie groupe les opérations agissant sur les éléments supportés par le modèle de données orientées-sujets, comme les attributs et les classes. Tandis que les changements sur des arêtes fournissent le support pour l'intégration des caractéristiques d'une version de schéma avec des versions de schémas appartenant à d'autres noeuds.

En plus, dans [GMS00, GM02] les auteurs introduisent la notion de temps pour la création des versions temporelles. A chaque version temporelle nouvelle est associée un temps-valide pertinent. Ainsi, les données stockées peuvent être accédées par le biais des versions de schémas du passé, du présent et du futur.

2.7 Bilan

Dans ce chapitre, nous avons traité le sujet des entrepôts de données qui étendent les concepts et la technologie traditionnelle des bases de données pour créer des systèmes d'aide à la décision.

En utilisant l'architecture d'un entrepôt de données, nous avons expliqué les différents composants qu'il intègre, comme les diverses sources, les types de données et les différents outils pour arriver à la visualisation de l'information. Nous avons décrit les différents modèles multidimensionnels pour la construction d'un entrepôt de données, ainsi que les différentes opérations pour la manipulation des données multidimensionnelles.

L'avant dernière partie a été consacrée aux types de serveurs décisionnels. Dans un premier temps, nous avons décrit le serveur ROLAP qui utilise une base de données relationnelle, tant au niveau du stockage qu'au niveau de la gestion de données. Dans cette architecture, les stratégies d'optimisation représentent le point principal qui distingue les systèmes existants. Nous avons donné également une description des systèmes ROLAP existants sur le marché.

Le serveur MOLAP a été la deuxième architecture que nous avons traitée. Ces types de systèmes utilisent une base de données multidimensionnelle pour le stockage des données. Les systèmes MOLAP doivent gérer le problème de données clairsemées, quand seulement un nombre réduit des cellules d'un cube contiennent

une valeur de mesure associée. La façon de résoudre cette problématique, par les produits commerciaux, représente un des plus importants critères pour les systèmes MOLAP. Nous avons aussi donné une liste des produits existants.

La troisième architecture que nous avons décrite est le serveur HOLAP. Un système de ce type supporte et intègre un stockage de données multidimensionnelles et relationnelles. La plupart des systèmes commerciaux utilisent une approche hybride pour manipuler les informations de l'entrepôt avec un moteur ROLAP, tandis que pour la gestion des magasins des données (*data marts*), ils utilisent l'approche MOLAP. Finalement, nous avons décrit quelques produits commerciaux existants.

En ce qui concerne les projets de recherche dans le domaine des entrepôts de données, nous avons décrit succinctement WHIPS, SIRIUS, DWQ et EVOLUTION. Les deux premiers sont centrés sur des problématiques liées à l'extraction et à la maintenance incrémentale des données. Le projet de la communauté Européenne DWQ traite de la qualité des entrepôts, tandis que le dernier, le projet EVOLUTION du laboratoire PRiSM à Paris, propose le développement d'une méthodologie et d'outils pour l'aide à la conception et à l'évolution des entrepôts.

Dans la dernière partie, nous avons décrit l'approche des versions de schéma. D'abord nous avons donné l'état courant des versions dans les modèles de données existants. Ensuite nous avons décrit l'espace de versions, leurs types, la gestion de données extensionnelles et intensionnelles, ainsi que les opérations primitives à utiliser.

En ce qui concerne l'évolution de schémas, elle permet la récupération des données existantes par le biais de leur adaptation au nouveau schéma. Néanmoins, l'évolution n'implique pas un support historique du schéma et par conséquent ne considère pas un mécanisme pour la visualisation des données à travers des schémas évolués. Dans le paradigme des entrepôts la manipulation de données historiques représente une tâche essentielle. Par conséquent le concept d'évolution de schéma n'est pas suffisant et il est nécessaire établir une stratégie spécifique pour répondre à cette nécessité.

Nous considérons l'approche de versions de schéma comme une alternative aux systèmes qui doivent gérer et manipuler des données historiques. Dans le cas précis du projet ADELEM, nous proposons l'utilisation de versions de schéma pour gérer les aspects temporels de l'entrepôt de données. Pour aboutir à cela, nous devons établir un mécanisme pour la gestion, le stockage et la visualisation de l'ensemble de données (courantes et historiques) médicales.

Nous présentons dans la suite une architecture et un modèle que nous proposons pour un système décisionnel dans le contexte médical.

Chapitre 3

Architecture et modèle pour un entrepôt de données médicales

La conception d'un entrepôt de données est une tâche complexe et délicate. Elle se compose de plusieurs étapes. Dans un premier temps, nous devons analyser l'ensemble des sources de données internes et externes. Cette analyse sert aussi bien à la sélection de l'ensemble de données à stocker dans l'entrepôt, qu'à la sélection des outils requis pour l'extraction et la transformation de ces données avant leur stockage. La deuxième étape consiste à organiser ces données à l'intérieur de l'entrepôt. Pour ce faire, nous devons concevoir le modèle multidimensionnel à utiliser ainsi que définir l'ensemble optimal de vues à matérialiser. Finalement, la troisième étape consiste à déterminer les outils nécessaires pour la visualisation de l'ensemble des données.

Dans le chapitre précédent, nous avons présenté l'architecture typique d'un entrepôt de données. Dans cette architecture, nous trouvons les trois étapes déjà décrites : extraction-intégration, organisation et interrogation. Néanmoins, en ce qui concerne le concept d'organisation, nous considérons qu'il est pertinent de faire une distinction entre la structuration et l'organisation. Pour nous le terme de structuration correspond à la conception du modèle multidimensionnel et à la sélection des vues à matérialiser, tandis que l'organisation entraîne la gestion de l'ensemble de données (intensionnelles et extensionnelles) courantes et historisées stockées soit à l'intérieur, soit à l'extérieur de l'entrepôt.

Ce chapitre présente la première partie du processus de structuration, à savoir, la description de la conception d'un modèle multidimensionnel. Nous proposons d'abord une architecture pour un système décisionnel. Cette architecture comprend trois composants qui traitent les deux derniers processus (la structuration-organisation et l'interrogation). Dans une deuxième partie, nous définissons un métamodèle multidimensionnel qui est composé de trois classes : Cube, Dimension et Hiérarchie. Ce métamodèle sert à la conception d'un modèle multidimensionnel avec quatre définitions : le schéma d'une base multidimensionnelle, d'un cube, d'une dimension et d'une hiérarchie. Nous donnons aussi la définition d'instances de chacune

des classes.

Nous avons utilisé des données médicales réelles, ainsi, les dernières parties de ce travail présentent l'application de notre modèle sur ces données.

3.1 Architecture d'un système décisionnel

Dans cette partie, nous décrivons l'architecture qui a été conçue. Elle est basée sur trois composants. Le premier est l'**Interface pour la Génération (Semi-automatique) d'Indicateurs**. Nous plaçons ce composant à l'intérieur du processus d'interrogation. Les composants de l'**Entrepôt de données** et les **Versions de Schémas Bitemporels** font partie du processus de structuration-organisation.

La figure 3.1 montre notre architecture ainsi que les différents liens entre les composants :

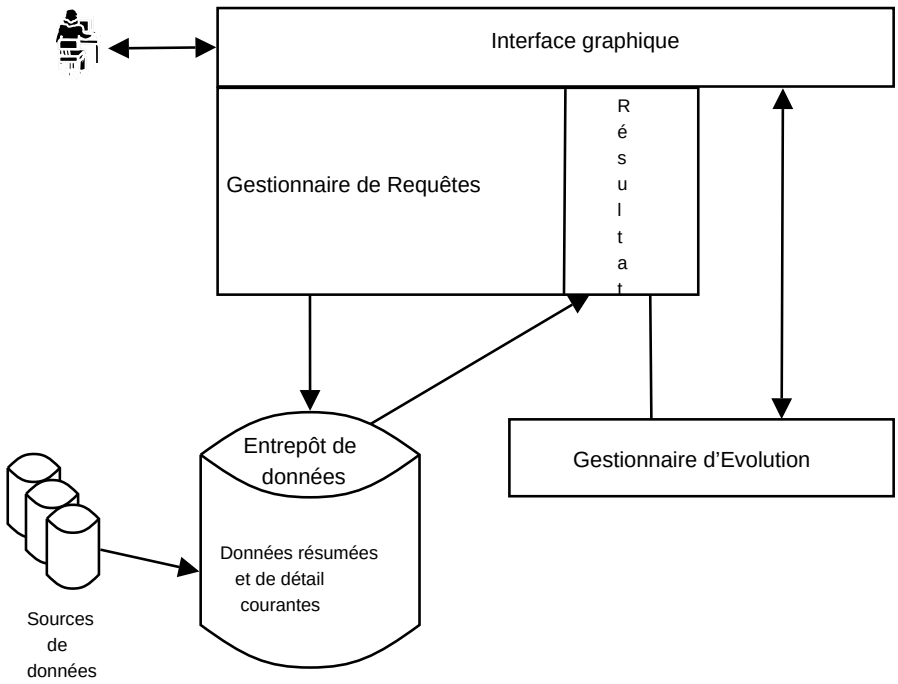

FIG. 3.1 – Architecture proposée d'un système décisionnel

3.1.1 Interface graphique pour la Génération (Semi - automatique) d'Indicateurs

Nous décrivons les principaux éléments de l'interface graphique [SA04] :

Interface Graphique : Outil graphique où les utilisateurs peuvent choisir les divers composants (relations, mesures et propriétés) correspondant à l'indicateur

exprimé. L'objectif de l'interface est de faciliter la génération semi-automatique des requêtes.

Gestionnaire de Requêtes : Composant pour traduire le méta-schéma en schéma local (relationnel), pour générer le code SQL et pour exécuter la requête sur les données de l'entrepôt.

Résultat : Les données qui correspondent à la requête et qui doivent être affichées par l'interface graphique.

Le chapitre 4 contient une description détaillée de notre interface.

3.1.2 Entrepôt de données

Entrepôt de Données : Représente la collection des données à consulter. L'entrepôt contient aussi bien des données de détail que résumées. Par exemple :

Données de détail : "Le nombre de ventes par magasin, par produit et par ville du mois de janvier 2000".

Données résumées : Nous pouvons avoir des données légèrement ou fortement résumées. Par exemple, "les ventes mensuelles par magasin et par produit des dix dernières années" sont des données faiblement résumées, tandis que "les ventes semestrielles par région des dix dernières années" sont fortement résumées.

Sources : Elles rassemblent l'ensemble de sources internes, par exemple, les différents systèmes opérationnels qui sont utilisés pour mettre à jour les données stockées dans l'entrepôt. Les sources externes sont des données qui n'appartiennent pas à l'entreprise et qui sont souvent achetées, par exemple, les données géographiques.

3.1.3 Gestionnaire d'Evolution

Il est responsable de la gestion, du stockage et de la visualisation des données historisées. Nous proposons l'utilisation des versions de schémas bitemporels pour la gestion de l'évolution des données intensionnelles et extensionnelles. Le chapitre 5 décrit en détail ce composant.

3.2 Métamodèle Multidimensionnel

Il représente la structure pour la mise en oeuvre des modèles de type multidimensionnel (*cf.* Section 2.3) qui se compose des métaclasses[1]. La figure 3.2 montre les

[1]Une métaclasse est une classe qui modélise des classes [PCTM02, PCTM03]. Nous avons utilisé UML (*Unified Modeling Language*) pour la spécification du métamodèle multidimensionnel.

Métaclasses `Cube, Dimension` et `Hiérarchie` qui constituent notre **MétaSchéma**.

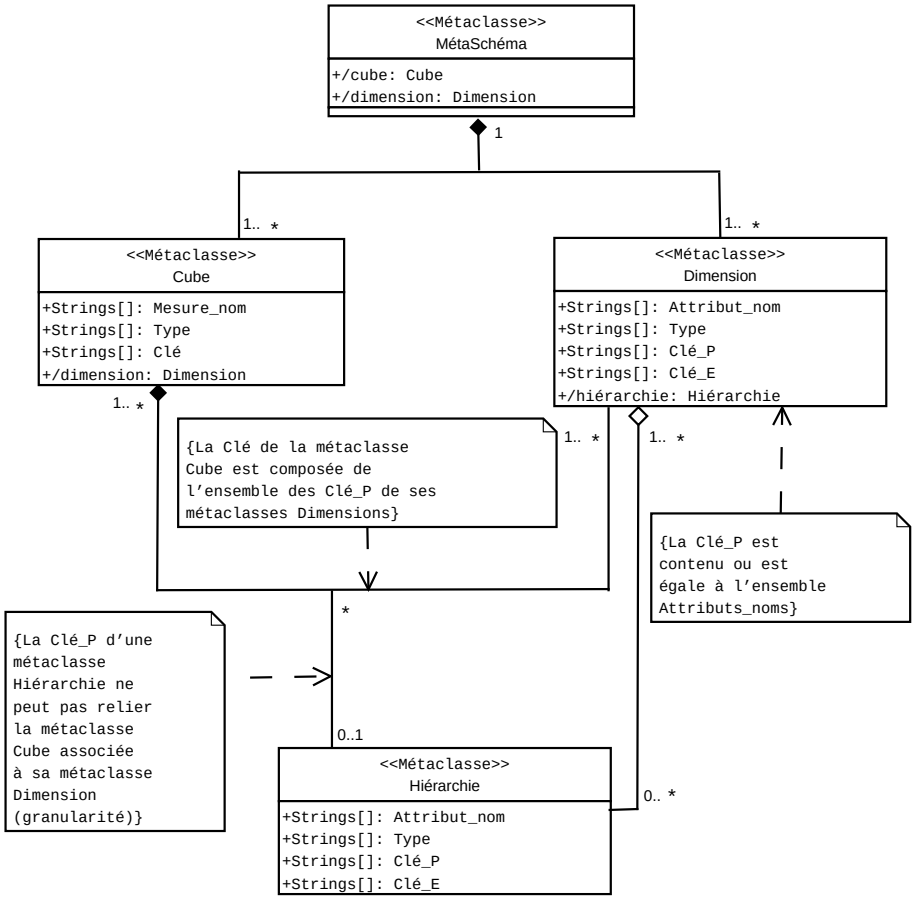

FIG. 3.2 – Métamodèle Multidimensionnel

La spécification du notre métamodèle, nous permet de représenter et de visualiser les différents composants. Ceci, nous a permis d'abord pour identifier les associations et leur cardinalité, mais principalement pour identifier l'ensemble de restrictions, soit à niveau des associations, soit à l'intérieur d'un composant.

3.2.1 Relations entre les métaclasses

Pour relier les métaclasses, nous pouvons utiliser les types de relations suivants [PCTM03, Var00, VQVJ00] :

Association : C'est une relation entre deux classes qui représente les connections entre leurs objets. Une association est bidirectionnelle ou unidirectionnelle.

Agrégation : Elle spécifie une relation de composition et peut être :

Partagée : Un élément composant à l'intérieur d'un ou de plusieurs éléments composites ; ou

Composée : L'ensemble de composants intègre l'élément composite.

Généralisation : C'est une relation entre une classe plus générale et une classe plus spécifique.

Dépendance : C'est une relation entre deux classes où les changements appliqués sur la classe indépendante affectent la classe dépendante.

3.2.2 Contraintes entre les métaclasses

Les contraintes permettent de compléter la spécification du métamodèle. Ils existe deux types de contraintes : **syntaxiques** ou **sémantiques**. Par exemple, la contrainte : une classe est représentée graphiquement par un rectangle compartimenté, est une contrainte syntaxique. Nous avons classé les contraintes sémantiques de la manière suivante :

Contraintes de cardinalité : Représentent le nombre de liens qui peuvent exister entre deux classes. Par exemple : 0..1 (zéro ou un), 1 (un et un seul), * (zéro à plusieurs), M..N (de M à N),...

Contraintes d'intégrité référentielle : Représentent le fait qu'un ensemble d'attributs qui appartient à une classe apparaît dans une autre classe. Par exemple, soient $r_1(R_1)$ et $r_2(R_2)$ deux classes avec les clés primaires C_1 et C_2. Le sous-ensemble α de R_2 est une clé externe qui fait référence à C_1 de r_1, si pour chaque t_2 de r_2 existe une n-uplet t_1 de r_1 tel que $t_1[C_1] = t_2[\alpha]$.

Pour décrire des contraintes additionnelles qui ne sont pas exprimées directement par la structure du métamodèle, nous pouvons utiliser le langage OCL (*Object Constraint Language*) fourni par UML ou décrire les contraintes en langue naturelle. Par exemple, nous avons la contrainte : {La Clé_P d'une métaclasse **Hiérarchie** ne peut pas relier la métaclasse **Cube** associée à sa métaclasse **Dimension** (granularité)}.

3.2.3 Description des Métaclasses

Nous décrivons les métaclasses qui composent le métamodèle multidimensionnel :

Métaclasse MétaSchéma : La métaclasse MétaSchéma est composée des métaclasses **Cube** et **Dimension**. Elle représente un schéma multidimensionnel composé des classes **Cube** et **Dimension**.

Métaclasse Cube : La métaclasse **Cube** modélise des classes de type **Cube**. Elle représente les différentes mesures d'analyse. Une métaclasse **Cube** est représentée par deux ensembles de couples :

⟨ **Mesure_nom : string, Type : string** ⟩ représentent le(s) sujet(s) d'analyse.

⟨ **Clé : string, Type : string** ⟩ représentent l'ensemble des métaclasses Dimensions (perspectives d'analyse) qui appartiennent à la métaclasse **Cube**.

Métaclasse Dimension : Cette métaclasse modélise des classes de type **Dimension**. Elle représente une perspective d'analyse ou un axe de localisation d'une cellule d'un **Cube**. Une métaclasse **Dimension** est composée d'ensembles de couples de type :

⟨ **Attribut_nom : string, Type : string** ⟩ ensemble d'attributs textuels à l'intérieur de la classe **Dimension**.

⟨ **Clé_P : string, Type : string** ⟩ identificateur unique pour la classe **Dimension**.

⟨ **Clé_E : string, Type : string** ⟩ représentent des liens entre les classes **Dimension** et **Hiérarchie**.

Métaclasse Hiérarchie : Elle représente les relations de dépendance entre les niveaux à l'intérieur d'une classe **Hiérarchie**. Une métaclasse **Hiérarchie** est composée des ensembles de couples de type :

⟨ **Attribut_nom : string, Type : string** ⟩ représentent des paramètres textuels dans la classe **Hiérarchie**.

⟨ **Clé_P : string, Type : string** ⟩ identificateur unique pour la classe **Hiérarchie**.

⟨ **Clé_E : string, Type : string** ⟩ représentent des liens entre les classes de type Hiérarchie.

3.2.4 Définition d'un modèle multidimensionnel

Un modèle multidimensionnel est une instance de la métaclasse **MétaSchéma**. Nous l'appelons classe **ModèleMultidimensionnel (MM)**. Nous utilisons les descriptions des métaclasses de la section précédente pour aboutir à la définition d'une base de données multidimensionnelle. Nous sommes partis de travaux existants [Ben02, GM02, HMV99a, HMV99b, LW96, Qui99] pour la définition d'un schéma et d'une instance. La plupart des travaux proposent un ensemble d'opérateurs pour

l'évolution d'un système multidimensionnel. Dans [GM02], l'auteur propose un modèle formel des versions de schémas temporelles pour des bases de données à objets.

Nous faisons la différence entre une dimension et une hiérarchie. Ainsi, notre modèle se compose de quatre éléments essentiels : une base multidimensionnelle, un cube, une dimension et une hiérarchie. Nous donnons pour chacun la définition de schéma et d'instance. Nous supposons l'existence de :

DOM = dom(char) $\bigcup$ dom(integer) $\bigcup$ dom(float) $\bigcup$ dom(decimal) $\bigcup$ dom(date) $\bigcup$ $\{t\}$ contenant le domaine de chaque type atomique plus la valeur distinguée t.
C = Ensemble de noms de cubes.
D = Ensemble de noms de dimensions.
H = Ensemble de noms de hiérarchies.
M = Ensemble de noms de mesures.
P = Ensemble de noms de propriétés.
L = Ensemble de noms de niveaux.
R = Ensemble de contraintes.

Nous associons à chaque cube $c \in C$ un ensemble de valeurs $\{a_1, ..., a_n\}$ tel que $\forall_{i=1,...n} a_i \in DOM$. De même pour chaque dimension $d \in D$, nous associons un ensemble de valeurs $\{b_1, ..., b_n\}$ tel que $\forall_{i=1,...n} b_i \in DOM$. Finalement, pour chaque niveau $l \in L$, nous associons un ensemble de valeurs $\{f_1, ..., f_n\}$ tel que $\forall_{i=1,...n} f_i \in DOM$.

Nous supposons l'existence des fonctions suivantes :

measuredom : $M \longrightarrow E(DOM)$[2], qui retourne l'ensemble des valeurs associées à une mesure.

propertydom : $P \longrightarrow E(DOM)$, qui retourne l'ensemble des valeurs associées à une propriété.

levelset : $H \longrightarrow E(DOM)$, qui retourne l'ensemble des membres associés à un niveau.

Définition 3.1 : Schéma de base de données et d'instance multidimensionnelle.

Le schéma d'une base de données multidimensionnelle est un n-uplet $SM = (C_s, D_s, H_s, R)$, où C_s est un ensemble de schémas de cubes, D_s est un ensemble de schémas de dimensions, H_s est un ensemble de schémas de hiérarchies et R est un ensemble de contraintes. □

[2]L'ensemble $E(A)$ est l'ensemble des parties de A. $E(A) = \{X | X \subset A\}$.

L'instance d'une base multidimensionnelle est un n-uplet IM = (C_i, D_i, H_i, R_i), où C_i est un ensemble d'instances de cubes tel que, pour chaque instance $c_i \in C_i$, il existe un schéma $c_s \in C_s$ avec c_i conforme à c_s. D_i est un ensemble d'instances de dimensions tel que, pour chaque instance $d_i \in D_i$, il existe un schéma $d_s \in D_s$ avec d_i conforme à d_s. H_i est un ensemble d'instances de hiérarchies tel que, pour chaque instance $h_i \in H_i$, il existe un schéma $h_s \in H_s$ avec h_i conforme à h_s. Finalement, R_i est un ensemble d'instances de contraintes tel que chaque instance $r_i \in R_i$. □

Définition 3.2 : Schéma et instance du cube.

Un schéma de cube est un n-uplet C_s = (c_n, M, D), où $c_n \in C$ est le nom du cube, M est un ensemble de mesures et D est un ensemble de dimensions. □

Une instance de cube est un ensemble de cellules. Une cellule est un n-uplet I_c = (c_n, {v_1,...,v_k}, {d}), où c_n est le nom du cube, {v_1,...,v_k} est l'ensemble de mesures tel que $\forall_{i=1,...n} v_i \in$ measuredom(m_i) avec $m_i \in M$ et $d \in D$ est l'ensemble de dimensions. □

Par exemple, la figure 3.3 représente le schéma et l'instance du cube Ventes.

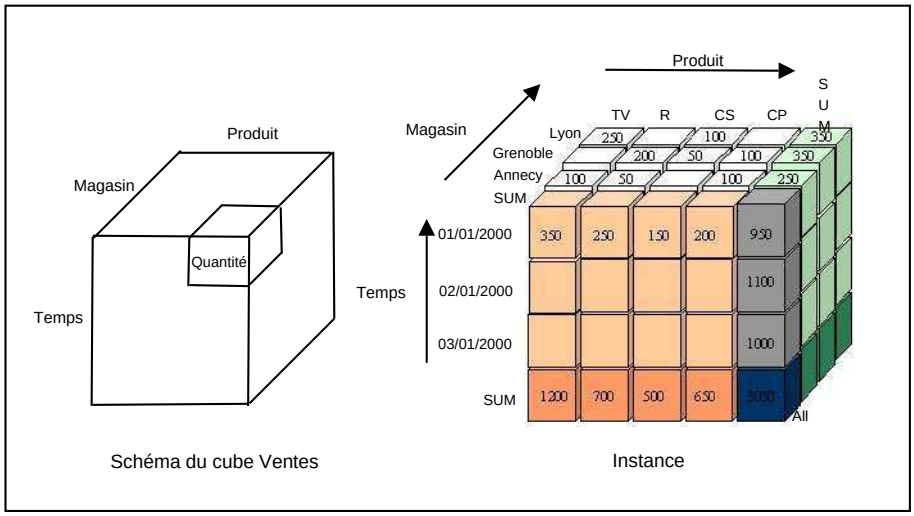

FIG. 3.3 – Schéma et une instance possible du Cube

La partie gauche montre le schéma du cube de nom Ventes. L'ensemble {Temps, Magasin, Produit} sont les axes du cube et la mesure est Quantité. La partie droite montre une instance de ce cube. Par exemple, nous avons 100 Téléviseurs vendus dans le magasin d'Annecy le 1er janvier 2000.

Définition 3.3 : Schéma et instance de dimension.

Un schéma de dimension est un n-uplet D_s = (d_n, P, H), où $d_n \in D$ est le nom de la dimension, P est l'ensemble de propriétés de la dimension d_n et H est l'ensemble

de hiérarchies. □

Une instance de dimension est un n-uplet $I_d = (d_n, \{(v_1,...,v_i)\}, \{h\})$*, où* $d_n \in D$ *est le nom de la dimension,* $(v_1,...,v_i)$ *est un ensemble de propriétés tel que* $\forall_{j=1,...n} v_j \in propertydom(p_j)$ *avec* $p_j \in P$*. Enfin,* $h \in H$ *est l'ensemble de hiérarchies qui appartiennent à* d_n*.* □

Par exemple, le schéma de la dimension Magasin est défini de la manière suivante :

d_n = Magasin

$\{p_1,...p_i\}$ = {Cle_Magasin, Libelle_Magasin }

h_n = H_Geo (*cf.* Définition 3.4)

Schéma et instance de hiérarchie.

Pour nous un schéma et une instance de hiérarchie peuvent prendre la forme d'un graphe séquentiel ou d'un graphe acyclique dirigé représentant les hiérarchies de niveaux. Chaque noeud du graphe contient un niveau et un arc entre deux noeuds représente une association entre les niveaux contenus par les noeuds. Nous supposons l'existence de deux noeuds, nommés *base* et *sommet*, à partir desquels tous les autres noeuds peuvent être atteints directement ou par transitivité. Le noeud *sommet* contient le niveau $\top$ de la hiérarchie. La figure 3.4 présente un exemple de graphe séquentiel. Comme exemple de graphe acyclique dirigé, nous pouvons considérer le schéma de hiérarchie suivant : H_Temps = ({(Jour, Mois), (Mois, Semestre), (Semestre, Annee)}, {(Jour, Semaine), (Semaine, Annee)}), où la *base*=Jour et le *Sommet*=Annee. Dans notre modèle, nous considérons les caractéristiques suivantes :

1. Nous distinguons les termes de dimension et de hiérarchie. Ainsi, une dimension peut relier 0, une ou plusieurs hiérarchies. Pour une hiérarchie donnée, il y a une liaison directe avec la dimension associée dans le cube. Par exemple, si le cube `Ventes` comporte la dimension `Magasin` reliée à une commune, ce cube sera alors indirectement lié aux niveaux supérieurs à commune (Département, Région, $\top$). Ainsi, il sera possible d'effectuer des agrégats sur ces niveaux supérieurs.
2. Nous pouvons insérer un niveau de hiérarchie soit à côté de la hiérarchie déjà définie soit à l'intérieur. Pour faire cela, nous avons besoin de spécifier trois niveaux, le premier est le niveau à insérer, les deux derniers représentent les niveaux inférieur et supérieur à partir desquels le nouveau niveau sera placé.
3. Dans la définition du schéma de hiérarchie, nous considérons le cas où $l=\top$, c'est à dire quand nous avons seulement les niveaux *base* et *sommet*.

Nous donnons dans la suite la définition de schéma et d'instance de hiérarchie.

Définition 3.4 : Schéma et instance de hiérarchie.

Un schéma de hiérarchie est un n-uplet $H_s = (h_n, L, \prec)$*, où* $h_n \in H$ *est le nom de la hiérarchie,* L *est l'ensemble de niveaux à l'intérieur de la hiérarchie* h_n *et le symbole* $\prec$ *est une relation de dépendance entre les niveaux telle que sa fermeture transitive et réflexive* $\preceq^*$ *représente un ordre partiel dans* L*. Il existe un seul niveau* $l_\perp$ *tel que (i) si* $l \neq \top$*,* $\forall_{l \in L}\ l_\perp \preceq^* l \wedge \forall_{l \in L}\ l \preceq^* \top$ *et (ii) si* $l = \top$*,* $l_\perp \preceq^* \top$*. Un niveau* $l_j \in L$ *est un niveau immédiatement supérieur (relation de dépendance) de* $l_i \in L$ *si* $l_i \prec l_j$*.* □

Une instance de hiérarchie organise les propriétés participant aux hiérarchies d'agrégation. Pour chaque arc dans un schéma de dimension, il existe une fonction de RUP (*rollup*, *cf.* Définition d'instance en-dessous) associée.

Une instance de hiérarchie est un n-uplet $I_h = (\bigcup_{l_i \in L} levelset(l_i), RUP)$*, où RUP est un ensemble de fonctions* $rup_{l_i}^{l_j}$ *tel que (i)* $\forall (l_1, l_2) \in L$ *tels que* $l_1 \prec l_2$*, il existe une fonction* $rup_{l_1}^{l_2} : levelset(l_1) \longrightarrow levelset(l_2)$ *et (ii)* $\forall (l_1, l_2, l_3) \in L$ *tels que* $l_1 \prec l_2$ *et* $l_2 \prec l_3$*,* $ran(rup_{l_1}^{l_2}) \subseteq dom(rup_{l_2}^{l_3})$*.* □

Par exemple, dans la figure 3.4, la partie gauche présente le schéma de la hiérarchie H_Geo. Il est défini de la manière suivante : h_n est le nom de la hiérarchie H_Geo, l'ensemble de propriétés (niveaux) à l'intérieur de la hiérarchie est {Commune, Département, Région, $\top$} et la relation $\prec$ est représentée par {(Commune, Département), (Département, Région), (Région, $\top$)}.

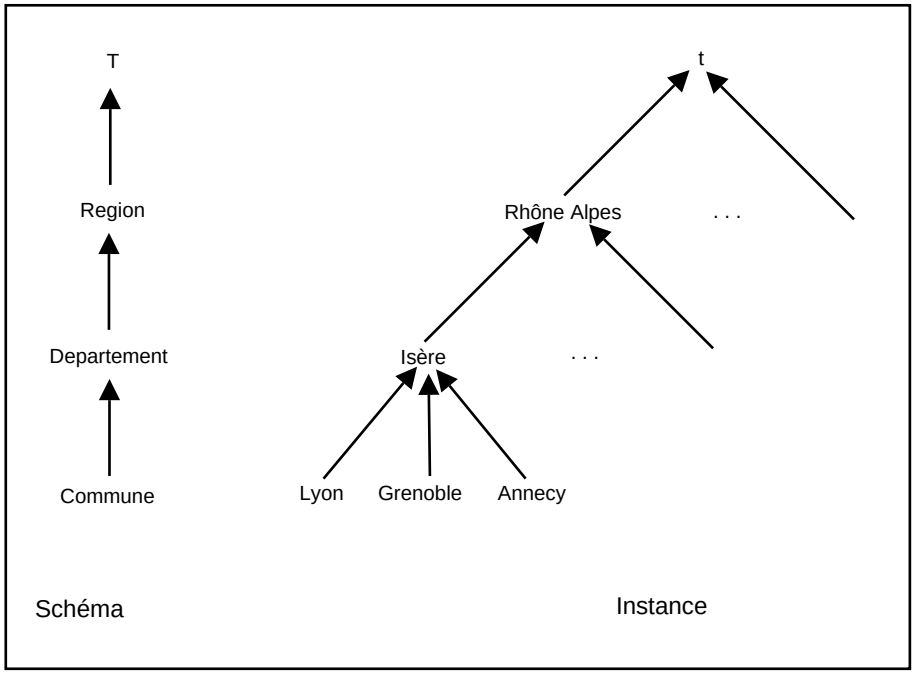

FIG. 3.4 – Schéma et une instance possible de la hiérarchie H_Geo

La partie droite montre une instance possible de ce schéma. La fonction $\text{rup}_{Commune}^{Departement}$ met en relation les propriétés `Lyon, Grenoble` et `Annecy` du niveau `Commune` avec la propriété `Isère` du niveau `Departement`. Les propriétés du niveau `Region` sont mises en relation avec la propriété unique t du niveau $\top$.

Le reste du chapitre montre une application de notre modèle. Nous utilisons des sources de données réelles et un ensemble d'indicateurs dans le cadre du projet ADELEM.

3.3 Analyse des données d'une application médicale

Cette section est consacrée à l'analyse des données du projet. D'abord, nous analysons l'ensemble des sources existantes, ce qui nous permet d'identifier deux types de données, les données publiques concernant la santé (RSA, RHA, CIM10 et FINESS) et les données démographiques (RP99) [Bel02].

3.3.1 Sources de données

Nous décrivons les fichiers RSA (Résumés de Sorties Anonymes), RHA (Résumé Hebdomadaire Anonyme), FINESS (Fichier National des Établissements Sanitaires et Sociaux), CIM10 (Classification International de Maladies version 10). Ces fichiers sont hétérogènes et répartis et ils devront être intégrés (les préparer, les nettoyer, les transformer, etc.), avant leur stockage dans l'entrepôt de données.

Fichier RSA (Résumés de Sorties Anonymes) Tout séjour hospitalier, effectué dans la partie court séjour d'un établissement, fait l'objet d'un Résumé de Sortie Standardisé RSS). Un RSS est constitué d'un ou plusieurs Résumé(s) d'Unité Médicale (RUM). La transmission d'informations médicales à la direction de l'établissement ou à l'extérieur de celui-ci, s'opère sur la base de données agrégées ou Résumés de Sortie Anonymes (RSA).

Le fichier RSA est obtenu par transformation automatisée des RSS. A partir du fichier de RSS groupés, le médecin responsable du DIM (Département d'Information Médicale) utilise le logiciel GENRSA (Générateur de RSA), propriété de l'Etat, pour produire le fichier de RSA. Le RSA comprend toujours un enregistrement unique par séjour (environ 15 à 18 millions d'enregistrements pour chaque année) [PMS04].

Caractéristiques du fichier :

- Propriétaire : Agence Régionale d'Hospitalisation
- Type du fichier : Texte (ASCII fixe) ou Access
- Année : 2000
- Mode d'obtention : gratuit
- Mise à jour : annuelle

- Taille : 11 Go

Fichier RHA (Résumé Hebdomadaire Anonyme) Il comprend environ 1600 établissements avec 28 millions de journées d'hospitalisation qui correspondent à 18% de l'activité hospitalière. Etant donnée que la moyenne d'un séjour est d'environ 35 jours, nous avons calculé une taille de 4 millions d'enregistrements hebdomadaires annuels ((28000000 / 35)*5) pour le fichier RHA.

Les objectifs sont doubles : le premier est la poursuite de soins médicaux et de rééducation, le deuxième est la réadaptation (sociale, professionnelle, scolaire,...) [PMS04].

Caractéristiques du fichier :

- Propriétaire : Agence Régionale d'Hospitalisation
- Type du fichier : Texte (ASCII fixe) ou Access
- Mode d'obtention : gratuit

Fichier FINESS (Fichier National des Établissements Sanitaires et Sociaux) Recense les structures sanitaires et sociales au niveau national (5079 établissements pour la France Métropolitaine) [SMS04].

Caractéristiques du fichier :

- Propriétaire : Ministère Chargé des Affaires Sanitaires et Sociales
- Type du fichier : Texte (ASCII fixe) au Excel
- Année : 2000
- Mode d'obtention : gratuit
- Mise à jour : annuelle
- Taille : 2.24 Mo

Fichier CIM10 (Classification International de Maladies version 10) Ce fichier contient le code et le libellé pour chaque maladie (diagnostic) environ 17694 enregistrements. La taille du fichier est de 1.34 Mo.

Dans la suite, nous décrivons la source de données démographiques RP90/99 (recensement de la population de 1990 et 1999).

Fichier RP90/99 Ce fichier recense la population de chaque commune répartie en 96 tranches d'âge : < 1 an, 1 an ≤ 2 ans, ..., 95 ans et plus. Les communes sont désignées par leurs codes administratifs et leurs intitulés.

Caractéristiques du fichier :

- Propriétaire : INSEE
- Type du fichier : BDF
- Année : 1999
- Mode d'obtention : achat
- Taille : 50.8 Mo (fichier avec tranches d'âge d'hommes et de femmes par commune)

L'annexe A contient une description en détail des sources RSA et FINESS.

3.3.2 Analyse des données

Dans cette partie nous analysons les différentes sources décrites précédemment pour proposer des mécanismes adaptés selon les caractéristiques des informations. Nous proposons d'abord un tableau qui contient les propriétés des sources. Ensuite, nous donnons un tableau d'analyse de données en considérant une dizaine d'années de stockage.

Le tableau 3.1 décrit les différentes propriétés des sources :

Source	**Propriétaire**	**Type de fichier**	**Mode d'obtention**	**Taille**	**Année**	**Mise à jour**
RSA	ARH	Texte (ASCII fixe) ou Access	Gratuit	11 Go	2000	Annuelle
RHA	ARH	Texte (ASCII fixe) ou Access	Gratuit	3 Go	2000	Annuelle
FINESS	MCASS	Texte (ASCII fixe) ou Access	Gratuit	2.24 Mo	2000	Annuelle
CIM10	OMS	Excel	-	1.34 Mo	1995	-
RP90/99	INSEE	DBF	Achat (625,04)	50.8 Mo	1999	Tous les 9 ans[3]

TAB. 3.1 – Description des sources de données

Le tableau précédent indique les caractéristiques essentielles à prendre en compte lors des premières phases de la conception d'un entrepôt. Néanmoins, si nous reprenons les deux dernières propriétés, l'année et la mise à jour, pour les représenter dans un tableau 3.2 qui aura des données sur 10 ans, nous avons les caractéristiques des données historisées.

L'analyse du tableau précédent, nous permet de remarquer les sources RSA (Résumés de Sortie Anonymes) et RHA (Résumés Hebdomadaire Anonymes). Leurs

[3]Il a été décidé une mise à jour annuelle des résultats du recensement par le biais d'une nouvelle méthode de collecte. Le départ de cette nouvelle collecte a été fixé au début de l'année 2004 pour obtenir les premiers résultats à la fin de l'année 2008. A partir de cette année (2008) des résultats récents et réguliers seront publiés sur la population de chaque commune, avec une ancienneté maximale de trois ans.

[4]A partir de 2008.

Source	Taille	Optimisation du stockage	Rafraîchissement	Requêtes complexes	Agrégats	Evolution du schéma
RSA	110 Go	Oui	Annuel	Oui	Oui	Oui
RHA	30 Go	Oui	Annuel	Oui	Oui	Oui
FINESS	22.4 Mo	Non	Annuel	Non	Non	Oui
CIM10	13.4 Mo	Non	-	Non	Non	Oui
RP90/99	508 Mo	Oui	Annuel[4]	Non	Oui	Oui

TAB. 3.2 – Caractéristiques des sources de données historiques

tailles nous obligent à rechercher des mécanismes d'optimisation pour le stockage (partitionnement, parallélisme, agrégats) et pour traiter des requêtes complexes.

Une autre partie à gérer est l'évolution du schéma, pour prendre en compte des changements dans la structure logique de la source et qui peuvent avoir des répercussions sur la structure logique de notre entrepôt, voire l'affecter. Ceci nous oblige à proposer une conception adaptable de l'entrepôt de données pour mieux intégrer ces changements à notre schéma.

Dans les chapitres suivants, nous reprenons cette problématique. Par exemple, pour le cas du volume de stockage à gérer, nous devons définir l'ensemble optimal des vues à matérialiser en considérant leur coût de calcul pour la matérialisation et leur coût de stockage (*cf.* Chapitre 4). Par rapport à l'évolution du schéma, nous proposons (*cf.* Chapitre 5) l'utilisation des versions de schémas bitemporels pour la gestion des aspects temporels de l'entrepôt de données. Pour cette raison, nous avons décidé de garder toutes les données, même celles qui ne changent pas, car elles formeront les versions historisées annuelles.

3.4 Classification des indicateurs du projet

Nous énumérons les divers indicateurs, qui ont été créés pour le projet ADELEM. Nous les avons divisés en trois types, les deux premiers rassemblent les différents indicateurs créés par le Laboratoire de Biométrie et Biologie de Lyon (les indicateurs d'offre "géographiques" et les indicateurs de consommation, de besoins et de flux "temporels"). Le troisième type sont des indicateurs que nous proposons, ils correspondent à des indicateurs "temporels", qu'il est intéressant de prendre en compte.

Dans les divers types d'indicateurs, il existe des valeurs qui sont des **paramètres** de la requête. Par exemple dans l'indicateur : "Nombre de séjours par maladie de l'établissement CHU GRENOBLE durant le 1er janvier 2000", nous trouvons pour l'attribut `Raison_sociale` de la dimension `Etablissement` la valeur "CHU GRENOBLE" ainsi que la valeur "01-01-2000" pour l'attribut `Jour` de la dimension `Temps`.

3.4.1 Indicateurs d'offre (géographiques - spatio-temporels)

Représentent des ressources que possède chaque Établissement, et qui permettent de constater sur une carte la répartition d'offres, en fonction des besoins sanitaires dans le système de soins d'une Région.

Indicateur d'offre 1 : Localiser sur une carte tous les établissements de court séjour en faisant apparaître leur capacité en nombre de lits MCO (Médecine-Chirurgie-Obstétrie).

Indicateur d'offre 2 : Localiser sur une carte toutes les maternités avec leur niveau (I, II ou III).

Niveau I : Maternité sans service de réanimation ou de néonatalogie.
Niveau II : Maternité avec un service de néonatalogie.
Niveau III : Maternité avec un service de réanimation néonatale.

Indicateur d'offre 3 : Représenter un ou plusieurs indices d'attraction pour chaque Établissement.

3.4.2 Indicateurs de consommation, de besoins et de flux (temporels)

Cette section rassemble des indicateurs de trois types : consommation de soins, besoins et flux de patient.

a) Indicateurs de consommations de soins : Ils décrivent les séjours dans les différents Établissements de soins. L'équipe de Lyon a sélectionné trois grandes catégories d'activité médicale (l'Obstétrique, les AVC et l'Infarctus), néanmoins dans notre entrepôt avec une dimension par Maladie et une autre par Établissement, nous pouvons analyser les divers séjours pour n'importe quelle maladie dans n'importe quel hôpital. Voici les indicateurs qui ont été retenus :

Indicateur de consommation 1 : Le nombre annuel de séjours par établissement.

Indicateur de consommation 2 : Représentation du nombre annuel d'accouchements par maternité et région.

Indicateur de consommation 3 : Représentation des effectifs de nouveau-nés par maternité et par poids de naissance.

Indicateur de consommation 4 : Représentation des effectifs de nouveau-nés par poids de naissance et par niveau de maternités (niveau I, II ou III).

Indicateur de consommation 5 : Provenance des accouchements pris en charge dans tous les établissements de la région ou dans une sélection d'établissements.

b) Indicateurs de besoins : Ces indicateurs définissent l'ensemble des individus susceptibles d'être pris en charge par un établissement de santé. Par exemple, dans obstétrique, un indicateur de besoin serait le nombre de femmes en âge de procréer actuellement ou bien une prévision de cet effectif dans les 5 à 10 années à venir.

Indicateur de besoin 1 : Population dans les diverses tranches d'âge par secteurs géographiques (communes, département, région ou pays).

Indicateur de besoin 2 : Nombre de personnes dans les diverses tranches d'âge dans 5 ans, 10 ans, selon le lieu d'habitation.

c) Indicateurs de flux de patient : Ces indicateurs s'intéressent soit à l'origine des données (d'où viennent les patients) soit à leur destination (où vont-ils).

Indicateur de flux 1 : D'où proviennent les mères qui accouchent dans les maternités (éventuellement selon le niveau de maternité) ?

Indicateur de flux 2 : Où vont les mères qui accouchent en fonction de leur lieu d'habitation ?

3.4.3 Nouveaux indicateurs de consommation et de flux (temporels)

Ce sont des indicateurs qui permettent de faire des analyses de comparaison, soit sur l'âge des patients, soit sur le mode de sortie de l'unité médicale.

a) Indicateurs de consommation : Indicateurs de comparaison sur l'âge du patient. En gardant l'âge du patient dans la relation `Prise_MCO`, on peut faire des analyses comme :

Indicateur de comparaison 1 : Nombre annuel de patients par établissement et par âge.

Indicateur de comparaison 2 : Nombre annuel de personnes de plus de 60 ans par maladie et par établissement.

b) Indicateurs de flux : Indicateurs de comparaison sur le mode de sortie. Avec cette dimension on peut faire des analyses sur le mode de sortie, en utilisant le code suivant :

6 = mutation, le départ vers une autre unité médicale de la même entité juridique.

7 = transfert normal, le départ vers une autre entité juridique.
8 = domicile, le patient rentre chez lui.
9 = décès, le patient est décédé dans l'unité médicale.
0 = transfert pour ou après réalisation d'un acte.

Indicateur 1 : Nombre annuel de patients qui sont transférés (code 7) par maladie et par établissement.

Indicateur 2 : Nombre annuel de patients décédés par maladie, par établissement et par région.

3.5 Application du modèle multidimensionnel dans le cadre du projet

Nous avons donné précédemment un ensemble de définitions requises pour la conception d'un modèle multidimensionnel. Dans cette partie, nous reprenons cet ensemble pour aboutir à la description d'un modèle multidimensionnel dans un contexte médical. D'abord, nous décrivons le schéma en constellation conçu ainsi que les hiérarchies définies pour le projet ADELEM et nous terminons cette partie par une description détaillée de l'ensemble des schémas qui composent ce modèle.

3.5.1 Schéma en constellation pour ADELEM

La figure 3.5 montre le schéma en constellation avec trois relations de faits et leurs dimensions. La relation de faits `Prise_MCO` (Médecine-Chirurgie-Obstétrique) a été conçue pour la gestion des séjours hospitaliers effectués dans la partie court séjour d'un établissement. La relation de faits `Population` manipule des informations démographiques. La troisième `Prise_SSR` (Soins de Suite et de Réadaptation) à différence de `Prise_MCO` enregistre des données correspondantes à des longs séjours.

Précedemment, nous avons défini une base de données multidimensionnelle, un cube, une dimension et une hiérarchie. Ainsi, en utilisant la figure 3.5, nous décrivons notre schéma en constellation de la manière suivante :

Base de données multidimensionnelle :

Une base de données multidimensionnelle est un n-uplet SM = (C_s, D_s, H_s, R), où

C_s = {Prise_MCO, Prise_SSR, Population}
D_s = {Etablissements, CIM10, Temps, Zone_geo, Age, Mode_sortie, Poids_naissance, RP99, Semaine_debut, Semaine_fin}
R = {C_Cube, C_Dimension, C_Hiérarchie}

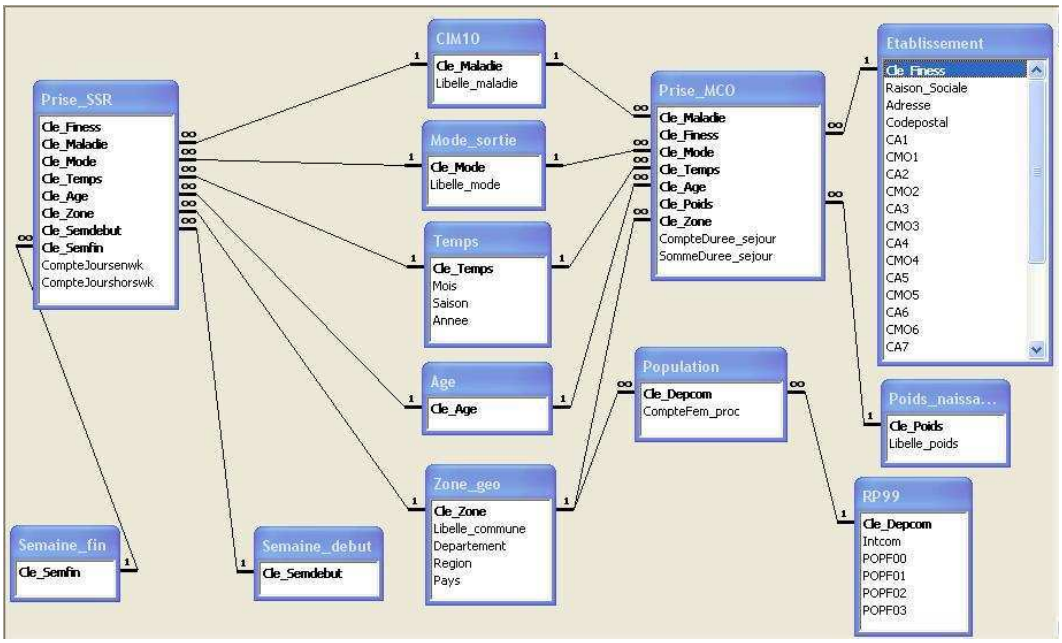

FIG. 3.5 – Schéma en Constellation pour ADELEM

contrainte :

C_Cube = le component **Clé_P** est composé de l'ensemble des Clé_P du component /**Dimension**.

C_Dimension = permet d'avoir une instance D avec un seul attribut, dimension dégénérée [Kim96].

C_Hiérarchie = la **Clé_P** d'une métaclasse **Hiérarchie** ne peut pas relier la métaclasse **Cube** associée à sa métaclasse **Dimension** et ceci en raison de sa granularité.

3.5.2 Hiérarchies du schéma

La figure 3.6 montre deux hiérarchies que nous avons conçues pour le schéma en constellation précédent [JLS99, WB97]. La première, que nous appelons H_Geo, à été définie pour les dimensions `Etablissement` et `Zone_geo` de la manière suivante.

Etant donné que le domaine de la dimension est : x ={C1,...,Cn}, représentant les communes du pays, nous définissons :

$x' \in \{D1, ..., Dn\}$ comme Départements
$x'' \in \{R1, ..., Rn\}$ comme Régions
$x''' \in \{P\}$ comme Pays.

La seconde hiérarchie, appelée H_Temps, correspond à la dimension `Temps`, dont le domaine est : x ={1, ..., 12}, représentant les mois de l'année, nous définissons :

$x' \in \{H, P, E, A\}$ comme les saisons (Hiver, Printemps, Eté, Automne)

$x'' \in \{A\}$ représentant l'année.

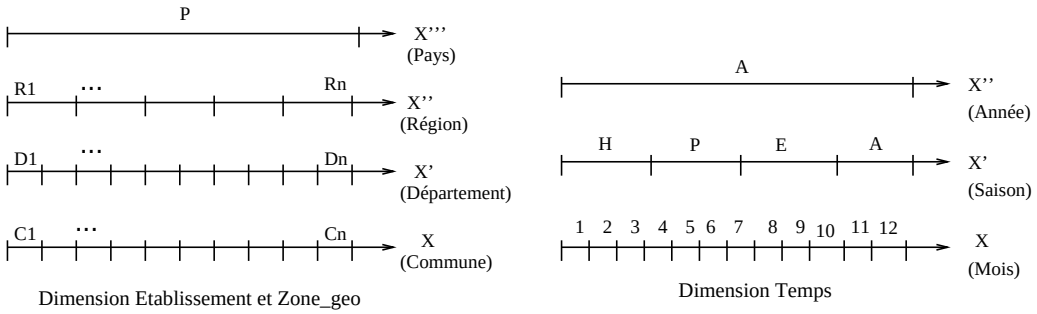

FIG. 3.6 – Hiérarchies de la dimension x

A l'intérieur de la hiérarchie `H_Geo`, l'élément `Commune` représente la donnée de granularité inférieure, tandis que `Pays` représente le niveau supérieur d'agrégation. Dans la hiérarchie `H_Temps`, la donnée de granularité inférieure est représentée par le `Mois` et celle de granularité supérieure est l'`Annee`. Par exemple, si N est définie comme une fonction f de x, y, z, dénoté par $N = f\ (x,y,z)$, donc le domaine de f est l'espace multidimensionnel construit par les dimensions (x,y,z), nous pouvons grouper N le long de la dimension x pour le niveau x', de la manière suivante :

$$N' = F(x', y, z) = \sum_{x \in \{x'\}} f(x, y, z)$$

L'équation précédente fait une agrégation de niveau x' à l'intérieur de la dimension x. Le résultat représente le nombre de patients par `Commune` groupés par `Departement`. Nous pouvons aussi faire des agrégations par rapport au nombre de patients le long de plusieurs dimensions.

3.5.3 Description du schéma `Prise_MCO` et de ses dimensions

Nous montrons la spécification de l'étoile : `Prise_MCO`. Pour faire cela, nous prenons la figure 3.5 et nous isolons la première étoile. La figure 3.7 représente un schéma composée de la relation de faits `Prise_MCO` et de ses dimensions. La saisie de l'information se fera à partir du fichier RSA (public2000.mdb et prive2000.mdb), du fichier FINESS (FINESSTOT.xls), du fichier CIM10 (libcim10-95.xls) et du fichier RP99 (DA99CCMC.xls). Dans cette relation, il y aura un enregistrement pour chaque séjour effectué dans la partie court séjour d'un établissement. La taille estimée pour la relation `Prise_MCO` est environ 15 à 18 millions d'enregistrements par année.

Un schéma de cube est un n-uplet C_s = (c_n, M, D), où

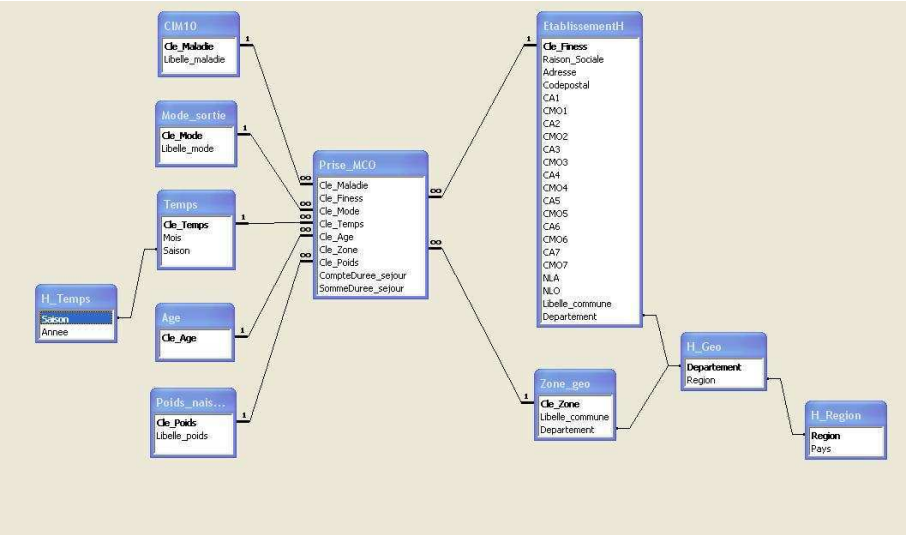

FIG. 3.7 – Schéma en flocon de neige `Prise_MCO`

c_n = Prise_MCO
M = {CompteDuree_sejour, SommeDuree_sejour}
D = {CIM10, Etablissement, Temps, Zone_geo, Poids_naissance, Age, Mode_sortie}

Un schéma de dimension est un n-uplet $D_s = (d_n, P, H)$, où

d_n = CIM10
P = {Cle_Maladie, Libelle_maladie}
H = {∅}

d_n = Etablissement
P = {Cle_Finess, Raison_sociale, Adresse, Codepostal, CA1..CA7, CMO1..CMO7, NLA, NLO, Libelle_commune, Departement, Region, Pays}
H = {H_Geo}

Les attributs CA1..CA7 et CMO1..CMO7 représentent le nombre de lits autorisés et le nombre de lits mis en oeuvre respectivement par discipline. Une discipline désigne une activité homogène qui est en fonction du type de soins ou de service social. Par exemple, médecine générale, pédiatrie, chirurgie, hébergement, accueil temporaire, éducation. Dans le domaine sanitaire, certaines autorisations sont faites par Grands Groupes de Disciplines GGDE qui sont des regroupements de disciplines.

d_n = Temps
P = {Cle_Temps, Mois, Saison, Annee}
H = {H_Temps}

Saison : Type char, en utilisant le code comme suit :

Hiver = janvier, février et mars
Printemps = avril, mai et juin
Été = juillet, août et septembre
Automne = octobre, novembre et décembre.
Pour arriver à faire des analyses saisonnières.

d_n = Zone_geo
P = {Cle_Zone, Libelle_commune, Departement, Region, Pays}
H = {H_Geo}

d_n = Poids_naissance
P = {Cle_Poids, Libelle_poids}
H = {$\emptyset$}

Dans cette relation nous avons pour le poids à la naissance les intervalles suivants :

1 <1500g
2 >=1500g et <2000g
3 >=2000g et <2500g
4 >=2500g et <3000g
5 >=3000g et <3500g
6 >3500g

d_n = Age
P = {Cle_Age}
H = {$\emptyset$}

d_n = Mode_sortie
P = {Cle_Mode, Libelle_mode}
H = {$\emptyset$}

Un schéma de hiérarchie est un n-uplet $H_s = (h_n, L, \prec)$, où

h_n = {H_Geo}
L = {Commune, Département, Région, $\top$}
$\prec$ = {(Commune, Département), (Département, Région), (Région, $\top$)}

h_n = {H_Temps}
L = {Mois, Saison, $\top$}
$\prec$ = {(Mois, Saison), (Saison, $\top$)}

3.5.4 Description du schéma `Population` et de ses dimensions

La saisie de l'information se fera à partir du fichier RP99 (DA99CCMC.xls) qui contient une pyramide des âges hommes et une autre pour les femmes. Néanmoins, si nous nous intéressons seulement aux prévisions des accouchements, nous pouvons stocker des données par rapport aux femmes ou bien, nous pouvons calculer et agréger les informations directement dans la relation Prévision. Celle-ci deviendra une sorte de relation d'agrégats et dans ce cas nous n'avons pas besoin de la dimension RP99. La figure 3.8 représente le schéma `Population`.

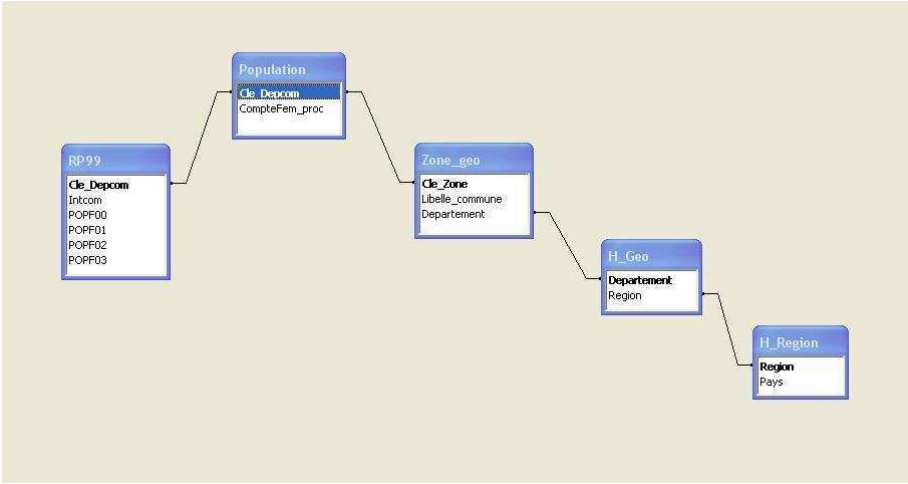

FIG. 3.8 – Schéma en flocon de neige `Population`

Un schéma de cube est un n-uplet $C_s = (c_n, M, D)$, où

c_n = Population
M = {CompteFem_proc}
D = {Zone_geo, RP99}

Un schéma de dimension est un n-uplet $D_s = (d_n, P, H)$, où

d_n = Zone_geo
P = {Cle_Zone, Libelle_commune, Departement, Region, Pays}
H = {H_Geo}

d_n = RP99
P = {Cle_Depcom, Intcom, POPF00 .. POPF>95}
H = {∅}

Un schéma de hiérarchie est un n-uplet $H_s = (h_n, L, \prec)$, où

h_n = {H_Geo}

L = {Commune, Département, Région, ⊤}
$\prec$ = {(Commune, Département), (Département, Région), (Région, ⊤)}

3.5.5 Description du schéma `Prise_SSR` et de ses dimensions

Dans cette relation, il y aura un enregistrement pour chaque séjour effectué dans la partie long séjour d'un établissement (Soins de Suite ou de Réadaptation). La taille estimée pour la relation `Prise_SSR` est d'environ 4 millions d'enregistrements par an correspondants à des séjours hebdomadaires. La figure 3.9 représente le troisième schéma `Prise_SSR`.

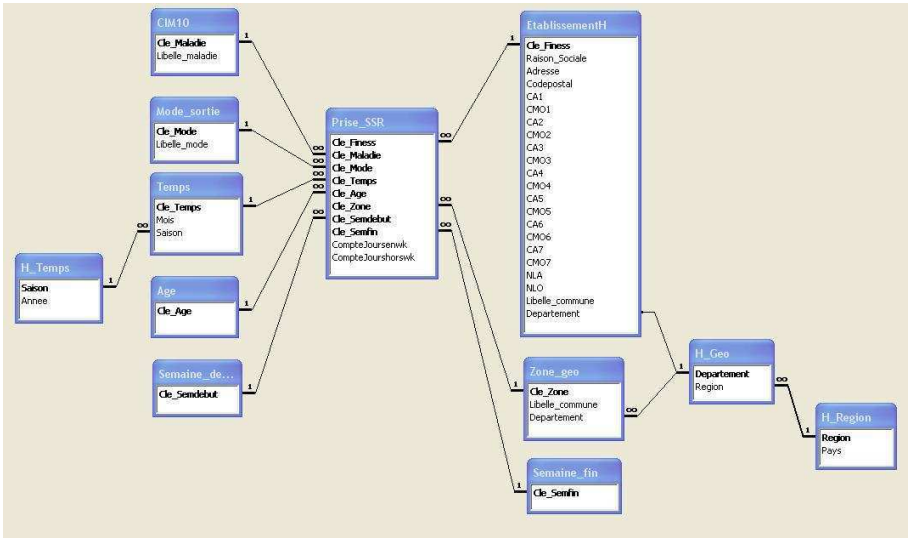

FIG. 3.9 – Schéma en flocon de neige `Prise_SSR`

Un schéma de cube est un n-uplet C_s = (c_n, M, D), où

c_n = Prise_SSR
M = {CompteJoursenwk, CompteJourshorswk}
D = {Etablissment, CIM10, Mode_sortie, Temps, Age, Zone_geo, Semaine_debut, Semaine_fin}

Un schéma de dimension est un n-uplet D_s = (d_n, P, H), où

d_n = Semaine_debut
P = {Cle_Semdebut}
H = {∅}

d_n = Semaine_fin
P = {Cle_Semfin}
H = {∅}

Les autres dimensions ont été traitées au paragraphe 3.5.3.

Un schéma de hiérarchie est un n-uplet $H_s = (h_n, L, \prec)$

A l'intérieur du cube Prise_SSR, nous avons les hiérarchies H_Geo et H_Temps, néanmoins, elles aussi ont été traitées au paragraphe 3.5.3.

3.6 Bilan

Dans ce chapitre, nous avons présenté la première partie de ce que nous appelons le processus de structuration. Nous avons conçu d'abord une architecture pour un système décisionnel basée sur trois composants, une interface graphique pour la génération semi-automatique des indicateurs, un entrepôt multidimensionnel qui rassemble les données internes et externes et les versions de schémas bitemporels.

Dans une deuxième phase, nous nous sommes focalisé sur le métamodèle et le modèle multidimensionnel. Pour le métamodèle, nous avons spécifié trois métaclasses : Cube, Dimension et Hiérarchie. Nous avons proposé un ensemble de définitions pour chaque métaclasse et leurs instances, ainsi qu'une définition d'une base de données multidimensionnelle et de son instance. Ceci nous a permis d'aboutir à la conception d'un schéma en constellation pour la construction d'un entrepôt multidimensionnel de données médicales. Nous avons utilisé les informations concernant l'ensemble des sources de données réelles et des indicateurs du projet ADELEM pour prouver et vérifier le modèle proposé.

Pendant la première étape de notre expérimentation, nous avons analysé les sources de données. Ceci nous a permis d'identifier deux types de données, les données démographiques (RP99) et les données publiques concernant la santé (RSA, RHA, CIM10 et FINESS). Principalement, dans ces dernières, nous avons constaté le besoin d'offrir des mécanismes pour une manipulation efficace de l'ensemble d'information. Pour le projet, nous avons conçu un tableau d'analyse qui regroupe les données sur dix années. Ce tableau nous a permis d'identifier les sources qui ont besoin d'utiliser des techniques d'optimisation de stockage comme la création des agrégats.

L'analyse des indicateurs établis pour le projet nous a permis de les classer en trois types. Les deux premiers rassemblent les identificateurs d'offre "géographiques" et les identificateurs de consommation, de besoins et de flux "temporels". Le troisième type correspond à des nouveaux indicateurs également "temporels". Nous les avons identifiés comme des indicateurs de consommations et de flux. Ils nous permettent de faire des analyses par rapport à l'âge du patient ainsi qu'à son mode de sortie de l'hôpital.

Dans la dernière partie, nous nous sommes focalisés sur la conception d'un modèle

multidimensionnel pour le projet. Nous avons conçu une constellation composée de trois schémas : `Prise_MCO, Prise_SSR` et `Population`, où chacun est composé de ses propres dimensions ainsi que des dimensions partagées pour l'ensemble de schémas. Nous avons utilisé les définitions proposées pour aboutir à la description des schémas.

Nous avons dit précédemment que l'ensemble des sources comporte aussi bien des données réparties et hétérogènes qu'externes au domaine médical. Ainsi, nous avons d'un côté des problématiques liées à la nature particulière des données médicales : type, format, sémantique, confidentialité, degré de fiabilité et de confiance, informations manquantes ou incomplètes, et d'un autre côté l'hétérogénéité des données fournies par des organismes hors du contexte médical. Par exemple : les renseignements sur l'origine géographique des patients pour la consommation de soins sont basés sur les codes postaux et non pas sur le code INSEE de la commune de résidence du patient.

Dans le chapitre suivant, nous décrivons la deuxième partie du processus de structuration. Elle consiste en la sélection de l'ensemble optimal de vues à matérialiser. Nous présentons aussi le fonctionnement de l'interface graphique conçue pour la génération semi-automatique de requêtes à partir des indicateurs.

Chapitre 4

Système d'aide à la décision médicale : une expérimentation

Dans le chapitre précédent, nous avons décrit le modèle multidimensionnel conçu pour le projet ADELEM. Tout au long de ce chapitre, nous focalisons notre expérimentation sur un système décisionnel.

Nous avons divisé le travail réalisé en trois parties. La première décrit le schéma en étoile construit et la matérialisation de l'hypercube pour celui-ci. La deuxième partie traite le sujet des vues matérialisées qui nous permettent d'optimiser l'exécution de requêtes. Nous avons construit le schéma `Adelem_MCO` et nous avons crée la base de données correspondante en utilisant un échantillon du 10% des données réelles du projet. Nous avons utilisé le système décisionnel d'Oracle9$_i$ Entreprise Edition Release 9.2.0.1.0 pour la création du schéma et pour la génération des vues matérialisées. Ceci nous a permis de connaître le coût de stockage (représenté par le nombre de n-uplets du résultat) de chaque vue et de pouvoir facilement déterminer leur coût de calcul (produit des cardinalités approximatives des relations de base). Nous proposons un algorithme pour la sélection de l'ensemble optimal des vues à sélectionner qui utilise comme paramètres la fréquence d'utilisation, le coût de calcul et la probabilité de changement des relations de base. Nous terminons cette partie avec les relations de dépendance et la création des vues à matérialiser de cet ensemble optimal.

Finalement, la troisième partie se focalise sur le fonctionnement de l'interface pour la génération semi-automatique d'indicateurs. Nous donnons d'abord l'architecture pour cette interface et la description de ses composants. L'utilisation de l'interface requiert un module pour la création du schéma, ainsi, nous décrivons d'abord le fonctionnement de ce module, ensuite, nous classifions les types de requête à exécuter et finalement, nous décrivons le fonctionnement de l'interface graphique en utilisant un exemple de requête.

4.1 Construction du schéma

Dans cette section, nous décrivons la construction du schéma en étoile `Adelem_MCO`. Il est composé d'une relation de faits et de quatre relations de dimensions. Nous définissons d'abord la structure de chaque relation qui intègre le schéma et la matérialisation de l'hypercube.

4.1.1 Schéma en étoile `Adelem_MCO`

Pour la construction du schéma, nous avons seulement utilisé une partie de l'étoile `Prise_MCO` du schéma en constellation conçu pour le projet (*cf.* Figure 3.5). La figure 4.1 montre le schéma `Adelem_MCO` construit. Il est composé de la relation de faits `Prise_MCO` et des dimensions `Etablissement, CIM10, Temps` et `Mode_sortie`.

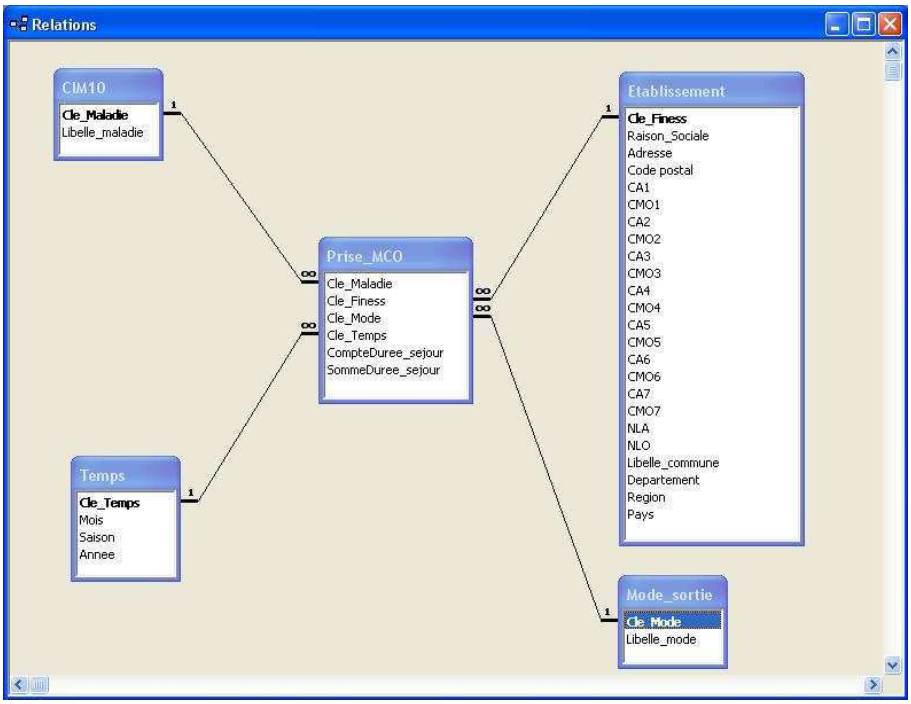

FIG. 4.1 – Schéma en étoile `Adelem_MCO`

Nous décrivons la structure de chaque relation qui compose le schéma :

Fait :

```
Prise_MCO = {Cle_Finess, Cle_Maladie, Cle_Temps, Cle_Mode,
   CompteDuree_sejour, SommeDuree_sejour}
```

Dimensions :

```
Etablissement = {Cle_Finess, Raison_sociale, Adresse, Code_postal, CA1,
```

```
  CMO1, CA2, CMO2, CA3, CMO3, CA4, CMO4, CA5, CMO5, CA6, CMO6, CA7,
  CMO7, NLA, NLO, Libelle_commune, Departement, Region, Pays}

CIM10 = {Cle_Maladie, Libelle_maladie}

Temps = {Cle_Temps, Mois, Saison, Annee}

Mode_sortie = {Cle_Mode, Libelle_mode}
```

Le tableau 4.1 contient le type et le nombre d'enregistrements de chaque relation.

Relation	**Fait/Dimension**	**Taille**
Prise_MCO	Fait	53799 n-uplets
Etablissement	Dimension	5079
CIM10	Dimension	17788
Temps	Dimension	12
Mode_sortie	Dimension	5

TAB. 4.1 – Liste de relations de base

4.1.2 Matérialisation de l'Hypercube

Les utilisateurs des entrepôts travaillent dans un environnement graphique et ils visualisent les données comme un cube de 2, 3 ou plusieurs dimensions. Nous utilisons le schéma en étoile de la figure 4.1 pour construire un hypercube. Chaque cellule (E, C, T, M, m1, m2) contient le nombre de séjours (m1) et leur somme en jours (m2) qui ont eu lieu dans l'Hôpital E, avec la maladie C, pendant le mois T et avec un mode de sortie M (par exemple : le patient rentre chez lui).

Dans les systèmes décisionnels, les utilisateurs sont intéressés par des requêtes du type : "le nombre de séjours de l'hôpital E1 pour la maladie C1". Dans ce cas, la cellule (E1, C1, ALL[1], ALL, m1), contient la valeur : "le nombre de séjours de l'hôpital E1 pour la maladie C1 pour tous les mois (ALL) et pour tous les modes de sortie (ALL)".

Nous pouvons calculer la valeur de la cellule (E1, C1, ALL, ALL) comme la somme des valeurs des cellules de (E1, C1, T_1, M_1), ..., (E1, C1, T_{Ntemps}, M_{Nmode}), où T_{Ntemps} et M_{Nmode} représentent l'ensemble de mois et l'ensemble de modes de sortie respectivement. Toutes les cellules contenant la valeur ALL dans un de ses axes sont des cellules dépendantes. Ainsi, pour la sélection des vues à matérialiser, le

[1]Nous utilisons la recommandation d'ajouter la valeur ALL au domaine de la dimension T et M, présentée dans [GBLP95]

problème se réduit à déterminer l'ensemble des cellules dépendantes à matérialiser. Etant donné que le coût de stockage [2] pour une matérialisation de toutes les vues est de 205929 n-uplets ($\sum_{i=1,\ldots,n}$CS(v_i), où CS(v_i) est le coût de stockage des vues du tableau 4.2), nous avons un pourcentage de 74% ((206781-53799)*100/206781) de cellules dépendantes pour le schéma `Prise_MCO` construit.

Pour la matérialisation de l'hypercube, nous avons les possibilités suivantes : la matérialisation complète, pas de matérialisation et la matérialisation partielle. Dans notre expérimentation, nous considérons la dernière approche. Nous définissons d'abord la taille de la matérialisation complète d'un cube qui est composé de 5 relations de la manière suivante :

Taille du Cube

Fait = 1
Dimensions = 4
Total = 16 (2^n où n est le nombre de dimensions = 2^4)

Nous définissons l'ensemble des vues possibles à matérialiser. Le tableau 4.2 contient les 15 vues pour une matérialisation complète ainsi que leur coût de stockage et de calcul. La figure 4.2 représente l'hypercube sur 4 dimensions du schéma `Prise_MCO`, nous utilisons la notation : M pour million et K pour mille.

Vue	Relations	C. de Stockage	C. de Calcul
V1	Prise_MCO+Etablissement+CIM10+Temps+Mode_sortie	53799 n-uplets	90M
V2	Prise_MCO+Etablissement+CIM10+Temps	47133	90M
V3	Prise_MCO+Etablissement+CIM10+Mode_sortie	18456	90M
V4	Prise_MCO+Etablissement+Temps+Mode_sortie	603	61K
V5	Prise_MCO+CIM10+Temps+Mode_sortie	32918	346K
V6	Prise_MCO+Etablissement+CIM10	13973	90M
V7	Prise_MCO+Etablissement+Temps	184	60K
V8	Prise_MCO+Etablissement+Mode_sortie	58	25K
V9	Prise_MCO+CIM10+Temps	26058	216K
V10	Prise_MCO+CIM10+Mode_sortie	8186	90K
V11	Prise_MCO+Temps+Mode_sortie	48	60
V12	Prise_MCO+Etablissement	19	5K
V13	Prise_MCO+CIM10	5329	18K
V14	Prise_MCO+Temps	12	12
V15	Prise_MCO+Mode_sortie	4	5

TAB. 4.2 – Matérialisation complète de l'hypercube

La vue V1 représente les données de détail du sous-schéma `Prise_MCO`. Le coût de stockage (représenté par le nombre de n-uplets du résultat) est d'environ 54K et le coût de calcul (produit des cardinalités approximatives des relations de base) est de

[2] Le coût de stockage d'une requête est le nombre de n-uplets du résultat.

90M. En ce qui concerne le coût de calcul, nous le calculons de la manière suivante :

CS(Etablissement)*CS(CIM10) : 5K*18K = 90M, où CS représente le coût de stockage (*cf.* Tableau 4.1)
+
CS(V6)*CS(Temps) : 14K*12 = 168K
+
CS(V2)*CS(Mode_sortie) : 47K*5 = 235K
(90M + 168K + 235K) ≈ 90M

La figure 4.2 représente le treillis pour l'hypercube et elle contient le coût de stockage à droite de chaque noeud (vue) et le coût de calcul à gauche.

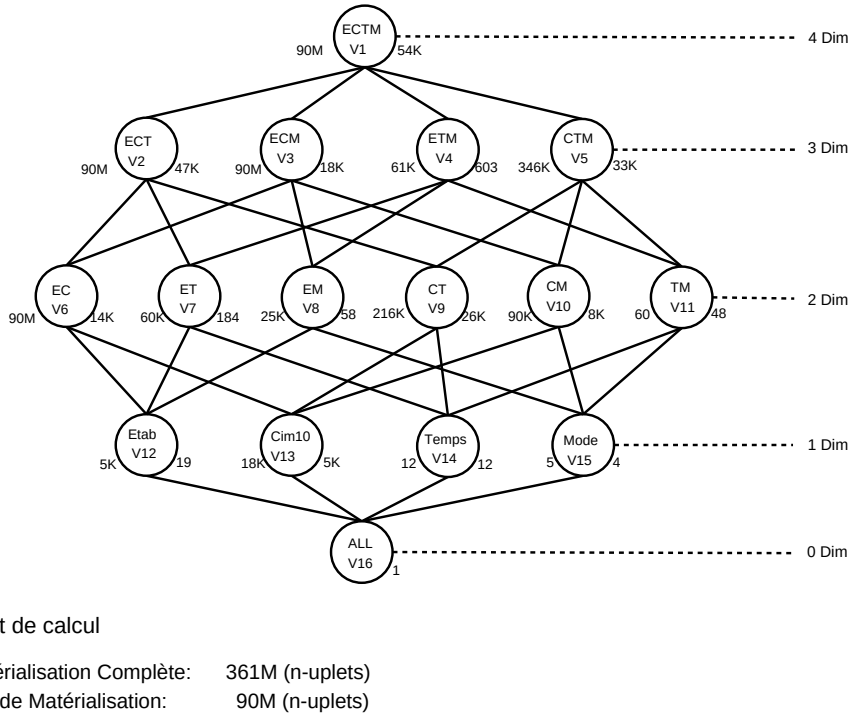

FIG. 4.2 – Hypercube avec le coût de calcul (à gauche) et le coût de stockage (à droite) de chaque noeud (vue)

4.2 Vues matérialisées

Dans cette section, nous nous focalisons sur la génération des vues matérialisées. Nous appliquons deux algorithmes aux données du projet ADELEM. Le premier est l'algorithme Greedy, l'autre est un algorithme que nous proposons pour la sélection des vues à matérialiser.

L'algorithme Greedy proposé par [HRU95, HRU96] repose sur un modèle de coût pour déterminer l'ensemble optimal de vues à matérialiser. Il utilise le coût de sto-

ckage et le nombre de vues dépendantes (optimales) d'une vue pour calculer le bénéfice optimal de celle-ci. Nous rappelons que une vue V_i est dépendante de V_j si, et seulement si, nous pouvons répondre V_i en utilisant V_j, ainsi, $V_i \preceq V_j$.

Nous présentons d'abord l'application de l'algorithme Greedy aux données réelles du projet. Pour faire cela, nous listons l'ensemble des vues dépendantes de chaque vue de l'hypercube. Ensuite, nous exposons le fonctionnement de l'algorithme, et pour finir, nous présentons le tableau résultant qui décrit le choix des 7 premières vues à matérialiser. Dans une deuxième partie, nous présentons le fonctionnement de notre algorithme, ainsi que le tableau pour une sélection aussi des 7 premières vues à matérialiser.

4.2.1 Sélection des vues à matérialiser

Nous formalisons la sélection des vues à matérialiser en utilisant l'algorithme Greedy. Nous avons décidé d'utiliser cet algorithme pour deux raisons, la première parce que nous avons toutes les données nécessaires pour leur utilisation. Ainsi, nous n'avons pas besoin de faire des hypothèses, principalement, pour le coût de stockage, car nous avons le coût de stockage réel de chaque vue possible d'être matérialisée. La seconde raison, à notre avis la plus importante, est l'idée que nous avons eue de réutiliser le nombre de vues dépendantes comme un paramètre initial pour la fréquence d'utilisation d'une vue.

Nous avons remarqué que plusieurs propositions utilisent comme fréquence d'utilisation un nombre assigné à chaque vue, par exemple : la fréquence d'utilisation pour la vue V1 est égale à 10, V2 à 5, V3 à 1, ... Ainsi, la particularité de notre algorithme réside dans la spécification de deux propositions. La première consiste à utiliser la complexité de la vue (nombre total de ses relations dépendantes) pour les deux premiers paramètres : la fréquence d'utilisation et le coût de calcul de la vue. Ainsi, nous considérons que la fréquence d'utilisation d'une vue augmente en proportion directe du nombre de ses relations dépendantes et que le coût de calcul diminue dans la même proportion. La deuxième proposition consiste à déterminer la probabilité de changement sur les relations de base. Pour faire cela, nous avons besoin du nombre d'éléments à l'intérieur de la vue qui peuvent subir des changements et nous le multiplions par le coût de calcul de celle-ci.

Nous utilisons l'hypercube de la fig 4.2 et nous considérons que le coût de stockage est représenté par le nombre de n-uplets de la vue. Ainsi, nous pouvons déterminer l'ensemble de vues dépendantes de chaque vue de la manière suivante.

Relations de dépendance à l'intérieur de l'hypercube :

V2 : V6 $\preceq$ V2, V7 $\preceq$ V2, V9 $\preceq$ V2, V12 $\preceq$ V2, V13 $\preceq$ V2, V14 $\preceq$ V2, V16 $\preceq$ V2

V3 : V6 $\preceq$ V3, V8 $\preceq$ V3, V10 $\preceq$ V3, V12 $\preceq$ V3, V13 $\preceq$ V3, V15 $\preceq$ V3, V16 $\preceq$ V3

V4 : V7 $\preceq$ V4, V8 $\preceq$ V4, V11 $\preceq$ V4, V12 $\preceq$ V4, V14 $\preceq$ V4, V15 $\preceq$ V4, V16 $\preceq$ V4

V5 : V9 $\preceq$ V5, V10 $\preceq$ V5, V11 $\preceq$ V5, V13 $\preceq$ V5, V12 $\preceq$ V5, V15 $\preceq$ V5, V16 $\preceq$ V5

V6 : V12 $\preceq$ V6, V13 $\preceq$ V6, V16 $\preceq$ v6

V7 : V12 $\preceq$ V7, V14 $\preceq$ V7, V16 $\preceq$ V7

V8 : V12 $\preceq$ V8, V15 $\preceq$ V8, V16 $\preceq$ V8

V9 : V13 $\preceq$ V9, V14 $\preceq$ V9, V16 $\preceq$ V9

V10 : V13 $\preceq$ V10, V15 $\preceq$ V10, V16 $\preceq$ V10

V11 : V14 $\preceq$ V11, V15 $\preceq$ V11, V16 $\preceq$ V11

V12 : V16 $\preceq$ V12

V13 : V16 $\preceq$ V13

V14 : V16 $\preceq$ V14

V15 : V16 $\preceq$ V15

Quelques notations de l'algorithme Greedy :

C(v) = Coût de stockage de la vue v.
$\preceq$ = Relation de dépendance. Par exemple, Q2 $\preceq$ Q1 si et seulement si Q2 peut être répondue par Q1, donc Q2 est dépendante de Q1.
S = Ensemble de vues sélectionnées.
B(v, S) = Gain de la vue v relative à S,
1) Pour chaque $w \preceq v$, définir la quantité B_w par :
a) Si u est la vue de coût inférieur dans S, tel que $w \preceq u$. Nous remarquons que l'ensemble S contient au moins la vue V1.
b) Si C(v) < C(u), alors B_w = C(v) - C(u). Sinon $B_w = 0$.
2) Définir B(v, S) = $\sum_{w \preceq v} B_w$.

La figure 4.3 décrit l'algorithme Greedy pour une sélection de n vues à matérialiser.

Nous utilisons l'hypercube de la fig 4.2 pour l'application de l'algorithme. La table 4.3 décrit les résultats avec n égal à 7.

```
S = {top view};
for i = 1 to n do begin
    select that view v not in S such that
        B(v, S) is maximized;
    S = S union {v};
end;
resulting S in the greedy selection;
```

FIG. 4.3 – Algorithme Greedy [HRU95]

Vue	**1ère Choix**	**2 Choix**	**3 Choix**	**4 Choix**	**5 Choix**	**6 Choix**	**7 Choix**
V2	7K*8=56K	7K*4=28K	7K*2=14K	7K*1=7K	7K*1=7K	7K*1=7K	-
V3	36K*8=288K	36K*4=144K	-	-	-	-	-
V4	53K*8=424K	-	-	-	-	-	-
V5	21K*8=168K	21K*4=84K	21K*2=42K	-	-	-	-
V6	40K*4=160K	40K*2=80K	4K*2=8K	4K*2=8K	4K*1=4K	4K*1=4K	4K*1=4K
V7	54K*4=216K	419*4=2K	419*4=2K	419*4=2K	419*4=2K	419*4=2K	419*4=2K
V8	54K*4=216K	545*4=2K	545*4=2K	545*4=2K	545*4=2K	545*4=2K	545*4=2K
V9	28K*4=112K	28K*2=56K	28K*1=28K	7K*1=7K	7K*1=7K	-	-
V10	46K*4=184K	46K*2=92K	10K*2=20K	10K*2=20K	-	-	-
V11	54K*4=216K	555*4=2K	555*4=2K	555*4=2K	555*4=2K	555*4=2K	555*4=2K
V12	54K*2=108K	584*2=1K	584*2=1K	584*2=1K	584*2=1K	584*2=1K	584*2=1K
V13	49K*2=98K	49K*1=49K	13K*1=13K	13K*1=13K	3K	3K	3K
V14	54K*2=108K	591*2=1K	591*2=1K	591*2=1K	591*2=1K	591*2=1K	591*2=1K
V15	54K*2=108K	599*2=1K	599*2=1K	599*2=1K	599*2=1K	599*2=1K	599*2=1K
V16	54K*1=54K	1K	1K	1K	1K	1K	1K
{V1}	S=S+V4	S=S+V3	S=S+V5	S=S+V10	S=S+V9	S=S+V2	S=S+V6

TAB. 4.3 – Application de l'algorithme Greedy aux données ADELEM

Pour la première itération, le fonctionnement de l'algorithme Greedy est assez simple, car nous avons seulement la vue V1 dans l'ensemble S ; de cette manière la valeur de u est égal à 54K (le coût de V1) car elle représente la vue avec le coût de stockage minimum dans l'ensemble S. Ainsi, pour obtenir le bénéfice optimal de V2, nous devons calculer la différence du coût de stockage de V2 par rapport à V1 (47K - 54K) et nous multiplions cette différence par le nombre de relations dépendantes de V2 (V2, V6, V7, V9, V12, V13, V14 et V16), ce qui nous donne 56K de bénéfice optimal pour V2. Pour V3, le mécanisme est pareil, ainsi nous avons 288K. Dans le cas de la vue V6, nous multiplions la différence du coût de stockage entre V6 et V1 par les vues dépendantes de V6 (V6, V12, V13 et V16) et ainsi de suite.

Une fois calculé le bénéfice optimal de l'ensemble de vues, nous devons choisir celle qui ait le bénéfice maximum. Dans la première itération, le choix est la vue V4, car elle représente le gain le plus élevé (53K*8=424K), le gain est calculé de la manière suivante : coût(V4) - coût(V1) = 53K multiplié par 8, le nombre de vues dépendantes de V4 (V4, V7, V8, V11, V12, V14, V15 et V16).

Néanmoins, au fur et mesure que nous remplissons l'ensemble S, nous devons toujours choisir le coût de stockage minimum de u à l'intérieur de S duquel la vue dépende. Ainsi, dans le second choix, pour certains vues, u est représentée par 54K (le coût de stockage de V1), tandis que pour les vues qui dépendent de V4 (qui a été choisie dans la première itération), u est 603. Par exemple, le bénéfice optimal de V2 est 28K, car la différence entre les coûts de stockage de V2 et V1 (toujours 7K) est multiplié par 4 (V2, V6, V9 et V13). Les autres vues dépendantes de V2 (V7, V12, V14 et V16) ne sont pas prises en compte, car elles donnent un gain plus élevé avec la vue V4.

Dans la seconde choix, la sélection porte sur la vue V3 avec 144K (36K*4 (V3, V6, V10 et V13)), les autres vues dépendantes de V3 ne sont pas prises en compte, car elles donnent un gain plus élevé avec la vue V4. Finalement, la dernière ligne contient le résultat, elle représente l'ensemble S={V4, V3, V5, V10, V9, V2, V6}.

La figure 4.4 montre l'application de l'algorithme Greedy par rapport aux coûts de calcul et de stockage. Nous considérons seulement les 7 premiers résultats, même si à partir de la 6ème itération, nous n'avons aucun avantage pour la matérialisation. Par exemple, la vue V2, résultat de la 6ème itération, a pour coût de stockage 47K et un coût de calcul de 90M. Si nous faisons la comparaison de ces résultats, par rapport à la vue V1, que nous sommes censés matérialiser[3], nous nous apercevons que le gain est presque minimum, mais que les coûts de leur matérialisation sont doublés.

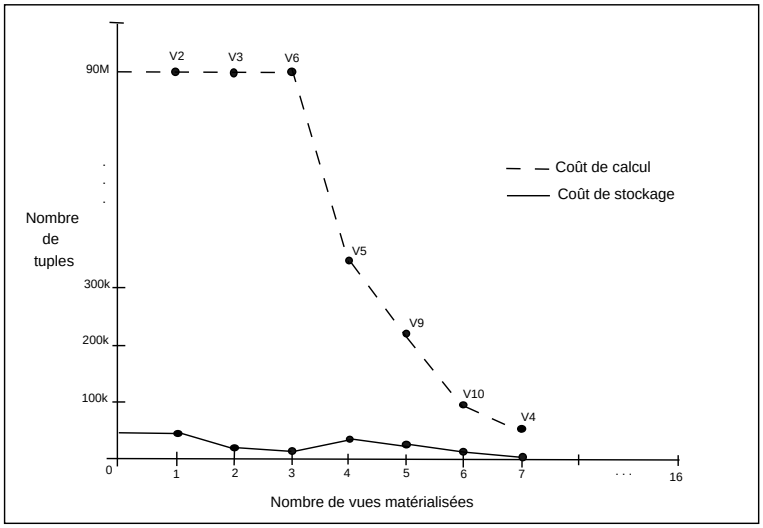

FIG. 4.4 – Ensemble ordonné des vues sélectionnées

La figure 4.4 contient les coûts de calcul et de stockage dans l'axe Y et les sept premières vues de la table 4.3 ordonnées (par rapport à leur coût de calcul) sur l'axe X.

[3]Car cette vue ne peut pas être construit à partir d'aucune autre vue.

L'algorithme Greedy est optimal dans les cas suivants :

1. Si V1 est plus grande que les autres vues, alors l'algorithme est proche de l'optimal.
2. Si toutes les vues sont égales, alors l'algorithme est optimal.

Si nous considérons le coût de stockage que nous avons dans l'hypercube de la figure 4.2, nous nous apercevons que notre cas est similaire au premier, car la seule vue proche de la vue V1 est la vue V5. Le reste des vues est plus petit que V1.

La sélection des vues à matérialiser est un sujet de recherche très important. Il existe de nombreux travaux de recherche [AAD+96, BB03, BPT97, CHS02, CLF99, GM95, Gup97, KMP02, KR99, LMSS95, SDJL96, YKL97, ZYY01, YW00] qui utilisent un modèle de coût, comme celui de Greedy, mais qui a été adapté avec l'inclusion de certains paramètres, tels que : la fréquence de la requête, la fréquence des mises à jour sur les relations de base, le coût de maintenance, entre autres.

Par exemple, dans [BB03], les auteurs proposent un algorithme qui permet de calculer le gain global de la vue par rapport au coût de la requête et le coût de maintenance. Ils différencient le coût d'évaluation d'une requête par rapport au type d'opération (*Select, Project* et *Join*). Ils utilisent aussi la fréquence de la requête ainsi que le coût de maintenance basé sur les opérateurs *Add* et *Delete*.

L'algorithme Greedy est simple et efficace, néanmoins, dans notre expérimentation, à partir du 6ème choix, il sélectionne les vues les plus coûteuses. Ceci nous motive pour proposer un mécanisme pour une sélection plus fiable. Nous proposons d'inclure les paramètres suivants : fréquence d'utilisation, coût de calcul et probabilité de changement des relations de base (coût).

4.2.2 Algorithme proposé pour la sélection des vues à matérialiser

Dans notre algorithme, les relations dépendantes d'une vue jouent un rôle essentiel, ainsi, nous considérons que leur nombre représente un paramètre initial pour la fréquence d'utilisation, ainsi que pour le coût de calcul. Pour le premier, nous multiplions le bénéfice optimal, donné par l'algorithme Greedy, par le nombre total des relations dépendantes, car pour nous, la fréquence d'utilisation augmente en proportion directe du nombre de relations dépendantes. Pour le coût de calcul, nous considérons qu'il diminue par rapport au nombre de relations dépendantes, ainsi, nous divisons le coût de calcul par le nombre total des relations dépendantes [SA05a].

Notre algorithme considère aussi la probabilité de changement des relations de base. Dans ce cas, nous avons besoin du nombre d'éléments à l'intérieur de la vue qui

peuvent subir des changements. Considérons par exemple la vue V2. Elle se compose des relations Etablissement, CIM10 et Temps. La relation Etablissement contient 24 propriétés, CIM10 et Mode_sortie contiennent 2 propriétés et Temps en contient 4 (*cf.* paragraphe 4.1.1). Nous avons 36 éléments pouvant évoluer (32 propriétés et 4 dimensions). Supposons que nous avons une probabilité de changement de 20%, ainsi, notre calcul pour la fréquence des mises à jour de V2 est (33*100/36*.20), où 33 est le nombre d'éléments de la vue V2.

Quelques notations :

CC = Coût de calcul (produit des cardinalités approximatives des relations de base) divisé par le nombre de relations dépendantes. Par exemple, le CC(v) si v=V2 (vue ECT, *cf.* Figure 4.4) est : CC(V2) = ((5K*18K) + (14K*12))/8 ≈ 90M/8 = 11M, où 8 est le nombre total des relations dépendantes de V2.
PC = Probabilité de changement des relations de base multiplié par le coût de calcul. Par exemple, si nous avons l'hypothèse de 20% de changement des éléments du schéma sur la vue V2 (3 dimensions et 30 attributs qui peuvent changer), alors PC(V2) = (3300/36*.20)*11M = 2M.
V = Ensemble de vues.
S = Ensemble de vues sélectionnées.
w = Nombre de choix.
fq(v) = Complexité de la vue (nombre total des relations dépendantes de la vue v).
v = Vue sélectionnée.
vo = C(vue ⊤ "*top view*").

La figure 4.5 montre l'algorithme proposé incluant la fréquence d'utilisation, le coût de calcul et la probabilité de changement.

Le tableau 4.4 montre les résultats de l'application de notre algorithme pour les 7 premiers choix, en considérant 20% comme fréquence des mises à jour sur les relations de base.

Pour le premier choix la valeur de *vo* est égale à 54K, le coût de V1, car elle représente la vue avec le coût de stockage minimum dans l'ensemble S. Pour le deuxième et le troisième elle est représentée par 603 (le coût de V4), car c'est la vue ayant un coût de stockage inférieur à l'intérieur de S. Dans la première itération, le choix est la vue V4, car elle représente le plus haut gain ((((53K*8)*8)=3392K)-((61K/8)*1.18=9K)=3M). Le gain est calculé de la manière suivante : coût(V4) - coût(V1) = (53K*8) représente le gain de Greedy, nous multiplions ce gain par le nombre de relations dépendantes de V4 (V4, V7, V8, V11, V12, V14, V15 et V16). Ensuite, nous le soustrayons le coût de calcul divisé par le nombre de relations dépendantes et multiplié par la fréquence des mises à jour.

Le second choix est la vue V5 avec 626K (((21K*4*8)=672K) - ((346K/8) *

```
S = {vue T}; vo = C(vue T); fq = 1; Bg = 0; t = 0; n = |V|;
for y = 1 to w do begin                          //nombre de choix
    for x = 2 to n do begin                      //parcours de l'hypercube pour chaque vue
        i = x + 1;
        repeat                                   //calculer la fréquence et le bénéfice de l'algorithme Greedy
            if vi dépend de vx
               if {vx} appartient à S
                  vo=C(vx)
                  else vo=C(vue T)
               endif
               if vi dépend de v tel que {v} appartient à S
                  vo=C(v)
               endif
               fq = fq + 1;
               if C(vx) < vo
                  B(vi) = C(vx) - vo
                  else B(vi) = 0
               endif
               Bg = Bg + B(vi)                   //bénéfice Greedy
            endif
            i = ++;
        until i = n;
        Bg = Bg + (C(vx) - vo);                  //bénéfice de vx
        B(vx, S) = (Bg * fq) - (CC(vx)/fq + PC(vx));   //bénéfice incluant la fréquence, le coût de calcul et la prob. chang.
        if B(vx, S) > t
          t = B(vx, S);                          //bénéfice maximal
          v = vx;
        endif
    endfor
    t=0;
    S = S U {v};
endfor
résultat S
```

FIG. 4.5 – Algorithme proposé

Vue	**1ère Choix**	**2 Choix**	**3 Choix**	**4 Choix**	**5 Choix**	**6 Choix**	**7 Choix**
V2	(448K-(11M*1.18))=-13M	-13M	-13M	-13M	-13M	-13M	-13M
V3	(2M-13M)=-11M	-12M	-12M	-12M	-12M	-12M	-12M
V4	(3392K-9K)=3M	-	-	-	-	-	-
V5	(1344K-46K)=1298K	626K	-	-	-	-	-
V6	(640K-26M)=-25M	-26M	-26M	-26M	-26M	-26M	-26M
V7	(864K-18K)=845K	-11K	-11K	-11K	-11K	-11K	-11K
V8	(864K-7K)=857K	2K	2K	2K	2K	-	-
V9	(448K-56K)=392K	168K	0K	-28K	-28K	-28K	-28K
V10	(736K-24K)=713K	345K	177K	-	-	-	-
V11	(864K-62)=864K	9K	9K	9K	-	-	-
V12	(216K-3K)=213K	-1K	-1K	-1K	-1K	-3K	-3K
V13	(196-9K)=187K	89K	47K	-3K	-3K	-3K	-3K
V14	(54K*2*2=216K)=216K	2K	2K	2K	30	30	30
V15	(54K*2*2=216K)=216K	2K	2K	2K	41	41	-
S=S+V1	S=S+V4	S=S+V5	S=S+V10	S=S+V11	S=S+V8	S=S+V15	S=S+V14

TAB. 4.4 – Application de notre algorithme aux données ADELEM

1.06=46K) = 626K), les vues dépendantes sont (V5, V9, V10 et V13), les autres vues dépendantes de V5 ne sont pas prises en compte, car elles ont un gain inférieur à celui de la vue V4. Néanmoins, nous multiplions le résultat par le nombre total des vues dépendantes et nous divisons le coût de calcul par ce nombre total. Finalement, la dernière ligne contient le résultat, elle représente l'ensemble S={V4, V5, V10, V11, V8, V15, V14}.

La figure 4.6 montre l'application de notre algorithme par rapport aux coûts de calcul et de stockage. Nous considérons les 7 premiers résultats. Si nous prenons comme référence la figure 4.2, nous remarquons que la sélection est faite de la manière suivante : du deuxième niveau (3 Dim), notre algorithme sélectionne les deux vues de gain optimal. Pour le troisième niveau (2 Dim), il sélectionne les trois meilleures et finalement pour le dernier niveau (1 Dim), il sélectionne les deux premières vues optimales.

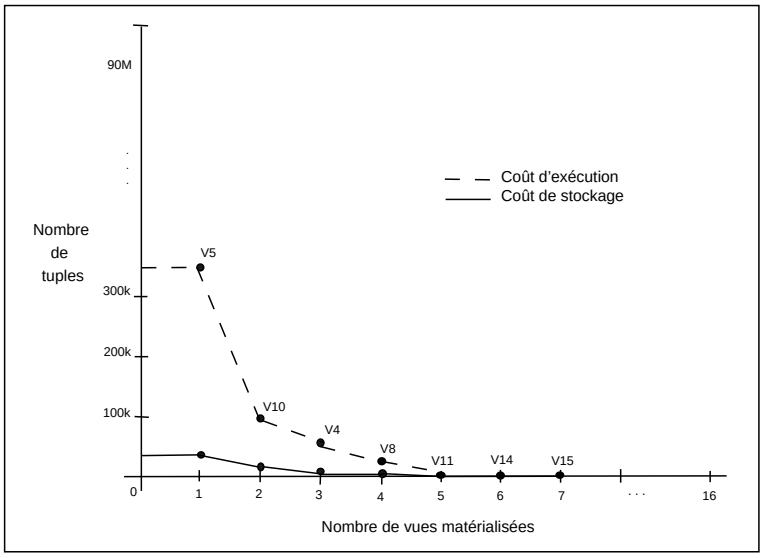

FIG. 4.6 – Ensemble ordonné des vues sélectionnées

Le graphique contient les coûts de calcul et de stockage sur l'axe des Y et les sept premières vues de la table 4.4 ordonnées (par rapport à leur coût de calcul) sur l'axe X. Nous constatons, qu'à la différence de l'algorithme Greedy, notre proposition donne de meilleurs résultats dans notre cas expérimental.

Finalement, nous devons reconnaître la faiblesse de notre algorithme. Nous avons deux paramètres que nous n'avons pas considéré, le premier est l'absence du coût d'évaluation d'une requête par rapport au type d'opération (*Select, Project* ou *Join*). Le second paramètre que nous ne considérons pas sont des restrictions, par exemple la restriction éventuelle sur l'espace de stockage.

4.2.3 Génération des vues matérialisées

Cette partie décrit la création de l'ensemble de vues à matérialiser. Le tableau 4.5 représente l'ensemble minimal des vues à matérialiser que nous avons obtenu en appliquant notre algorithme. Il contient pour chaque vue le nombre de jointures ainsi que les coûts de stockage et de calcul.

Vue	No de jointures	Coût de Stockage	Coût de Calcul
V4 E+T+M	3	603	61K
V5 C+T+M	3	33K	346K
V8 E+M	2	58	25K
V10 C+M	2	8K	90K
V11 T+M	2	48	60
V14 T	1	12	12
V15 M	1	4	5

TAB. 4.5 – Ensemble minimal des vues à matérialiser

L'ensemble S est = {V4, V5, V8, V10, V11, V14, V15}. Ainsi :

VM4 : Nombre de séjours par établissement, temps et mode de sortie

```
CREATE MATERIALIZED VIEW cs_sejour_etab_temps_mode_3d
ENABLE QUERY REWRITE AS
SELECT etablissement.cle_finess, temps.cle_temps, mode_sortie.cle_mode,
   sum(compteduree_sejour) AS compte_sejour,
   sum(sommeduree_sejour) AS somme_sejour
FROM etablissement, temps, mode_sortie, prise_mco
WHERE etablissement.cle_finess=prise_mco.cle_finess
   AND temps.cle_temps=prise_mco.cle_temps
   AND mode_sortie.cle_mode=prise_mco.cle_mode
GROUP BY etablissement.cle_finess, temps.cle_temps, mode_sortie.cle_mode ;
```

Résultats : 603 n-uplets

VM5 : Nombre de séjours par maladie, temps et mode de sortie

```
CREATE MATERIALIZED VIEW cs_sejour_cim10_temps_mode_3d
ENABLE QUERY REWRITE AS
SELECT cim10.cle_maladie, temps.cle_temps, mode_sortie.cle_mode,
   sum(compteduree_sejour) AS compte_sejour,
   sum(sommeduree_sejour) AS somme_sejour
```

```
FROM cim10, temps, mode_sortie, prise_mco
WHERE cim10.cle_maladie=prise_mco.cle_maladie
   AND temps.cle_temps=prise_mco.cle_temps
   AND mode_sortie.cle_mode=prise_mco.cle_mode
GROUP BY cim10.cle_maladie, temps.cle_temps, mode_sortie.cle_mode;
```

Résultats : 32918 n-uplets

VM8 : Nombre de séjours par établissement et mode de sortie

```
CREATE MATERIALIZED VIEW cs_sejour_etab_mode_2d
ENABLE QUERY REWRITE AS
SELECT etablissement.cle_finess, mode_sortie.cle_mode,
   sum(comptepduree_sejour) as compte_sejour,
   sum(sommeduree_sejour) as somme_sejour
FROM etablissement, mode_sortie, prise_mco
WHERE etablissement.cle_finess=prise_mco.cle_finess
   AND mode_sortie.cle_mode=prise_mco.cle_mode
GROUP BY etablissement.cle_finess, mode_sortie.cle_mode;
```

Résultats : 58 n-uplets

VM10 : Nombre de séjours par maladie et mode de sortie

```
CREATE MATERIALIZED VIEW cs_sejour_cim10_mode_3d
ENABLE QUERY REWRITE AS
SELECT cim10.cle_maladie, mode_sortie.cle_mode,
   sum(comptepduree_sejour) AS compte_sejour,
   sum(sommeduree_sejour) AS somme_sejour
FROM cim10, mode_sortie, prise_mco
WHERE cim10.cle_maladie=prise_mco.cle_maladie
   AND mode_sortie.cle_mode=prise_mco.cle_mode
GROUP BY cim10.cle_maladie, mode_sortie.cle_mode;
```

Résultats : 8186 n-uplets

VM11 : Nombre de séjours par temps et mode de sortie

```
CREATE MATERIALIZED VIEW cs_sejour_temps_mode_2d
ENABLE QUERY REWRITE AS
SELECT temps.cle_temps, mode_sortie.cle_mode,
   sum(comptepduree_sejour) as compte_sejour,
   sum(sommeduree_sejour) as somme_sejour
FROM temps, mode_sortie, prise_mco
```

```
WHERE temps.cle_temps=prise_mco.cle_temps
   AND mode_sortie.cle_mode=prise_mco.cle_mode
GROUP BY temps.cle_temps, mode_sortie.cle_mode ;
```

Résultats : 48 n-uplets

VM14 : Nombre de séjours par mois

```
CREATE MATERIALIZED VIEW cs_sejour_temps_1d
ENABLE QUERY REWRITE AS
SELECT temps.cle_temps, count(compteduree_sejour) as compte_sejour,
   sum(sommeduree_sejour) as somme_sejour
FROM temps, prise_mco
WHERE temps.cle_temps=prise_mco.cle_temps
GROUP BY temps.cle_temps ;
```

Résultats : 12 n-uplets

VM15 : Nombre de séjours par mode de sortie

```
CREATE MATERIALIZED VIEW cs_sejour_mode_1d
ENABLE QUERY REWRITE AS
SELECT mode_sortie.cle_mode, count(compteduree_sejour) as compte_sejour,
   sum(sommeduree_sejour) as somme_sejour
FROM mode_sortie, prise_mco
WHERE mode_sortie.cle_mode=prise_mco.cle_mode
GROUP BY by mode_sortie.cle_mode ;
```

Résultats : 4 n-uplets

4.3 Interface graphique pour la Génération (semi-automatique) d'Indicateurs

Nous avons créé deux modules, le premier permet la création et/ou modification de notre schéma et le deuxième facilite la génération de requêtes de manière quasi-automatique.

Nous avons groupé notre travail en quatre parties. La première montre l'architecture de l'interface graphique. La deuxième décrit le fonctionnement du module pour la création du schéma. La troisième schématise les types de requêtes à exécuter dans l'interface graphique. Finalement, la dernière partie expose le fonctionnement

de l'interface pour la génération (quasi-automatique) de requêtes. Nous décrivons en détail chaque partie.

4.3.1 Architecture pour l'interface graphique

Dans notre architecture, nous trouvons trois composants principaux. Le premier correspond à l'interface pour la génération (quasi-automatique) de requêtes. Le deuxième, l'entrepôt de données, a été décrit au chapitre précédent. Le troisième et dernier composant est notre gestionnaire d'évolution qui est décrit au chapitre suivant. La figure 4.7 montre notre architecture pour l'interface graphique.

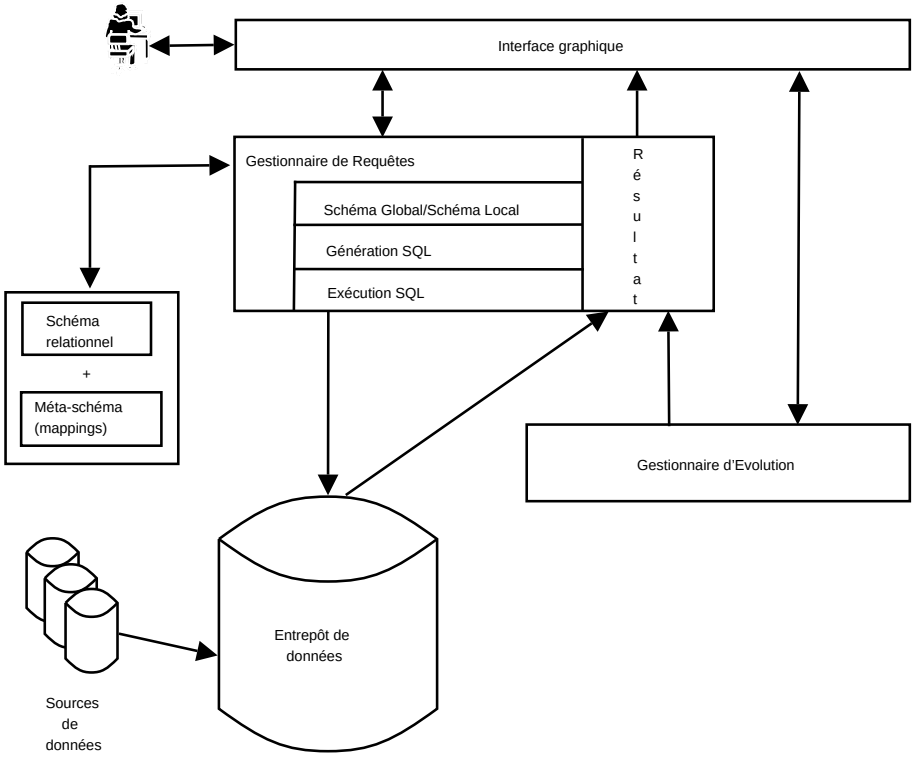

FIG. 4.7 – Architecture pour l'interface graphique

Nous décrivons les différents éléments qui composent l'interface :

Utilisateur : Les personnes qui utilisent cet outil pour exprimer leurs différentes requêtes (indicateurs).

Interface Graphique : Outil graphique où les utilisateurs peuvent choisir les divers composants (relations, mesures et propriétés) conformes à l'indicateur exprimé.

Gestionnaire de Requêtes : Il est composé de :

(Schéma Global/Schéma Local) : Composant qui permet de traduire le schéma global (méta-schéma) en schéma local (relationnel).

Génération SQL : Outil semi-automatique qui permet de générer le code SQL à partir du schéma global et du schéma local, pour répondre aux besoins des utilisateurs.

Exécution SQL : L'exécution de la requête SQL sur l'entrepôt de données.

Résultat : Les données qui correspondent à la requête et qui doivent être affichées par l'interface graphique.

Schéma Relationnel + Méta-Schéma (Mappings) : Composant qui contient les schémas (Relationnel et Méta-Schéma) et qui peut être consulté par le Gestionnaire de Requêtes.

Nous décrivons le fonctionnement des modules pour la création et/ou la mise à jour du schéma.

4.3.2 Fonctionnement de l'interface

Cette interface permet la création de la structure des relations. Nous avons construit un module pour la création et un autre pour la mise à jour des relations.

Création d'une relation

Nous pouvons créer des relations de faits ou de dimensions. La figure 4.8 montre un exemple de création de la dimension `E_Departement` pour décrire les différentes options :

Description des composants :

Relation : Il contient un n-uplet de type {Nom_relation : Type}, où le type représente une relation de Fait ou de Dimension. Par défaut il est de type Dimension.

Propriétés : Il représente un ensemble de n-uplets de type {Nom_propriété : Type}, où Type ∈ {Texte, Mémo, Numérique, Date, Monétaire, Booléen}.

Clé Primaire : Il représente un ensemble de n-uplets de type {Nom_propriété : Type}, où Nom_propriété : Type ∈ {Propriétés}.

Clé Etrangère : Il représente un ensemble de n-uplets de type {Nom_propriété : Type = Nom_relation.Nom_propriété : Type}. Où Nom_propriété : Type ∈ {Pro-

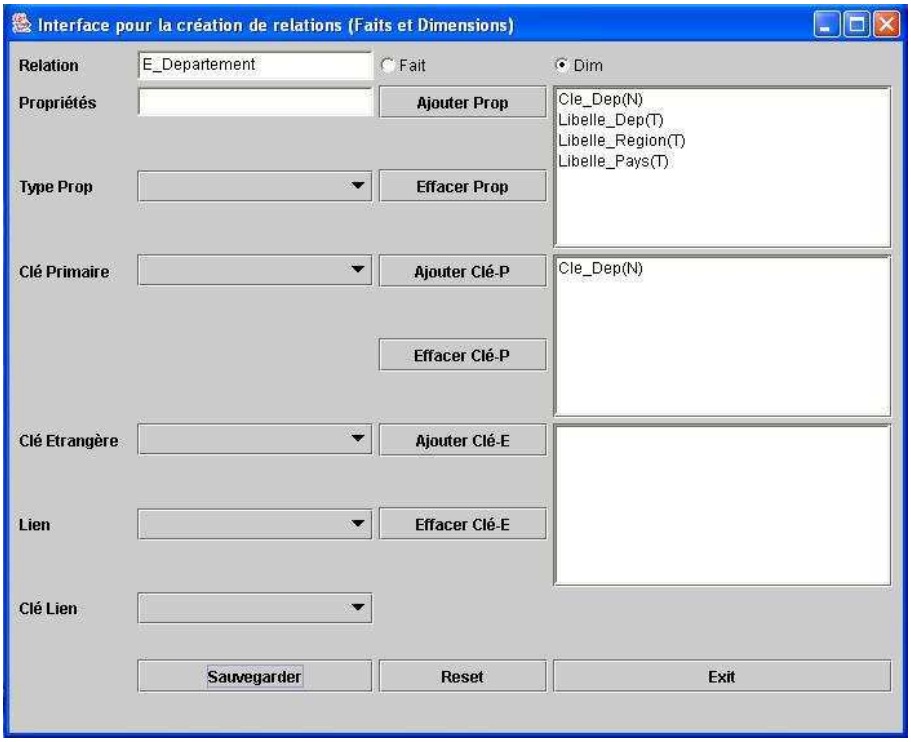

FIG. 4.8 – Création d'une dimension

priétés} et Nom_relation ∈ {Relation}.

Options : Elle est composée de :

Sauvegarder : Pour l'enregistrement de la relation.
Reset : Pour nettoyer l'écran et continuer la création des relations.
Exit : Pour sortir de l'interface de création.

La création d'une relation génère de façon automatique un fichier qui contient sa structure. L'ensemble de fichiers créés sont accédés de façon dynamique pour le module de génération d'indicateurs pendant l'exécution de requêtes.

Structure du fichier : Elle contient les méta données suivantes(nous utilisons le délimiteur #) :

#**Nom_relation(type)** : Il identifie le nom de la relation et leur type.
#**Propriétés** : Il représente l'ensemble d'attributs de la relation.
#**Key** : Il contient l'ensemble de clés primaires.
#**Fkey** : Il contient l'ensemble de clés étrangères.

La figure 4.9 montre la structure de la dimension `E_Departement`.

Elle contient le nom et type de la relation, un ensemble de trois propriétés et leur

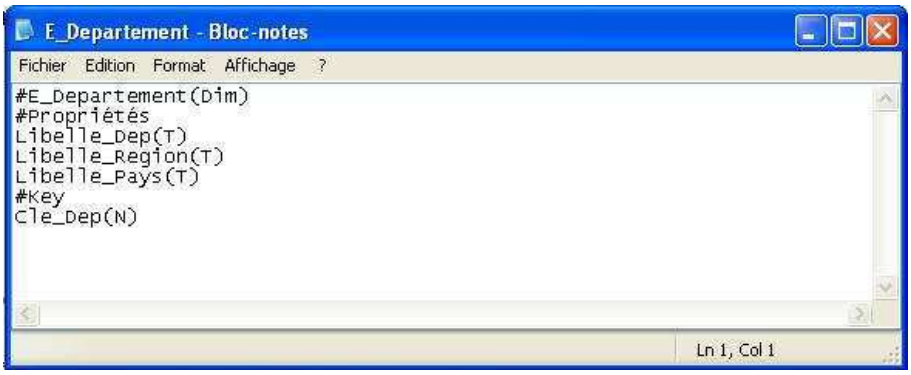

FIG. 4.9 – Structure de la dimension

clé_primaire.

Mise à jour d'une relation (fait ou dimension)

Suivant la structure de la relation à modifier, le module de mise à jour remplit les différentes options avec des méta données contenues dans la relation. Ainsi, l'utilisateur peut ajouter, effacer ou modifier les méta données contenues dans la structure, sauf le nom et le type (faits ou dimension) de la relation, car ils ne sont pas modifiables.

4.3.3 Types de requêtes à exécuter

Avant de décrire le fonctionnement de l'interface graphique, nous avons inclus des exemples de requêtes que nous pouvons exécuter. Nous avons classifié les types de requêtes par rapport au nombre de relations qu'elles intègrent. A l'intérieur du premier type, nous avons aussi divisé les requêtes par rapport à l'ensemble de paramètres qu'elles contiennent. Nous montrons quelques exemples :

Requêtes du type 1 : Correspondent aux requêtes sur 1 relation. Nous décrivons ce type :

Sélection sur 1 relation + condition sur un membre :

Q1 : Quels sont les établissements avec un nombre de lits autorisés supérieur à 100 ?

SELECT Cle_Finess, Raison_sociale, Adresse, NLA
FROM Etablissement
WHERE NLA > 100

Sélection sur 1 relation + comptage + condition sur une mesure :

Q2 : Nombre de séjours.

```
SELECT Sum(CompteDuree_sejour)
FROM Prise_MCO
HAVING Sum(SommeDuree_sejour)>100
```

Sélection sur 1 relation + comptage + group by + condition sur une mesure :

Q3 : Nombre de séjour par établissement.

```
SELECT Cle_Finess, Sum(CompteDuree_sejour)
FROM Prise_MCO
GROUP BY Cle_Finess
HAVING Sum(SommeDuree_sejour)>100
```

Requêtes de type 2 : Correspondent aux requêtes sur n relations, nous décrivons ce type avec un exemple :

Q4 : Nombre de séjours par établissement et par maladie pendant le mois janvier (avec nombre de séjours > 100).

```
SELECT Etablissement.Cle_Finess, Etablissement.Raison_sociale,
   CIM10.Libelle_maladie, Count(Prise_MCO.CompteDuree_sejour),
   Sum(Prise_MCO.SommeDuree_sejour)
FROM Prise_MCO, Etablissement, CIM10, Temps
WHERE Prise_MCO.Cle_Finess=Etablissement.Cle_Finess
   AND Prise_MCO.Cle_Maladie=CIM10.Cle_Maladie
   AND Prise_MCO.Cle_Temps=Temps.Cle_Temps
   AND Temps.Cle_Temps=1
GROUP BY Etablissement.Cle_Finess, Etablissement.Raison_sociale,
   CIM10.Libelle_maladie
HAVING Sum(SommeDuree_sejour)>100
ORDER BY Etablissement.Cle_Finess
```

Dans la partie suivante, nous décrivons l'interface graphique que nous avons conçue pour faciliter l'exécution de l'ensemble des requêtes.

4.3.4 Fonctionnement de l'interface graphique

L'objectif de notre interface est de faciliter la tâche de création d'indicateurs aux utilisateurs. La figure 4.10 décrit l'utilisation de l'interface pour la génération (semi-automatique) de requêtes. Nous avons utilisé l'indicateur : "Q4 : Nombre de séjours par établissement et par maladie pendant le mois janvier (avec nombre de séjours >

100)", pour arriver à la description des différents choix à l'intérieur de l'interface.

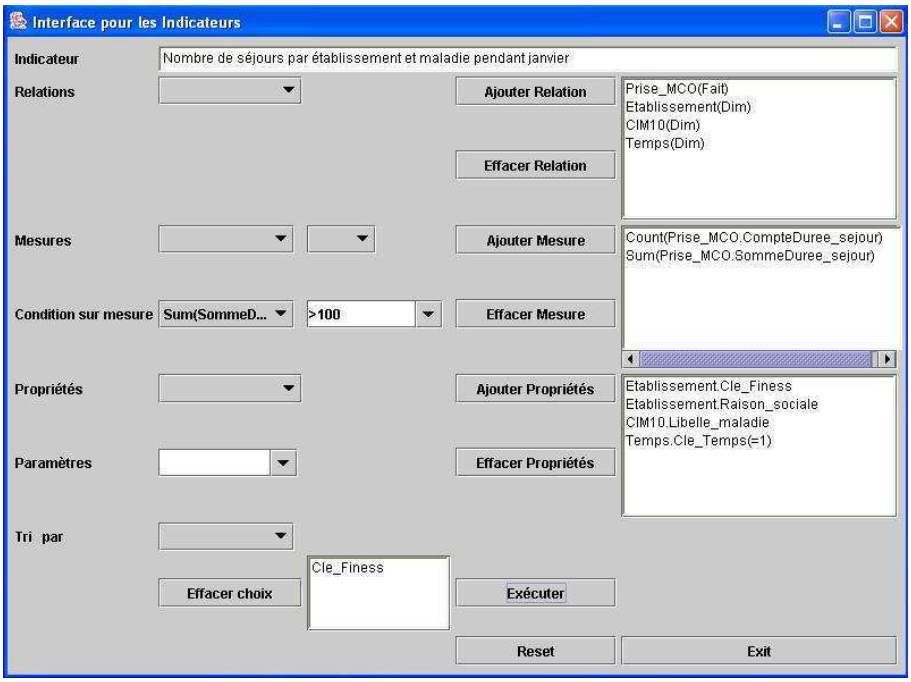

FIG. 4.10 – Interface pour la génération (semi-automatique) d'indicateurs

Par rapport à l'indicateur, nous devons choisir : la(s) relation(s), la(s) mesure(s) à calculer, la(s) propriété(s) à afficher, la(s) condition(s) de sélection (sur des mesures ou sur des membres) et l'ordre de présentation des résultats.

Description des composants :

Relations : Il contient l'ensemble des relations sélectionnées par rapport à l'indicateur exprimé.

Mesures et leur condition : Il contient les mesures à calculer ainsi que leur fonction de groupage. La condition représente une sélection sur la mesure. Dans le code SQL92, cette condition est exprimée dans la clause HAVING.

Propriétés et leurs paramètres : Ce composant contient l'ensemble de propriétés à afficher comme résultat de l'exécution de la requête. Les paramètres représentent les conditions de sélection sur les membres des dimensions. Dans le code SQL92, elles sont incluses dans la clause WHERE.

Tri : Il représente l'ensemble de propriétés à considérer pour la mise en ordre des résultats.

Options : Elle est composée de :

Exécuter : Pour l'exécution de la requête.
Reset : Pour nettoyer l'écran et continuer avec la création d'une nouvelle requête.
Exit : Pour sortir de l'interface de génération.

Finalement, pour les quatre premiers composants, nous pouvons ajouter et/ou effacer les relations, les mesures, les propriétés ou les conditions (aussi bien sur les mesures que sur les membres) qui ont été sélectionnées. Par exemple, si nous effaçons une relation, l'ensemble de ses attributs sont effacés de l'interface.

Par exemple, la figure 4.10 représente une requête qui requiert l'exécution de trois jointures et de deux conditions de sélection. La première condition est une sélection sur les mesures (`Sum(SommeDuree_sejour)>100`), tandis que la deuxième représente une sélection sur les membres de la dimension `Temps` (`Temps.Cle_Temps=1`). La fonction de groupage est automatiquement calculée par rapport aux propriétés sélectionnées. Finalement, le résultat est affiché trié par hôpital.

Chaque fois que nous choisissons l'option d'Exécution, nous créons un fichier appelé Requete.REL qui contient le code SQL92 généré. En utilisant l'outil d'administration ODBC pour établir la connexion, nous envoyons ce code vers le programme Access pour son exécution. La figure 4.11 montre le résultat de l'exécution de la requête.

FIG. 4.11 – Résultat de l'exécution de la requête Q4

L'option Exit de cet écran nous permet revenir en arrière pour réexécuter la requête, pour changer certains paramètres ou pour exécuter une nouvelle requête. Finalement, la figure 4.12 montre la requête générée.

Nous considérons que la requête Q4 illustre bien le rôle des différents composants de notre système.

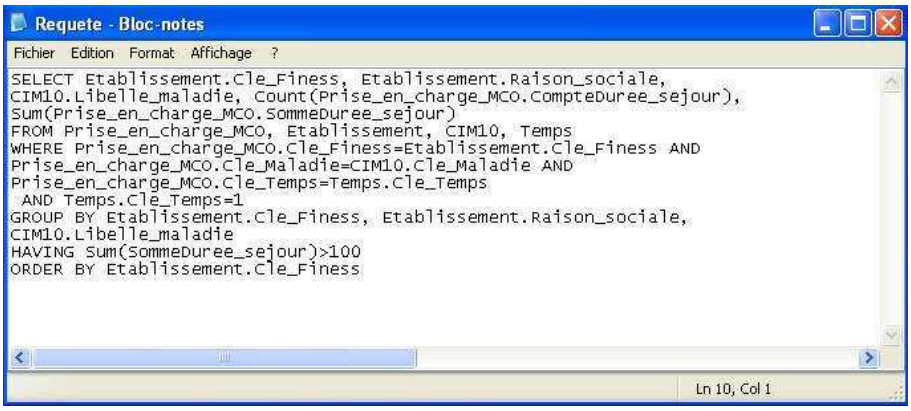

FIG. 4.12 – Fichier Requete.REL

4.4 Bilan

Dans ce chapitre, nous avons décrit notre expérimentation dans le cadre d'un système décisionnel. Pour aboutir à la mise en oeuvre du schéma, nous avons utilisé seulement une partie du modèle multidimensionnel que nous avons conçu pour le projet ADELEM (*cf.* Chapitre 3).

La première section s'est focalisée sur la construction du schéma, que nous avons appelé `Adelem_MCO`. Il se compose d'une table de faits (`Prise_MCO`) et de quatre dimensions (`Etablissement, CIM10, Temps` et `Mode_sortie`). Nous avons construit ce schéma en utilisant un échantillon du 10% des données réelles du projet.

La deuxième partie a été consacrée aux vues à matérialiser. Nous avons d'abord utilisé l'algorithme Greedy pour la sélection de l'ensemble optimal de vues à matérialiser, ensuite, nous avons appliqué notre algorithme. Nous les avons appliqué sur des données médicales réelles et nous avons constaté que notre proposition donne de meilleurs résultats pour notre cas d'expérimentation. Nous avons terminé cette partie avec la génération de l'ensemble optimal de vues à matérialiser.

La dernière section décrit l'interface pour la génération (semi-automatique) d'indicateurs. Nous avons divisé ce travail en quatre parties. La première partie décrit notre architecture pour l'interface graphique. La deuxième décrit le fonctionnement de l'interface pour la création et/ou la mise à jour du schéma. Nous avons groupé l'ensemble des éléments qui intègrent cette interface dans 5 composants et nous avons terminé cette partie avec la description de chaque composant.

Dans la troisième partie, nous avons classé les types de requêtes par rapport au nombre de dimensions. Nous avons aussi donné des exemples de chaque type de requêtes à exécuter sur l'interface graphique et nous avons terminé cette section avec la description du fonctionnement de l'interface graphique pour la génération (semi-automatique) de requêtes. Nous avons utilisé un exemple pour décrire son

fonctionnement.

Dans le chapitre suivant, nous déployons notre proposition pour la gestion, le stockage et la visualisation de l'ensemble de données contenues dans un entrepôt de données. Nous proposons l'utilisation des versions de schémas bitemporels pour la gestion de l'évolution des données aussi bien intensionnelles que extensionnelles.

Chapitre 5

Aspects temporels et versions de schémas dans les entrepôts

Malgré une conception attentive et soigneuse, la structure d'une base de données est sujette à de nombreux changements. Ces changements peuvent affecter aussi bien les données que leur structure. Pour gérer ces changements, nous pouvons utiliser l'évolution de schémas ou les versions de schémas. Pour résoudre le problème de perte de données après des changements du schéma, le concept d'évolution de schéma à été introduit pour récupérer les données existantes par le biais de leur adaptation au nouveau schéma. Néanmoins, nous considérons que pour les systèmes qui doivent gérer des données historiques, l'évolution de schéma n'est pas suffisante et la maintenance de plusieurs schémas est requise. Nous devons choisir la technique à utiliser par rapport aux caractéristiques de l'application cible.

Dans ce chapitre, nous utilisons un schéma en évolution qui contient aussi bien des changements sur les données extensionnelles que sur les données intensionnelles. Ceci nous permet d'introduire le concept de versions de schémas qui permettent de gérer ces changements en gardant l'ensemble des données. Nous nous inspirons des besoins et des caractéristiques d'une application médicale.

Nous proposons l'utilisation des versions de schémas pour la gestion, le stockage et la visualisation des données historisées de l'application médicale ADELEM. Néanmoins, la gestion des versions est une tâche complexe et elle présente plusieurs problèmes. Le principal, à notre connaissance, est l'explosion des versions, car l'exécution de chacune des primitives entraîne la génération d'une nouvelle version. Ainsi, nous devons établir un mécanisme pour contrôler cette explosion. Pour faire cela, nous avons défini un opérateur appelée *SetVersion* qui permet de grouper un ensemble de primitives à exécuter sur une version de schéma et qui génère une nouvelle [SA05b].

La gestion des versions de schémas requiert aussi la spécification d'un ensemble de composants. D'abord, nous avons besoin d'un ensemble de primitives permettant l'évolution d'une version vers une autre et d'un ensemble de règles d'intégrité. Nous

avons aussi besoin d'un gestionnaire d'évolution qui manipule les différentes versions de schémas historisées et d'un moteur de versions. Ce dernier gère le stockage des deltas et les règles de configuration nécessaires lors de la manipulation des versions de schémas.

5.1 Versions de schémas

Cette section présente une introduction aux versions de schémas. Nous utilisons d'abord un schéma en évolution pour décrire le besoin de la gestion des versions de schémas. Ensuite, nous montrons la représentation bitemporelle de cet ensemble de versions.

5.1.1 Versions de schémas annuels dans un contexte médical

Nous utilisons un schéma en étoile avec quatre dimensions qui est décrit par la figure 5.1. Il représente une évolution annuelle d'une application médicale. La table centrale représente une table de faits (`Prise_MCO`). Elle contient un n-uple pour chaque séjour dans un hôpital.

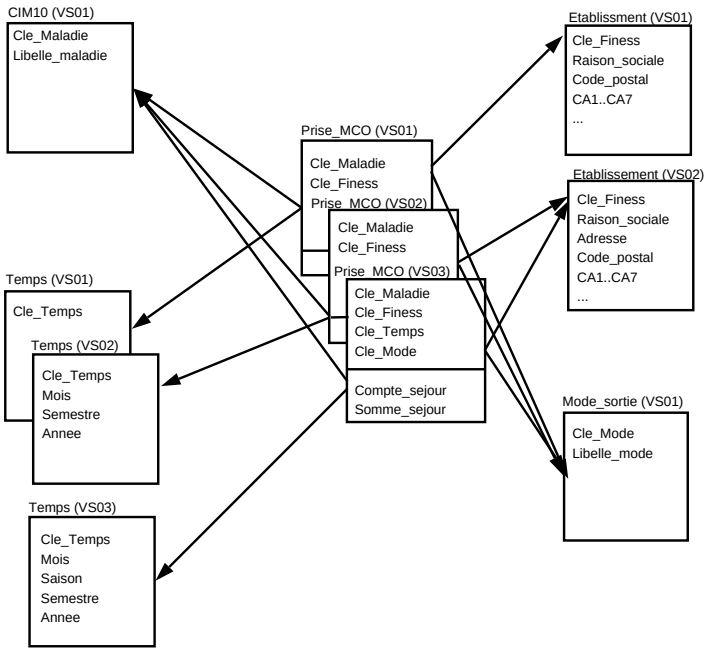

FIG. 5.1 – Trois versions de schémas annuels.

La figure 5.1 décrit trois versions de schémas annuels pour la table de faits `Prise_MCO` et pour la dimension `Temps`. Néanmoins, dans la dernière version de schéma de la dimension `Temps` VS03, le schéma a été modifié par l'ajout de l'attribut `Saison`. La dimension `Etablissement` représente deux versions, où la version de schéma VS01 représente la version pour l'année 2001 et la version VS02 correspond aux versions de schémas annuelles pour l'année 2002 et 2003. Cette version a été modifiée par

l'ajout de l'attribut `Adresse`. Finalement, les dimensions `CIM10` et `Mode_sortie` ne changent pas d'un schéma à l'autre.

5.1.2 Représentation bitemporelle des versions de schémas

Pour la gestion de la notion du temps dans notre modèle de versions de schémas, nous pouvons utiliser soit le temps transactionnel, soit le temps de validité, ou bien le bitemporel [DGS95, GM02, AQ86]. Le temps de validité d'une version correspond aux instants auxquels cette version est vraie dans le monde modélisé. De cette manière, nous pouvons l'interroger par rapport à la validité, des données de cette version, du schéma correspondant, ou en utilisant un schéma valide dans un intervalle différent. Par contre, le temps transactionnel correspond à l'instant où un fait est enregistré dans la base. Dans cette approche, nous pouvons seulement modifier le schéma courant. Ainsi, si par exemple, nous avons besoin d'accéder à une version historique pour corriger certains erreurs en gardant l'histoire complète des changements du schéma, nous devons choisir le bitemporel pour la gestion du temps. Dans notre cas, nous avons décidé de manipuler les versions de schémas bitemporels. Ainsi, nous devons gérer la combinaison du temps de transaction et du temps de validité (*cf.* Subsection 2.6.1.3).

La figure 5.2 montre la gestion du temps pour l'application médicale présentée dans la figure 5.1.

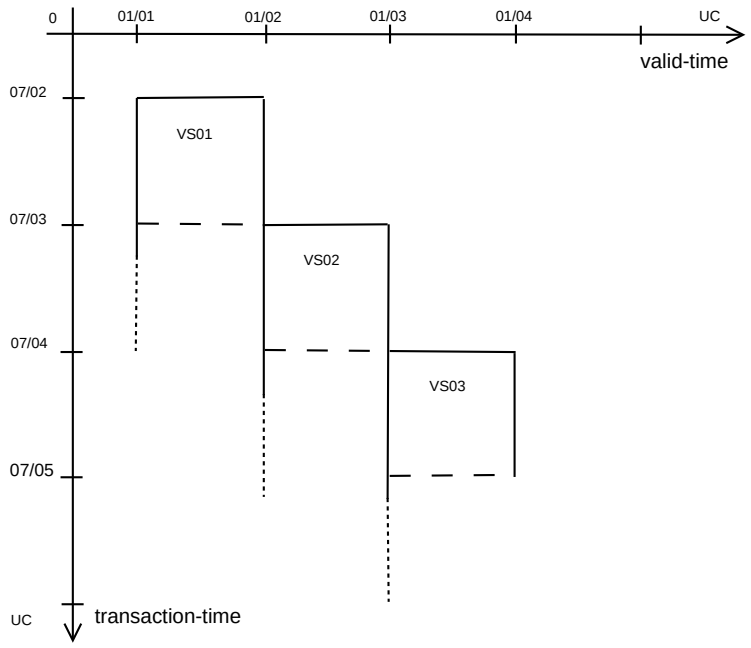

FIG. 5.2 – Représentation bitemporelle des versions de schémas.

Cette figure représente les versions de schémas bitemporels *VS01, VS02* et *VS03*. Chaque version de schéma a un temps de transaction et un temps de validité. Par

exemple pour la version VS01, nous avons un temps de transaction 2002-07 et un temps de validité de 2001-01 - 2001-12. Dans la suite, nous décrivons les versions de schémas bitemporels. Nous supposons l'existence de l'ensemble {*C, D, H, M, P, L, R*} (*cf.* paragraphe 3.2.4) et de :

V = Ensemble de noms des versions de schémas.
CS = Ensemble des changements sur les schémas.
δ = Représente une fonction *timestamp*[1].

$TIME = (TIME_t \text{ X } TIME_v)$

δ : TIME $\longrightarrow VS \bigcup \{\emptyset\}$

$$\delta(tt, vt) = \begin{cases} VS_i & \text{si } VS_i \text{ active dans } (tt, vt) \\ \emptyset & \text{sinon} \end{cases}$$

Nous dénotons par δ^{-1}(VS) le temps associé par la fonction δ à VS. Ainsi, δ^{-1}(VS) représente la pertinence temporelle (timestamp) de VS.

Finalement, notre version de schéma représente un n-uplet $VS = (V_s, C_s, D_s, H_s, R, CS, \delta^{-1})$ par rapport à la figure 5.1, tel que :

V_s = {VS01, VS02, VS03}
C_s = {Prise_MCO}
M = {Compte_sejor, Somme_sejour}
D_s = {Etablissement, CIM10, Temps, Mode_sortie}
P = {Cle_Finess, Raison_sociale, Adresse, Codepostal, CA1 .. CA7, CMO1 .. CMO7, NLA, NLO, Commune, Departement, Region, Pays, Cle_Maladie, ... }
H_s = {$\emptyset$}
L = {$\emptyset$}
CS = {(AddPropriete$_{Adresse}$, VS01, VS02), (AddPropriete$_{Saison}$, VS02, VS03)}
$\delta^{-1}(VS01)$ = (2002-07, UC[2])$_t$ X (2001-01 - 2001-12)$_v$
$\delta^{-1}(VS02)$ = (2003-07, UC)$_t$ X (2002-01 - 2002-12)$_v$
$\delta^{-1}(VS03)$ = (2004-07, UC)$_t$ X (2003-01 - 2003-12)$_v$

La gestion de l'ensemble des changements d'un schéma en évolution requiert un nombre d'opérations primitives qui permettent aux données intensionnelles et extensionnelles, de continuer d'exister ou d'être valides.

[1]Nous utilisons la fonction δ présentée dans [GM02]
[2]Until Change

5.2 Les opérations primitives

Nous décrivons les opérations primitives que nous avons définies pour la gestion d'un schéma en évolution. L'ensemble d'opérations a été inspiré par les travaux de recherche dans le domaine de l'évolution de schémas [Ben02, KC88, KC02, GM02, GMS00, Lau96, RKZ00]. Nous avons adapté l'ensemble de primitives pour les entrepôts de données et nous avons défini les règles d'intégrité requises.

5.2.1 Ensemble de primitives

Nous avons divisé les primitives en deux catégories, la première pour la gestion des cubes, des dimensions ou des hiérarchies. La deuxième correspond aux primitives qui agissent sur les versions de schémas.

Gestion de la relation (cube, dimension ou hiérarchie) :

Nous avons construit le tableau 5.1 avec l'ensemble des primitives qui agissent sur le contenu d'une relation avec leur définition et les règles d'intégrité.

Gestion des versions :

Le trableau 5.2 décrit l'ensemble des primitives construites pour la gestion de versions avec leur défintion et les règles d'intégrité.

5.2.2 Ensemble de restrictions

Nous présentons l'ensemble des règles d'intégrité que nous avons conçu. Le format général pour leur définition est le suivant :

constraint **nom_contrainte :=** *constraint* **;**

Nous listons l'ensemble des restrictions que nous avons défini par rapport aux opérations primitives précédentes. Nous avons aussi divisé les restrictions en deux catégories.

Règles d'intégrité définies

Sur une relation :

R1 : constraint addm := type(m_n) = numérique.
R2 : constraint deletem := {M} $\geq$ 2.
R3 : constraint created := clé primaire $\in$ {FC}, où FC est l'ensemble de clés étrangères qui appartiennent à la relation de faits.
R4 : constraint dropd := clé primaire $\notin$ {FC}.

Primitive	**Définition**	**Règle d'intégrité**
CreateC	CreateC(BM, c_n, {M}, {D})	
DropC	DropC(BM, c_n)	
RenameC	RenameC(BM, c_n, c'_n)	
AddM	AddM(BM, c_n, m_n)	(R1) type(m_n) = numérique
DeleteM	DeleteM(BM, c_n, m_n)	(R2) {M} ≥ 2
RenameM	RenameM(BM, c_n, m_n, m'_n)	
CreateD	CreateD(BM, d_n, {P}, {H})	(R3) clé primaire $\in$ {FC}
DropD	DropD(BM, d_n)	(R4) clé primaire $\notin$ {FC}
RenameD	RenameD(BM, d_n, d'_n)	
AddP	AddP(BM, d_n, p_n)	
DeleteP	DeleteP(BM, d_n, p_n)	(R5) propriété $\neq$ clé primaire
Change_ typeP	Change_typeP(BM, d_n, p_n, p'_n)	(R6) propriété $\neq$ clé primaire
RenameP	RenameP(BM, d_n, p_n, p'_n)	(R7) propriété $\neq$ clé primaire
CreateH	CreateH(BM, h_n, L, $\prec$)	(R8) clé primaire $\in$ {FC} $\vee$ clé primaire $\in$ {FD}
DropH	DropH(BM, h_n)	(R9) clé primaire $\notin$ {FC} $\vee$ clé primaire $\notin$ {FD}
RenameH	RenameH(BM, h_n, h'_n)	
AddL	AddL(BM, h_n, l_n, l_1, l_2)	
DeleteL	DeleteL(BM, h_n, l_n)	(R10) clé primaire $\notin$ {FC} $\vee$ clé primaire $\notin$ {FD}
RenameL	RenameL(BM, h_n, l_n, l'_n)	(R11) clé primaire $\in$ {FC} $\vee$ clé primaire $\in$ {FD}

TAB. 5.1 – Primitives pour la gestion de relation

Primitive	**Définition**	**Règle d'intégrité**
CreationV	AddV(BMS, VS_n)	
FreezeV	FreezeV(VS_n)	(R12) δ(VS_n)=vrai
DefrostV	DefrostV(VS_n)	
ExporterR	ExporterR(VS_n, CIM10, VS'_n)	(R13) $d_n \in VS_n$
DropV	DropV(VS_n)	(R14) DropV(VS_n) $\wedge$ β(VS_n)=vrai

TAB. 5.2 – Primitives pour la gestion des versions

R5 : constraint deletep := propriété $\neq$ clé primaire.
R6 : constraint change_typep := propriété $\neq$ clé primaire.
R7 : constraint renamep := propriété $\neq$ clé primaire.
R8 : constraint createh := clé primaire $\in$ {FC} $\vee$ clé primaire $\in$ {FD}, où FD est l'ensemble de clés étrangères à l'intérieur de la dimension.
R9 : constraint droph := clé primaire $\notin$ {FC} $\vee$ clé primaire $\notin$ {FD}.
R10 : constraint deletel := clé primaire $\notin$ {FC} $\vee$ clé primaire $\notin$ {FD}.
R11 : constraint renamel := clé primaire $\in$ {FC} $\vee$ clé primaire $\in$ {FD}.

Sur une version :

R12 : constraint freezev := version de schéma $\in$ à l'historique.
R13 : constraint exporterr := relation_nom $\in$ version de schéma.
R14 : constraint dropv := version de schéma $\notin$ à l'historique et version de schéma $\neq$ version courante.

5.3 Définition formelle des primitives

Dans cette section, nous traitons la définition de l'ensemble des primitives requises pour la création d'une version de schéma. Nous avons gardé les deux catégories spécifiées précédemment. Nous avons aussi identifié un ensemble de propriétés qui doivent être préservées lors des évolutions.

Noms uniques : Les versions de schémas, les cubes, les dimensions, les hiérarchies, les mesures, les propriétés et les niveaux ont des noms uniques.

Schéma du cube : La cardinalité de l'ensemble d'axes et de l'ensemble de mesures doit être supérieure ou égale à 1.

Schéma de dimension : La cardinalité de l'ensemble des propriétés doit être supérieure ou égale à 1.

Schéma de hiérarchie : Il est représenté par un graphe acyclique dirigé qui contient un seul noeud *base* $\perp$ et un seul noeud *sommet* $\top$.

Schéma de version : Il est représenté par un schéma multidimensionnel VS=(V, C, D, H, R, CS, δ^{-1}) et l'instance est IV=(V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}).

La sémantique de l'ensemble de primitives doit respecter ces propriétés. Pour la définition des primitives, nous avons repris les 4 définitions pour la spécification d'une base de données multidimensionnelle déjà présentées (*cf.* Chapitre 3).

5.3.1 Gestion de la relation (cube, dimension ou hiérarchie)

Dans cette partie, nous présentons l'ensemble des définitions des primitives qui agissent sur les schémas de cubes, de dimensions ou de hiérarchies.

Définition 5.1 : Primitive *CreateC*

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), un nom de cube $c_n \in C$, un ensemble de dimensions D tel que $|D| \geq 1$ et un ensemble de mesures M tel que $|M| \geq 1$, le résultat de la primitive CreateC(VS, c_n, M, D) est une version dont le schéma est VS' = (V_s, $C_s \bigcup \{c_s\}$, D_s, H_s, R, CS, δ^{-1}), où c_s est un schéma de cube (c_n, M, D) et l'instance est IM' = IM. □

Considérons l'insertion dans la version VS_i d'un nouveau cube `Achats` qui contient l'ensemble de dimensions `{Magasin, Temps, Produit, Fournisseur}` et la mesure `QuantiteA`, la primitive *CreateC(VS_i, Achats, {Magasin, Temps, Produit, Fournisseur}, QuantiteA)* crée le cube.

Définition 5.2 : Primitive *DropC*

Cette primitive élimine un schéma de cube, ses dimensions et ses hiérarchies d'une version de schéma VS.

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), un nom de cube $c_n \in C$, la primitive DropC(VS, c_n) crée une version dont le schéma est VS' = (V_s, C'_s, D_s, H_s, R, CS, δ^{-1}), où $C'_s = C_s \backslash \{(c_n, m_1, \ldots, m_j, D)\}$. L'instance est IV' = ($V_i$, $C_i \backslash \{c_i\}$, D_i, H_i, R_i, CS_i, δ_i^{-1}), où $c_i = \{(c_n, \{(v_1, \ldots, v_j) | \forall_{i=1,\ldots,n}\ v_i \in measuredom(m_i)\}, \{d\})\}$. □

Par exemple, pour éliminer le cube `Achats` de la version VS_i, nous exécutons la primitive *DropC(VS_i, Achats)*.

Définition 5.3 : Primitive *RenameC*

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), les noms de cubes c_n $c'_n \in C$, l'exécution de la primitive RenameC(VS, c_n, c'_n) fournit une version dont le schéma est VS' = (V_s, C'_s, D_s, H_s, R, CS, δ^{-1}), où $C'_s = C_s \backslash \{(c_n, M, D)\} \bigcup \{(c'_n, M, D)\}$, et l'instance est IV' = IV. □

Par exemple, pour renommer le cube `Ventes` par `VentesJournalieres` : *RenameC(VS_i, Ventes, VentesJournalieres)*.

Définition 5.4 : Primitive *AddM*

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), un nom de cube $c_n \in C$ et un nom de mesure $m_n \in M$, la primitive AddM(VS, c_n, m_n) crée une version dont le schéma est VS' = (V_s, C'_s, D_s, H_s, R, CS, δ^{-1}), où $C'_s = C_s \setminus \{(c_n, M, D)\} \bigcup \{(c_n, M \bigcup \{m_n\}, D)\}$, et l'instance est IV' = (V_i, C'_i, D_i, H_i, R_i, CS_i, δ_i^{-1}) où $C'_i = C_i \setminus \{c_i\} \bigcup \{c'_i\}$, où $c_i = (c_n, \{(m_1, \ldots, m_i)\}, \{d\})$ et $c'_i = (c_n, \{(m_1, \ldots, m_i)\} \bigcup \{m_n\}, \{d\})$. □

Par exemple, pour ajouter la mesure `QuantiteV` au cube `Ventes`, nous exécutons la primitive *AddM(VS_i, Ventes, QuantiteV).*

Définition 5.5 : Primitive *DeleteM*

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), un nom de cube $c_n \in C$ et un nom de mesure $m_n \in M$ tel que $|M| \geq 2$, le résultat de DeleteM(VS, c_n, m_n) est une version dont le schéma est VS' = (V_s, C'_s, D_s, H_s, R, CS, δ^{-1}), où $C'_s = C_s \setminus \{(c_n, M, D)\} \bigcup \{(c_n, M \setminus \{m_n\}, D)\}$. L'instance IV' = ($V_i$, C'_i, D_i, H_i, R_i, CS_i, δ_i^{-1}) où $C'_i = C_i \setminus \{c_i\} \bigcup \{c'_i\}$, où $c_i = (c_n, \{(m_1, \ldots, m_i)\}, \{d\})$ et $c'_i = (c_n, \{(m_1, \ldots, m_i)\} \setminus \{m_n\}, \{d\})$. □

Par exemple, pour supprimer la mesure `QuantiteV` de cube `Ventes` : *DeleteM(VS_i, Ventes, QuantiteV).*

Définition 5.6 : Primitive *RenameM*

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), un nom de cube $c_n \in C$ et les noms m_n et $m'_n \in M$, l'exécution de la primitive RenameM(VS, c_n, m_n, m'_n) donne une version dont le schéma est VS' = (V_s, C'_s, D_s, H_s, R, CS, δ^{-1}), où $C'_s = C_s \setminus \{(c_n, M, D)\} \bigcup \{(c_n, M \setminus \{m_n\} \bigcup \{m'_n\}, D)\}$. L'instance est IV' = IV. □

Par exemple, pour changer le nom de la mesure `Quantite` par `QuantiteJournaliere`, nous exécutons la primitive *RenameM(VS_i, Ventes, Quantite, QuantiteJournaliere).*

Définition 5.7 : Primitive *CreateD*

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), un nom de cube $c_n \in C$ un nom de dimension $d_n \in D$, un ensemble de propriétés P tel que $|P| \geq 1$ et H, le résultat de la primitive CreateD(VS, C_n, d_n, P, H) est une version dont le schéma est VS' = (V_s, C'_s, $D'_s \bigcup \{d_s\}$, H_s, R, CS, δ^{-1}), où $C'_s = C_s \setminus \{(c_n, M, D)\} \bigcup \{(c_n, M, D \bigcup \{d_n\})\}$ et $D'_s = D_s \bigcup \{d_s\}$. L'instance est IV' = ($V_i$, C'_i, D'_i, H_i, R_i, CS_i, δ_i^{-1}), où $C'_i = C_i \setminus \{c_i\} \bigcup \{c'_i\}$, $c_i = (c_n, \{m\}, \{d_1, \ldots d_i\})$, $c'_i = (c_n, \{m\}, \{d_1, \ldots d_i\} \bigcup \{d_n\})$ et D'_i

$= D_i$. □

Considérons par exemple la création de la dimension `Fournisseur` qui contient l'ensemble de propriétés `{Cle_F, Raison_sociale, Adresse, Téléphone, Pays}` au cube `Ventes`, nous exécutons la primitive *CreateD(VS_i, Ventes, Fourniseur, {Cle_F, Raison_sociale, Adresse, Téléphone, Pays}, {∅})*.

Définition 5.8 : Primitive *DropD*

Cette primitive élimine un schéma de dimension et leurs instances d'une version VS_i. La primitive ne peut pas être exécutée s'il existe un schéma de cube qui utilise cette dimension. Ainsi, nous supposons l'existence d'une fonction booléenne α indiquant si la dimension peut être supprimée.

Etant donnés une version de schéma $VS = (V_s, C_s, D_s, H_s, R, CS, \delta^{-1})$ et d'instance $IV = (V_i, C_i, D_i, H_i, R_i, CS_i, \delta_i^{-1})$, un nom de dimension $d_n \in D$ tel que $\alpha(d_n = vrai)$, la primitive DropD(VS, d_n) crée une version dont le schéma est $VS' = (V_s, C_s, D'_s, H_s, R, CS, \delta^{-1})$, où $D'_s = D_s \setminus \{(d_n, P, H)\}$. L'instance est $IV' = (V_i, C_i, D'_i, H_i, R_i, CS_i, \delta_i^{-1})$, où $D'_i = D_i \setminus \{d_i\}$, où $d_i = (d_n, \{(v_1,..., v_x) | \forall_{i=1,...,n} v_i \in propertydom(p_i)\}, \{h\})$. □

Par exemple, pour supprimer la dimension `Magasin`, nous exécutons la primitive *DropD (VS_i, Magasin) ∧ α(Magasin)=vrai.*

Définition 5.9 : Primitive *RenameD*

Etant donnés une version de schéma $VS = (V_s, C_s, D_s, H_s, R, CS, \delta^{-1})$ et d'instance $IV = (V_i, C_i, D_i, H_i, R_i, CS_i, \delta_i^{-1})$, un nom de cube $c_n \in C$ et deux noms de dimensions $d_n, d'_n \in D$, l'opération RenameD(VS, c_n, d_n, d'_n) fournit une version dont le schéma est $VS' = (V_s, C'_s, D'_s, H_s, R, CS, \delta^{-1})$, où $C'_s = C_s \setminus \{(c_n, M, D)\} \bigcup \{(c_n, M, D \setminus \{d_n\} \bigcup \{d'_n\})\}$ et $D'_s = D_s \setminus \{(d_n, P, H)\} \bigcup \{(d'_n, P, H)\}$ et l'instance est $IV' = IV$. □

Par exemple, pour changer le nom de la dimension `Magasin` par le nom `Boutique` du cube Ventes : *RenameD(VS_i, Ventes, Magasin, Boutique).*

Définition 5.10 : Primitive *AddP*

Etant donnés une version de schéma $VS = (V_s, C_s, D_s, H_s, R, CS, \delta^{-1})$ et d'instance $IV = (V_i, C_i, D_i, H_i, R_i, CS_i, \delta_i^{-1})$, un nom de dimension $d_n \in D$ et un nom de propriété $p_n \in P$, la primitive AddP(VS, d_n, p_n) crée une version dont le schéma est $VS' = (V_s, C_s, D'_s, H_s, R, CS, \delta^{-1})$, où $D'_s = D_s \setminus \{(d_n, P, H)\} \bigcup \{(d_n, P \bigcup \{p_n\}, H)\}$, et l'instance est $IV' = (V_i, C_i, D'_i, H_i, R_i, CS_i, \delta_i^{-1})$ où $D'_i = D_i \setminus \{d_i\} \bigcup \{d'_i\}$, où $d_i = (d_n, \{(p_1,..., p_i)\}, \{h\})$ et $d'_i = (d_n, \{(p_1,..., p_i)\} \bigcup \{p_n\}, \{h\})$. □

Par exemple, pour ajouter la propriété `Telephone` à la dimension `Magasin`, nous exécutons la primitive *AddP(VS$_i$, Magasin, Telephone)*.

Définition 5.11 : Primitive *DeleteP*

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), un nom de dimension $d_n \in D$ et un nom de propriété $p \in P$ tel que $|P| \geq 2$, le résultat de DeleteP(VS, d_n, p) est une version dont le schéma est VS' = (V_s, C_s, D'_s, H_s, R, CS, δ^{-1}), où $D'_s = D_s \setminus \{(d_n, P, H)\} \bigcup \{(d_n, P \setminus \{p\}, H)\}$. L'instance IV' = ($C_i$, D'_i, H_i, R_i, CS_i, δ_i^{-1}) où $D'_i = D_i \setminus \{d_i\} \bigcup \{d'_i\}$, où $d_i = (d_n, \{(p_1,..., p_i)\}, \{h\})$ et $d'_i = (d_n, \{(p_1,..., p_i)\} \setminus \{p\}, \{h\})$. □

Pour supprimer la propriété `Telephone` de la dimension `Magasin`, nous exécutons la primitive *DeleteP(VS$_i$, Magasin, Telephone)*.

Définition 5.12 : Primitive *Change_ typeP*

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), un nom de dimension $d_n \in D$ et les noms de propriétés p_n, $p'_n \in P$, le résultat de Change_ typeP(BM, d_n, p_n, p'_n) est une version dont le schéma est VS' = (V_s, C_s, D'_s, H_s, R, CS, δ^{-1}), où $D'_s = D_s \setminus \{(d_n, P, H)\} \bigcup \{(d_n, P \setminus \{p_n\} \bigcup \{p'_n\}, H)\}$. L'instance IV' = ($V_i$, C_i, D'_i, H_i, R_i, CS_i, δ_i^{-1}) où $D'_i = D_i \setminus \{d_i\} \bigcup \{d'_i\}$, où $d_i = (d_n, \{(p_1,..., p_i)\} \setminus \{p_n\}, \{h\})$ et $d'_i = (d_n, \{(p_1,..., p_i)\} \bigcup \{p'_n\}, \{h\})$. □

Par exemple, pour changer la propriété `Codepostal` de *integer* à *char* : *Change_ typeP(VS$_i$, Magasin, Codepostal, CodepostalChar)*.

Définition 5.13 : Primitive *RenameP*

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), un nom de dimension $d_n \in D$ et les noms de propriétés p_n, $p'_n \in P$, le résultat de RenameP(VS, d_n, p_n, p'_n) est une version dont le schéma est VS' = (V_s, C_s, D'_s, H_s, R, CS, δ^{-1}), où $D'_s = D_s \setminus \{(d_n, P, H)\} \bigcup \{(d_n, P \setminus \{p_n\} \bigcup \{p'_n\}, H)\}$. L'instance IV' = IV. □

Par exemple, pour changer le nom de la propriété `Raison` par `Raison _sociale` : *RenameP(VS$_i$, d_n, Raison, Raison_ sociale)*.

Définition 5.14 : Primitive *CreateH*

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), un nom de dimension $d_n \in D$ un nom de

hiérarchie $h_n \in H$, un ensemble de niveaux L et une relation $\prec$ définie sur L, le résultat de la primitive CreateH(VS, d_n, h_n, L, $\prec$) est une version dont le schéma est VS' = (V_s, C_s, D'$_s$, H'$_s$, R, CS, δ^{-1}), où D'$_s$ = $D_s \setminus \{(d_n, P, H)\} \bigcup \{(d_n, P, H \bigcup \{h_n\})\}$ et H'$_s$ = $H_s \bigcup \{h_s\}$, où h_s est un schéma de hiérarchie (h_n, L, $\prec$). L'instance est IV' = (V_i, C_i, D'$_i$, H'$_i$, R_i, CS_i, δ_i^{-1}), où D'$_i$ = $D_i \setminus \{d_i\} \bigcup \{d'_i\}$, d_i = (d_n, {p}, {(h_1, ..., h_k)}), d'$_i$ = (d_n, {p}, {(h_1, ..., h_k)}$\bigcup \{h_n\}$) et H'$_i$ = H_i. □

Considérons par exemple la création de la hiérarchie `H_Temps` qui contient l'ensemble de niveaux `{Jour, Mois, Saison, Annee}`. La relation $\prec$ entre les niveaux est {(Jour, Mois), (Mois, Saison), (Saison, Annee)}. La primitive *CreateH(VS_i, Etablissement, H_ Temps, L, $\prec$)* insère à la dimension `Etablissement`, la nouvelle hiérarchie.

Définition 5.15 : Primitive *DropH*

Nous ne pouvons pas utiliser cette primitive s'il existe un schéma de cube utilisant un niveau de la hiérarchie en tant qu'axe. Ainsi, nous supposons l'existence d'une fonction booléenne ϵ qui nous indique si nous pouvons exécuter la primitive.

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), un nom de hiérarchie $h_n \in H$ tel que $\epsilon(h_n)$=vrai, la primitive DropH(VS, h_n) fournit une version dont le schéma est VS' = (V_i, C_s, D_s, H'$_s$, R, CS, δ^{-1}), où H'$_s$ = $H_s \setminus \{(h_n, L, \prec)\}$. L'instance est IV' = ($V_i$, C_i, D_i, H'$_i$, R_i, CS_i, δ_i^{-1}), où H'$_i$ = $H_i \setminus \{(\bigcup_{l_i \in L}$ levelset(l_i), RUP)}. □

Par exemple, pour supprimer la hiérarchie `H_Temps` : *DropH(VS_i, H_ Temps).*

Définition 5.16 : Primitive *RenameH*

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), le nom de dimension $d_n \in D$, et deux noms de hiérarchies h_n et h'$_n \in H$, la primitive RenameH(VS, d_n, h_n, h'$_n$) crée une version dont le schéma est VS' = (V_s, C_s, D'$_s$, H'$_s$, R, CS, δ^{-1}), où D'$_s$ = $D_s \setminus \{(d_n, P, H)\} \bigcup \{(d_n, P, H \setminus \{h_n\} \bigcup \{h'_n\})$ et H'$_s$ = $H_s \setminus \{(h_n, L, \prec)\} \bigcup \{(h'_n, L, \prec)\}$. L'instance est IV'= IV. □

Par exemple, pour renommer la hiérarchie `H_Temps` de la dimension Temps par le nom `H_Jour`, nous exécutons la primitive *RenameH(VS_i, Temps, H_ Temps, H_ Jour).*

Définition 5.17 : Primitive *AddL*

Cette primitive modifie le schéma de hiérarchie h_s en construisant une relation entre les niveaux l_1 et l_2 qui appartiennent à h_s et un nouveau niveau l_n. Cette opération établit les relations de dépendances suivantes : $l_1 \prec l_n$ et $l_n \prec l_2$. Pour faire cela, nous supposons l'existence d'une fonction γ qui, étant donnés un nom de hié-

rarchie h_n et deux niveaux l_i et l_j, retourne une valeur booléenne égale à *vrai* si $l_i \prec l_j$.

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), un nom de hiérarchie $h_n \in H$, les niveaux l_n, l_1 et $l_2 \in L$, et un membre $m \in levelset(l_n)$, la primitive AddL(VS, h_n, l_n, l_1, l_2) crée une version dont le schéma est VS' = (V_s, C_s, D_s, H'_s, R, CS, δ^{-1}) tel que $H'_s = (H_s \setminus \{h_s\} \bigcup \{h'_s\})$, où $h_s = (h_n, L, \prec)$ et $h'_s = (h_n, L \bigcup \{l_n\}, \prec \bigcup \{(l_1 \prec l_n), (l_n \prec l_2)\})$. L'instance est IV' = ($V_i$, C_i, D_i, H'_i, R_i, CS_i, δ_i^{-1}) telle que $H'_i = (H_i \setminus \{h_i\} \bigcup \{h'_i\})$, où $h_i = (\bigcup_{l_i \in L} levelset(l_i), RUP)$ et $h'_i = (\bigcup_{l_i \in L \bigcup \{l_n\}} levelset(l_i), RUP')$. L'ensemble RUP' contient les fonctions $rup_{l_i}^{l_j}$ telles que $rup_{l_1}^{l_n} = \{(e, m) | \forall e \in levelset(l_1)\}$, $rup_{l_n}^{l_2} = \{(e, m) | \forall e \in levelset(l_2)\}$ et $rup_{l_i}^{l_j} = rup_{l_i}^{l_j}$ pour les autres niveaux l_i et l_j. □

Par exemple si nous voulons insérer le niveau `District` à la hiérarchie `H_Geo` de la figure 3.4 (*cf.* Chapitre 3) entre les niveaux `Departement` et `Region`, la primitive *AddL(VS$_i$, H_ Geo, District, Departement, Region, 'Columbia')* modifie le schéma et l'instance de la hiérarchie spécifiée. La figure 5.3 montre le résultat de cet opérateur.

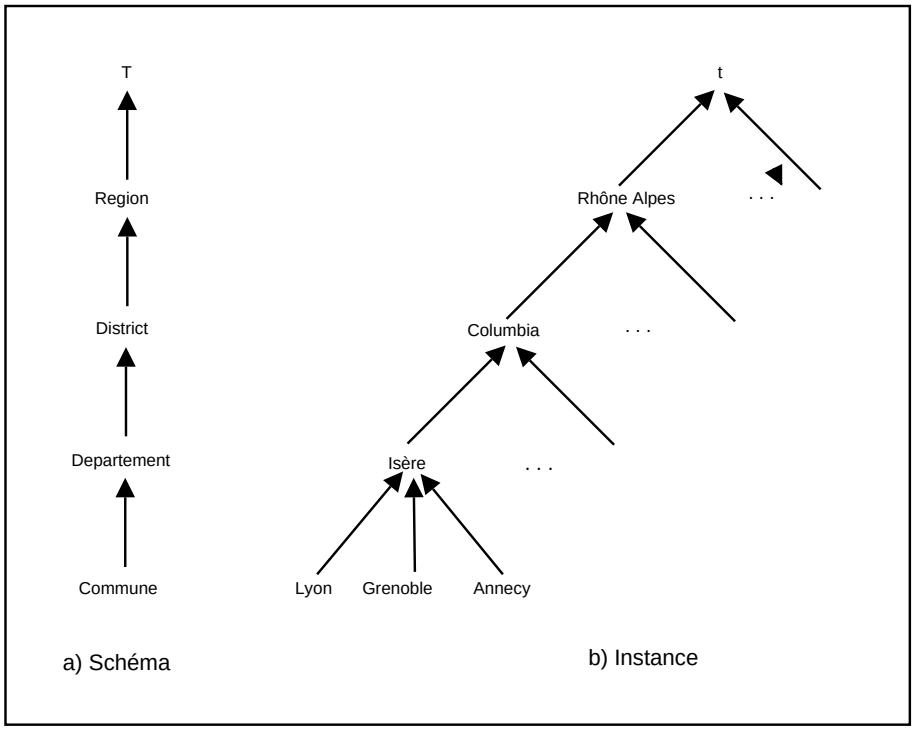

FIG. 5.3 – Insertion du niveau `District` à la Hiérarchie H_Geo.

Définition 5.18 : Primitive *DeleteL*

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), un nom de hiérarchie $h_n \in H$, et un niveau $l_n \in L$, tel que $l_n \neq \top$ et $l_n \neq l\bot$, la primitive DeleteL(VS, h_n, l_n) donne une version

dont le schéma est VS' = (V_s, C_s, D_s, H'_s, R, CS, δ^{-1}), où H'_s = ($H_s \setminus \{(h_n, L, \prec)\} \bigcup \{(h_n, L', \prec')\}$), où $L' = (L \setminus \{l_n\})$, $\prec' = (\prec \setminus \{(l_1, l_2) | l_n = l_1 \vee (l_n = l_2)\}) \bigcup \{(l_1, l_2) | l_1 \prec l_n \wedge l_n \prec l_2\}$. L'instance est IV' = ($V_i$, C_i, D_i, $H_i \setminus \{h_i\} \bigcup \{h'_i\}$, R_i, CS_i, δ_i^{-1}), où $h_i = (\bigcup_{l_i \in L}$ levelset(l_i), RUP) et $h'_i = (\bigcup_{l_i \in L'}$ levelset(l_i), RUP'). L'ensemble RUP' contient les fonctions $rup_{l_i}^{l_j}$ telles que (i) $rup_{l_1}^{l_2} = rup_{l_1}^{l_n} \circ rup_{l_n}^{l_2}$, si $l_1 \prec l_n \wedge l_n \prec l_2$, (ii) $rup_{l_1}^{l_2} = rup_{l_1}^{l_2}$, dans les autres cas. □

Par exemple si nous voulons effacer le niveau `Departement` à la hiérarchie `H_Geo` de la figure 5.3 entre les niveaux `District` et `Commune`, nous exécutons la primitive *DeleteL(VS_i, H_ Geo, Departement)*. La figure 5.4 montre le résultat de cet opérateur.

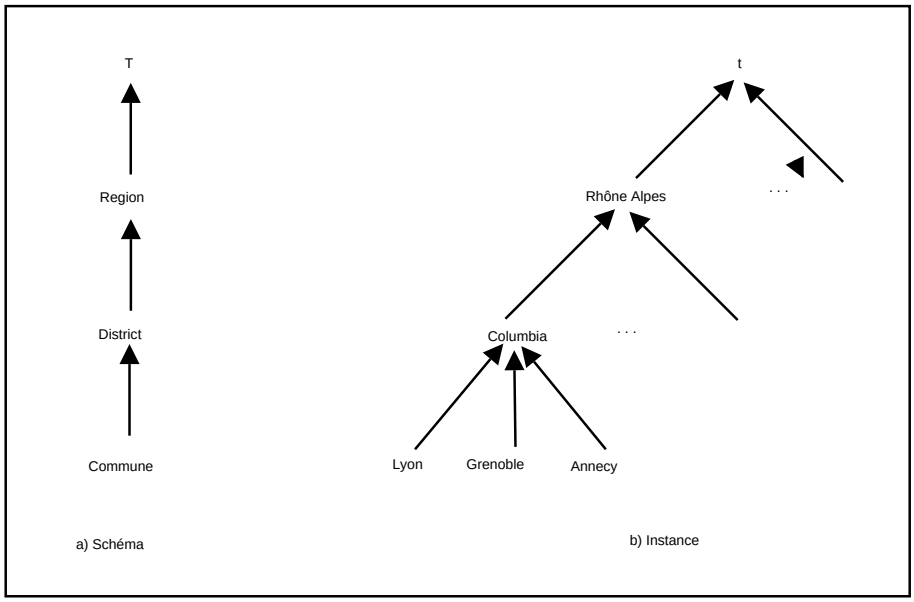

FIG. 5.4 – Résultat d'éliminer le niveau `Departement` à la Hiérarchie H_Geo.

Définition 5.19 : Primitive *RenameL*

Cette primitive change le nom d'un niveau, il prend en entrée le nom h_n de la hiérarchie, le niveau à renommer, l'ancien nom l_n et le nouveau nom l'_n.

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), un nom de hiérarchie $h_n \in H$, et les niveaux l_n et $l'_n \in L$, la primitive RenameL(VS, h_n, l_n, l'_n) crée une version dont le schéma est VS' = (V_s, C_s, D_s, H'_s, R, CS, δ^{-1}), où $H'_s = H_s \setminus \{(h_n, L, \prec)\} \bigcup \{(h_n, L \setminus \{l_n\} \bigcup \{l'_n\}, \prec)\}$, et l'instance est IV' = IV. □

Par exemple, pour renommer le niveau `Commune` de la hiérarchie `H_Geo` par le nom `Ville` : *RenameL(VS_i, H_ Geo, Commune, Ville).*

5.3.2 Gestion des versions

Nous présentons l'ensemble des définitions de primitives qui agissent sur la gestion de versions de schémas. Nous considérons l'existence de :

DOM = dom(char) $\bigcup$ dom(integer= $\bigcup$ dom(float) $\bigcup$ dom(decimal) $\bigcup$ dom(date) contenant le domaine de chaque type atomique.
OP = Ensemble de opérations primitives.

Définition 5.20 : Primitive *CreateV*

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), et un nom de version $v_n \in V$, le résultat de CreateV(VS, v_n) crée une nouvelle version dont le schéma est VS' = VS et l'instance est IV' = IV. □

Par exemple, supposons que nous voulons créer une version de schéma VS_j à partir de la version VS_i. La primitive *CreateV(VS_i, VS_j)* crée la version VS_j = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}).

Définition 5.21 : Primitive *FreezeV*

Cette primitive permet de geler une version (i.e., elle ne permet plus de changements). La seul primitive que nous pouvons appliquer à une version gelée est `DefrostV` (*cf.* Définition 5.22).

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), et un nom de version $v_n \in V$, la primitive FreezeV(v_n) bloque cette version en interdisant l'exécution des primitives d'évolution. La version de schéma est $VS_i = v_n$ dont le schéma est VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et l'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}). $\forall_{i=1,\ldots,n} p_i \in OP$, $p_i(VS_i)$ s'exécute si et seulement si VS_i n'est pas gelée. □

Définition 5.22 : Primitive *DefrostV*

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), et un nom de version $v_n \in V$, la primitive DefrostV(v_n) fournit une version de schéma $VS_i = v_n$ dont le schéma est VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et l'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}). $\forall_{i=1,\ldots,n} e_i \in E(VS_i)$ $p_i(VS_i)$ s'exécute, où $p_i \in OP$. □

Définition 5.23 : Primitive *ExporterR*

Cette primitive permet de faire migrer l'ensemble des données intensionnelles et extensionnelles d'une relation vers une nouvelle version de schéma.

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), un schéma de dimension $d_n \in D$ et un nom de version $v_n \in V$, le résultat de ExporterR(VS, d_n, v_n) est une version VS' dont le schéma est VS' = (V_s, C_s, $D_s \bigcup \{d_s\}$, H_s, R, CS, δ^{-1}), où d_s est un schéma de dimension (d_n, P, H), et l'instance est IV' = IV. □

Par exemple, pour faire migrer la dimension CIM10 qui appartient à la version VS_n, nous exécutons la primitive *ExporterR(VS_n, CIM10, VS'_n)*, où le schéma de la dimension CIM10 est (CIM10, {Cle_Maladie, Libelle_maladie}, {∅}).

Définition 5.24 : Primitive *DropV*

L'exécution de cette primitive élimine le schéma et les instances d'une base de données multidimensionnelle. Nous pouvons exécuter cette opération si la version de schéma n'existe pas dans le période historique établi. Ainsi, nous supposons l'existence d'une fonction booléenne β indiquant si la version de schéma peut être supprimée.

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), et un nom de version $v_n \in V$ tel que $\beta(v_n)$=vrai, la primitive DropV(VS_i), où $VS_i = v_n$, élimine la version de schéma SV_i ainsi que leurs instances. □

Définition 5.25 : Opérateur *SetVersion*

Etant donnés une version de schéma VS = (V_s, C_s, D_s, H_s, R, CS, δ^{-1}) et d'instance IV = (V_i, C_i, D_i, H_i, R_i, CS_i, δ_i^{-1}), un ensemble de primitives {EP} et les noms de versions v_n et $v'_n \in V$, la primitive SetVersion(VS_i, {EP}, VS_j), où $VS_i = v_n$ et $VS_j = v'_n$, crée la version de schéma VS_j dont le schéma est VS' = (V'_s, C'_s, D'_s, H'_s, R, CS, δ^{-1}) et l'instance est IV' = (V'_i, C'_i, D'_i, H'_i, R_i, CS_i, δ_i^{-1}). □

Si nous voulons créer une nouvelle version VS_j à partir de la version courante VS_i incluant les changements : {EP}=(DropC(VS_i, Prise_MCO), DropD(VS_i, Mode_sortie), ExporterR(VS_i, CIM10, VS_j), ExporterR(VS_i, Etablissement, VS_j), RenameP(VS_i, Etablissement, Codepostal, Code), ExporterR(VS_i, Temps, VS_j), DeleteP(VS_i, Temps, Saison)), l'exécution de *SetVersion (VS_i, {EP}, VS_j)* donne le résultat souhaité. Chaque primitive de l'ensemble EP doit respecter la définition donnée précédemment.

5.4 Gestion temporelle des entrepôts

Nous traitons dans cette section la gestion des données historisées de notre modèle. Nous montrons d'abord l'architecture conçue simplifiée du projet et l'interac-

tion entre leurs composants, puis nous présentons le modèle de versions de schémas bitemporels et nous terminons par la description du gestionnaire d'évolution de schémas.

5.4.1 Architecture simplifiée du projet

L'architecture que nous avons conçue pour le projet ADELEM est composée de trois composants principaux : l'interface graphique, l'entrepôt de données médicales et le gestionnaire d'évolution. La figure 5.5 montre l'interaction entre le gestionnaire de requêtes et le gestionnaire d'évolution.

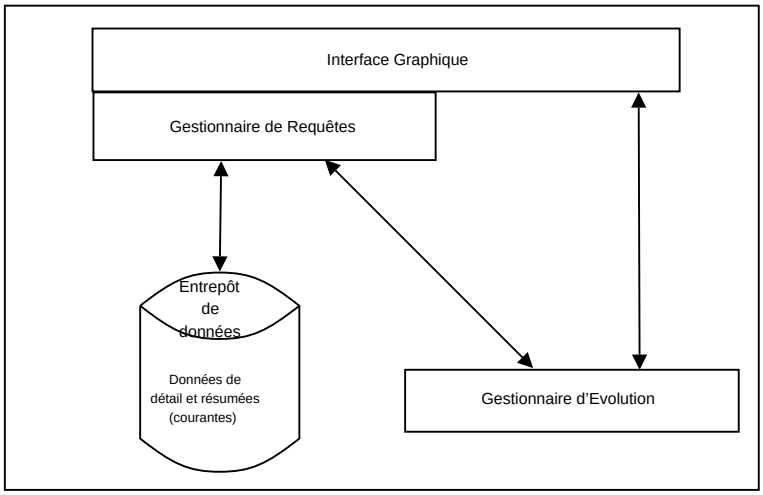

FIG. 5.5 – Interaction des composants à l'intérieur de l'architecture.

Le composant Interface Graphique contient un gestionnaire de requêtes qui interagit avec le gestionnaire d'évolution dans l'exécution de requêtes sur des schémas historisés. Nous donnons dans la suite une description de la gestion de versions de schémas.

5.4.2 Modèle de versions de schémas bitemporels

Le gestionnaire d'évolution est composé d'un entrepôt de données historisées et du moteur de versions. La figure 5.6 montre le modèle de versions de schémas bitemporels.

L'entrepôt de données historisées contient le schéma et les instances de chaque version de schéma hisorisée ainsi que la table de correspondance. Celle-ci est requise pour spécifier la pertinence temporelle entre les données intensionnelles de la version de schéma et les données extensionnelles. Le moteur de versions contient à la fois le stockage des deltas entre versions et les règles de configuration que nous devons respecter lors de la gestion des versions.

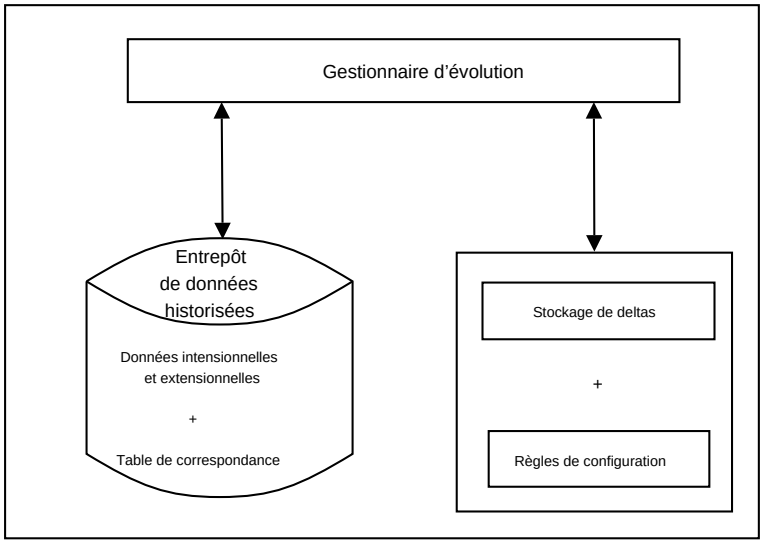

FIG. 5.6 – Modèle de versions de schémas bitemporels.

Les deltas représentent l'ensemble de différences entre versions. Un delta symétrique entre SV_i et SV_j = (SV_i - SV_j) $\bigcup$ (SV_j - SV_i) et un delta direct est une séquence de changements c_1, ..., c_n qui appliqués sur une version v_1 en génerent une nouvelle v_2 [CW98, RGMS01].

Notre gestionnaire de versions bitemporels est au-dessus du système de gestion de bases de données. Dans cette approche, le modèle de version est orthogonal au modèle de données du SGBD [CW98, WMC01]. Nous considérons le stockage de l'ensemble de changements qui produisent une version, pour cela, le méta-schéma se compose de :

Table pour la version de schéma : Identification de la version.

`TVS` = {`ID_Version, ID_Conteneur, Version_nom, Temps_transaction, Temps_validite, Version_immuable`}

L'attribut `Version_immuable` est de type booléen. Par défaut il est faux.

Table pour la relation :

`TR` = {`ID_Relation, ID_Version, Relation_nom, Temps_transaction, Temps_validite, Relation_immuable, Relation_type`}

L'attribut `Relation_type` spécifie les types : cube, dimension, hiérarchie, vue matérialisée, vue.

Table pour les attributs :

TA = {ID_Attribut, ID_Relation, Attribut_nom,
Attribut_type, Attribut_taille, Temps_transaction, Temps_validite,
Attribut_immuable, Key}

L'attribut Key est de type booléen et sert pour indiquer si ID_Attribut fait partie de la clé primaire.

Table d'équivalences :

TE = {ID_Attribut, ID_Relation, Attribut_nomancien,
Attribut_nomnouveau, Temps_transaction, Temps_validite, Attribut_immuable}

Cette table stocke l'équivalence entre attributs renommés.

Les règles de configuration contiennent l'ensemble des règles d'intégrité déjà définies, néanmoins, nous pouvons aussi en définir de nouvelles, par exemple :

$$version_{courante} = \delta^{-1}(VS_i) = maximum$$

Cette règle établit que la version courante sera celle dont la pertinence temporelle est maximum.

Nous avons dit précédemment que la requête est exécutée sur les données courantes. Néanmoins, pour les requêtes sur des données historisées, le composant Interface Graphique interagit avec le composant Gestionnaire d'évolution pour déterminer la possibilité d'exécution de la requête.

5.4.3 Gestionnaire d'évolution de schémas

Dans cette partie, nous traitons la gestion de données historisées et l'activité du gestionnaire d'évolution de schémas par niveau. La figure 5.7 montre le détail de la gestion des données historiques.

Nous considérons une base de données pour stocker le méta-schéma et un ensemble de bases de données historiques pour enregistrer les données intensionnelles et extensionnelles ainsi que la table de correspondance (TC) de chaque version de schéma. Cette table stocke la pertinence temporelle des données extensionnelles par rapport à la pertinence temporelle de leurs données intensionnelles.

La figure 5.8 décrit l'activité du gestionnaire d'évolution par niveau.

Le niveau 3 stocke l'ensemble des changements des versions de schéma et l'ensemble des règles de configuration lors de la création et de la manipulation des versions. A l'intérieur de ce niveau, la pertinence temporelle de requêtes d'analyses ou de comparaison sur des données courantes et historisées est déterminée. Par

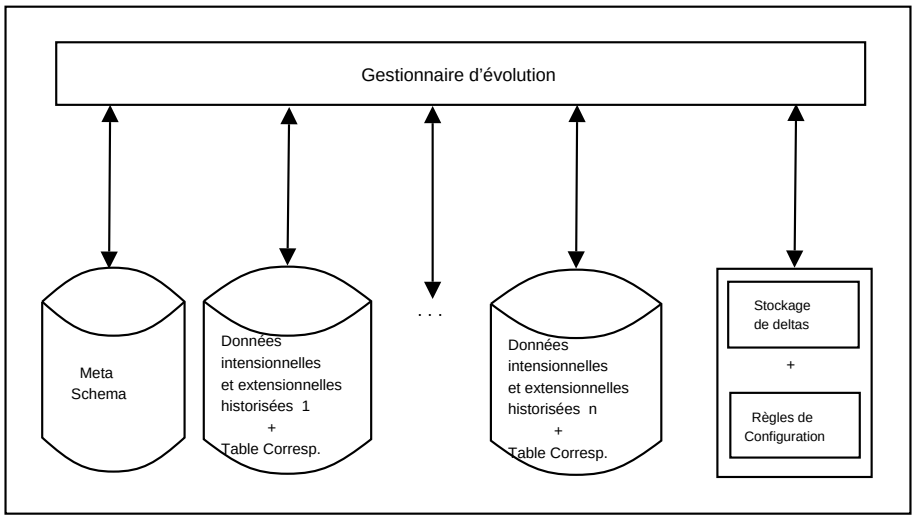

FIG. 5.7 – Gestionnaire d'évolution de schémas.

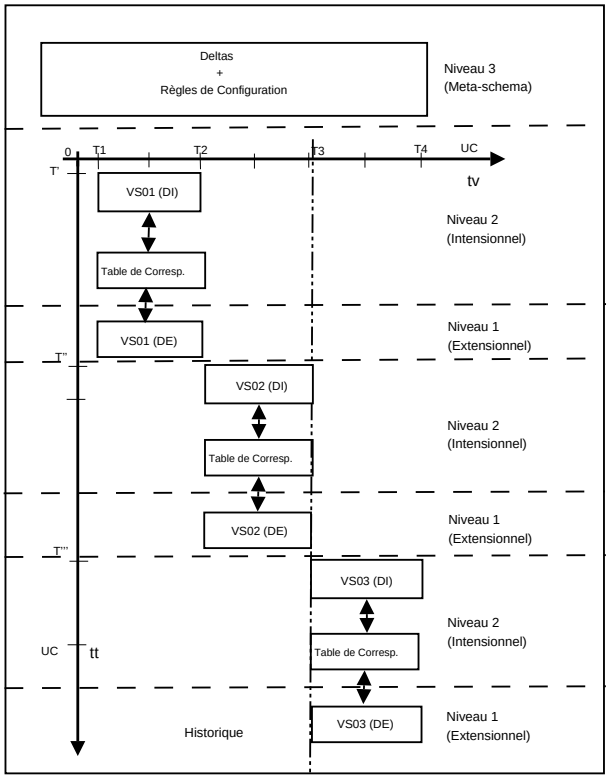

FIG. 5.8 – Gestion d'évolution de schémas par niveau.

exemple, si le méta-schéma de la requête sur la version de schéma courant coïncide avec celui de la version de schéma historique, nous pouvons exécuter la requête sur les deux versions de schémas. Dans le cas contraire, la requête est annulée et l'utilisateur doit être informé. Les niveau 1 et 2 représentent les différentes versions de données (extensionnelles et intensionnelles) de façon linéaire par rapport au temps de création. La table de correspondance fait partie du niveau intensionnel, dans la suite, nous décrivons son fonctionnement.

Table de Correspondance :

Nous avons placée dans le niveau 2 la table de correspondance nécessaire pour la pertinence temporelle entre l'ensemble de données. Le schéma de la table est :

{`ID_Table, ID_Version, ID_Conteneur, Cle_Temps, Temps_transaction, Temps_validite`}.

Ceci nous permet de stocker les données extensionnelles sans leur ajouter les propriétés `Temps_transaction` et `Temps_validite`. Etant donné la taille de l'ensemble des données extensionnelles, cette table nous permet d'avoir un gain très considérable sur la quantité des données à stocker.

Par exemple, en utilisant la syntaxe du langage TSQL2 [Sno95], nous pouvons exprimer les requêtes suivantes :

SET SCHEMA DATE 'now'
SELECT * FROM Prise_MCO
WHERE VALID (Prise_MCO) OVERLAPS PERIOD '2003-01 - 2003-12'
ou
SET SCHEMA TRANSACTION DATE '2004-07'
SELECT * FROM Prise_MCO
WHERE VALID (Prise_MCO) OVERLAPS PERIOD '2003-01 - 2003-12'

Nous avons le même résultat pour les deux requêtes, car elles sont exécutées sur la version de schéma courant, aussi bien sur les données intensionnelles qu'intensionnelles. Au contraire, la requête :

SET SCHEMA VALID PERIOD '2002-01 - 2002-12'
SELECT * FROM Prise_MCO
WHERE VALID (Prise_MCO) OVERLAPS PERIOD '2002-01 - 2002-12'

prend des données de la table `Prise_MCO` qui ont été valides durant la période 2002-01 - 2002-12, cette requête utilise le schéma valide durant la même période du temps. Dans la suite, nous montrons un exemple plus complexe :

```
SET SCHEMA VALID PERIOD '2003-01 - 2003-12'
SELECT * FROM Prise_MCO
WHERE VALID (Prise_MCO) OVERLAPS PERIOD '2002-01 - 2002-12'
```

La requête sélectionne le schéma valide entre 2003-01 et 2003-12 qui correspond, dans notre cas, au schéma courant. Les données valides entre 2002-01 et 2002-12 correspondent aux données de la version historisée VS02.

5.5 Bilan

Dans un entrepôt de données, la manipulation des données historisées est essentielle. Néanmoins, leur gestion est une tâche complexe et coûteuse. Dans ce chapitre, nous avons présenté notre proposition pour la gestion, le stockage et la visualisation de l'ensemble de données historisées. Nous avons introduit d'abord un ensemble de versions de schémas annuels dans un contexte médical. Ce schéma est composé d'une étoile avec quatre dimensions présentant divers changements au cours du temps. En considérant ces changements, nous avons crée un ensemble de versions de schémas qui nous permet la manipulation des données sans perte. Nous avons aussi donné une représentation bitemporelle pour chaque version de schéma.

Pour la création et la gestion des versions de schémas, nous avons spécifié une liste d'opérations primitives divisée en deux catégories, la première rassemble un ensemble de 19 primitives nécessaires pour la gestion des cubes, des dimensions et des hiérarchies. La deuxième contient 5 primitives qui agissent sur les versions de schémas. Nous avons aussi défini l'opérateur *SetVersion* qui permet de grouper un ensemble de primitives à exécuter sur une version de schéma et qui génère une nouvelle version. De cette manière, nous pouvons contrôler le nombre de versions générées. Nous avons aussi défini un ensemble de contraintes pour la manipulation des versions. La sémantique de l'ensemble de primitives doit respecter cet ensemble de contraintes pour assurer une gestion cohérente.

Pour chaque primitive, nous avons donné sa définition formelle. Pour ce faire, nous avons repris la définition d'une base de données multidimensionnelle, d'un cube, d'une dimension, d'une hiérarchie ainsi que de leurs instances données au Chapitre 3. La définition formelle de la primitive considère aussi bien son schéma que ses instances.

La dernière section a traité la gestion temporelle des entrepôts de données. Nous avons donné une architecture simplifiée qui décrit l'interaction entre gestionnaires de requêtes et d'évolution. Nous avons présenté notre modèle de versions de schémas bitemporels composé d'un entrepôt de données historisées et d'un moteur de versions stockant les deltas et les règles de configuration. Nous avons donné la description du gestionnaire d'évolution de schémas. Nous avons présenté aussi bien la gestion de données historisées que l'activité du gestionnaire d'évolution par niveau. Le niveau

1 décrit la gestion des données extensionnelles, le niveau 2 présente la manipulation des données intensionnelles et le niveau 3 montre la gestion des deltas ainsi que des règles de configuration. Finalement, nous avons terminé cette partie avec quelques exemples de requêtes en utilisant la syntaxe du langage TSQL2 pour spécifier la version de schéma souhaitée.

Chapitre 6

Conclusions et perspectives

6.1 Bilan du travail réalisé

Les entrepôts de données étendent les concepts et la technologie traditionnelle des bases de données pour créer des systèmes d'aide à la décision. Ils intègrent des informations provenant des sources de données qui sont souvent hétérogènes et distribuées. En conséquence, la conception et la mise en oeuvre d'un entrepôt est une tâche complexe, elle se compose de trois processus : extraction-intégration, organisation et interrogation. Notre proposition se place à l'intérieur des processus d'organisation et d'interrogation.

6.1.1 Principales contributions

Nous listons nos contributions pour la conception et la mise en oeuvre d'un entrepôt de données :

- Définition d'un modèle multidimensionnel.

- Traitement de requêtes et algorithme pour la sélection optimale de vues à matérialiser.

- Interface graphique pour la génération (semi-automatique) des requêtes.

- Versions de schémas bitemporels pour la gestion de données médicales historisées.

6.1.2 Définition d'un modèle multidimensionnel

Notre métamodèle multidimensionnel se compose de trois classes : Cube, Dimension et Hiérarchie. Ce métamodèle sert à la conception d'un modèle multidimensionnel. Pour faire cela, nous avons donné quatre définitions : le schéma d'une base multidimensionnelle, d'un cube, d'une dimension et le schéma d'une hiérarchie.

Ceci nous a permis d'aboutir à la conception d'un schéma en constellation pour la construction d'un entrepôt multidimensionnel de données médicales. Il se compose de trois sous-schémas en étoile : `Prise_MCO, Prise_SSR` et `Population`, où chaque étoile contient ses propres dimensions ainsi que des dimensions qui sont partagées entre l'ensemble d'étoiles. Nous avons identifié deux hiérarchies : `H_Geo` et `H_Temps`. Finalement, nous avons utilisé les informations concernant l'ensemble des sources de données réelles et des indicateurs du projet ADELEM pour prouver et vérifier le modèle proposé.

Tout au long de notre travail, nous avons été confrontés à plusieurs difficultés. La principale a été la conception du modèle multidimensionnel. Pour nous, cette étape a été très dure, même avec l'aide d'une méthodologie pour sa réalisation. Ceci dû principalement à la manque d'un métamodèle qui puisse orienter les choix des différents éléments (cubes, dimensions et hiérarchies) et ses relations, mais aussi, en raison de la nature sensible de données médicales. C'est la raison pour laquelle nous avons proposé un métamodèle dont la caractéristique fondamentale est sa simplicité. Notre objectif est de normaliser et de faciliter dans la mesure du possible le développement de cette phase.

6.1.3 Algorithme pour la sélection des vues à matérialiser

Nous avons appliqué notre algorithme et celui de Greedy aux données du projet ADELEM. L'algorithme Greedy est simple et efficace, néanmoins, dans notre expérimentation, à partir du 6ème choix, l'algorithme sélectionne les vues les plus coûteuses. Ceci nous a motivé pour proposer un algorithme qui étend Greedy avec les paramètres : fréquence d'utilisation, coût de calcul et fréquence des mises à jour sur les relations de base.

Nous avons remarqué que plusieurs propositions utilisent comme fréquence d'utilisation un nombre assigné à chaque vue, par exemple : la fréquence d'utilisation pour la vue V1 est égale à 10, V2 à 5, V3 à 1, ... Ainsi, la particularité de notre algorithme réside dans la spécification de deux propositions. La première consiste à utiliser la complexité de la vue (nombre de ses relations dépendantes) pour les deux premiers paramètres : la fréquence d'utilisation et le coût de calcul de la vue. Ainsi, nous considérons que la fréquence d'utilisation d'une vue augmente en proportion directe du nombre de ses relations dépendantes et que le coût de calcul diminue dans la même proportion. La deuxième proposition consiste à déterminer la fréquence des mises à jour sur les relations de base, dans ce cas, nous avons besoin du nombre d'éléments à l'intérieur de la vue qui peuvent subir des changements. Nous avons constaté que notre proposition donne de meilleurs résultats dans notre cas expérimental.

La faiblesse de notre algorithme est l'absence du coût d'évaluation d'une requête par rapport au type d'opération (*Select, Project* ou *Join*). Nous ne considérons pas

des restrictions comme l'espace de stockage.

6.1.4 Génération (semi-automatique) des requêtes

Le développement de notre interface graphique est dans sa première version. Nous avons développé deux modules : le module pour la création et la mise à jour du schéma et le module pour la génération (semi-automatique) des requêtes. Le premier module nous permet d'adapter l'interface à tout schéma, ainsi nous avons seulement besoin d'interroger les données correspondantes à ce schéma en utilisant le deuxième module.

Nous avons présenté nos résultats aux participants du projet ADELEM, ce qui nous a permis de vérifier et de prouver le fonctionnement de l'interface.

6.1.5 Versions de schémas bitemporels

Les actualisations réalisées dans les sources des données doivent être reportées sur l'entrepôt. Ces changements peuvent affecter aussi bien les données que leur structure. Pour sa gestion, nous pouvons utiliser l'évolution de schémas ou les versions de ceux-ci.

En ce qui concerne l'évolution de schémas, elle permet la récupération des données existantes par le biais de leur adaptation au nouveau schéma. Néanmoins, l'évolution n'implique pas un support historique de celui-ci et elle ne considère non plus un mécanisme pour la visualisation des données à travers des schémas évolués. Par conséquence le concept d'évolution de schéma n'est pas suffisant et il est nécessaire d'établir une stratégie spécifique pour répondre à cette nécessité.

L'approche des versions de schéma a été utilisée principalement pour la cohérence des données et de leurs schémas courants. Nous considérons cette approche comme une alternative aux systèmes qui doivent gérer et manipuler des données historiques. Dans le cas précis du projet ADELEM, nous proposons l'utilisation de versions de schéma pour gérer les aspects temporels de l'entrepôt de données.

La principale caractéristique de notre proposition consiste à fournir une approche simple et adaptée au paradigme des entrepôts pour la gestion et la manipulation aussi bien des données historiques que courantes. En plus, à la différence des systèmes temporels, où l'ensemble de données doit contenir la notion du temps, les données extensionnelles dans notre approche sont stockées telles quelles.

Bien que nous nous sommes inspirés des caractéristiques et des besoins du projet ADELEM, notre approche de versions peut être utilisée par tout système qui doit gérer des données historisées. C'est la raison pour laquelle nous avons décidé d'uti-

liser les temps de transaction et de validité et de fournir l'ensemble de primitives définies.

6.2 Perspectives

Nous avons un ensemble de perspectives permettant de continuer le travail présenté ainsi que son amélioration. Nous avons dit précédemment que nous avons seulement développé une version initiale de l'interface, ainsi, nous décrivons d'abord l'extension du prototype réalisé. Nous donnons aussi une autre perspective pour la gestion des données, celle d'inclure les aspects spatiaux aux données médicales. Finalement, nous incluons d'autres perspectives qu'il est intéressant de prendre en compte.

6.2.1 Extension du prototype

Nous avons divisé ce travail en deux parties : l'implantation de l'algorithme pour la sélection des vues à matérialiser et les versions de schémas pour la gestion de données historisées.

6.2.1.1 Algorithme proposé

Le développement de l'algorithme a deux objectifs : disposer d'un mécanisme pour la sélection automatique de l'ensemble de vues à matérialiser et récupérer par le biais de leur utilisation, des statistiques réelles pour améliorer et perfectionner les paramètres initiaux que nous avons établis. De cette manière, nous pouvons collecter la fréquence d'utilisation réelle des vues matérialisées. Ceci nous permettra de proposer de nouvelles vues pour leur matérialisation ou d'en effacer d'autres. Une autre statistique à collecter est la fréquence réelle des mises à jour des relations de base, ce qui permettra d'adapter la probabilité de changement à appliquer.

L'utilisation des agrégats est l'outil le plus efficace pour améliorer les performances. Néanmoins, nous devons établir un mécanisme permettant l'utilisation des vues matérialisées dans l'exécution d'une requête. Une idée assez simple consiste à ordonner l'ensemble de vues matérialisées par rapport à leur coût de stockage. De cette manière, si la requête peut être exécutée en utilisant la première vue, nous arrêtons le processus d'optimisation. Dans le cas contraire, nous prenons la deuxième vue de l'ensemble et nous continuons le processus. Si nous arrivons à la fin de l'ensemble, nous devons exécuter la requête en utilisant les relations de base.

Finalement, nous considérons comme important les problèmes de la gestion des index et la possibilité d'utiliser un cache pour l'optimisation des requêtes.

6.2.1.2 Versions de schémas bitemporels

L'implantation des versions de schémas permettra l'exécution des requêtes sur les données historisées. Ainsi, nous pourrons faire des calculs prévisionnels ou des requêtes comparatives qui comportent des données aussi bien courantes que historisées. Par exemple, si le méta-schéma de la requête sur la version de schéma courant coïncide avec celui de la version de schéma historique, nous pouvons exécuter la requête sur les deux versions de schémas. Pour faire cela, le gestionnaire d'évolution interagit avec le gestionnaire de requêtes pour déterminer la possibilité d'exécution de cette requête.

Notre gestionnaire de versions bitemporels est au-dessus du système de gestion de bases de données. De cette manière, le modèle de version est orthogonal au modèle de données du SGBD. Cette caractéristique nous permet d'adapter notre modèle de versions à tout SGBD et pouvoir faire le choix par rapport aux besoins de l'application cible.

6.2.2 Aspects spatiaux des données

L'information géographique décrit des phénomènes qui adviennent sur, au dessus, et en dessous de la surface de la terre [Sch01]. Les cartes sont un des supports de cette information. Une carte contient des objets géographiques tels que des parcelles d'occupation des sols, des routes, des territoires communaux, etc., dans une région géographique donnée. Un objet géographique a des propriétés de deux types :

- Une propriété spatiale représentée par un attribut géométrique ou encore attribut spatial. Cet attribut décrit l'emplacement d'un objet dans l'espace à deux ou trois dimensions.

- Des propriétés qui décrivent l'objet représentées par des attributs dits thématiques ou descriptifs.

La définition suivante du Système d'Information Géographique (SIG) est généralement admise :

Un système d'information géographique est un ensemble organisé de matériels informatiques, de logiciels, de données géographiques et de personnel capable de saisir, stocker, mettre à jour, manipuler, analyser et présenter toutes formes d'informations géographiques référencées.

La nature des données stockées et exploitées dans les SIGs entraîne beaucoup de problèmes, tant au niveau modélisation et conception qu'au niveau gestion au sein d'un même système, du fait de la complexité de stockage, de consultation et de présentation [Dbo97].

La diversité de ces données est une question critique : données textuelles, données spatiales de type vectoriel telles que les données géométriques, de type matriciel (ou rasteur) comme les données cartographiques, et données multimédia, par exemple les images satellitales et aériennes, les données audio et vidéo. Ces données proviennent de sources diverses et sous différentes échelles et formats de stockage ou d'échange.

La représentation des données dans les SIGs :

Tous les SIGs utilisent trois primitives, le point, la ligne, et la région pour construire la géométrie d'un objet. Deux choix sont à faire pour représenter cette géométrie [Sch01] :

1. Quelle primitive utiliser (point, ligne ou région) ?
2. Quel mode rasteur ou vecteur dans chacun des cas ?

La description d'un terrain à partir de données de source satellitaire ou provenant d'images aériennes est faite dans un mode représentation discrète appelé rasteur, au moyen d'une grille : le plan est soit divisé en un nombre fini de points soit décomposé en un nombre fini de cellules régulières. Une ligne ou une région peut être représentée en mode rasteur par des cellules connexes. Plus la résolution de la grille est grande, meilleure est l'approximation obtenue.

La plupart des systèmes actuels utilisent le mode vecteur : les points sont représentés par leurs coordonnées, les lignes sont représentées par des segments de droite qui sont des approximations linéaires par morceaux, aussi appelées polylignes. Quant aux régions elles sont représentées par une approximation polygonale de leur frontière.

La figure 6.1 montre les deux modes pour représenter la géométrie d'un objet.

Aujourd'hui l'utilisation des SIGs et des techniques de cartographie dans le domaine de la santé en Santé Publique se développe rapidement. Les ressources de Santé Publique, les maladies spécifiques et les événements sanitaires peuvent être représentés et cartographiés en fonction de leur environnement et de l'existence d'infrastructures sanitaires et sociales. De telles informations, quand elles sont cartographiés ensemble, créent un outil puissant pour gérer et contrôler les priorités en matière de Santé Publique. Le SIG fournit un sens excellent de l'analyse de données épidémiologiques et programmées, révélant les tendances, les dépendances et les corrélations, ce qui est plus difficile à mettre en valeur lorsque ces données sont sous forme de tableau.

Finalement, il est intéressant de faire un couplage entre l'architecture conçue pour ADELEM et un SIG. Pour faire cela, nous pouvons utiliser le logiciel HealthMapper, ceci nous permettra d'afficher les données sélectionnées au travers de leur

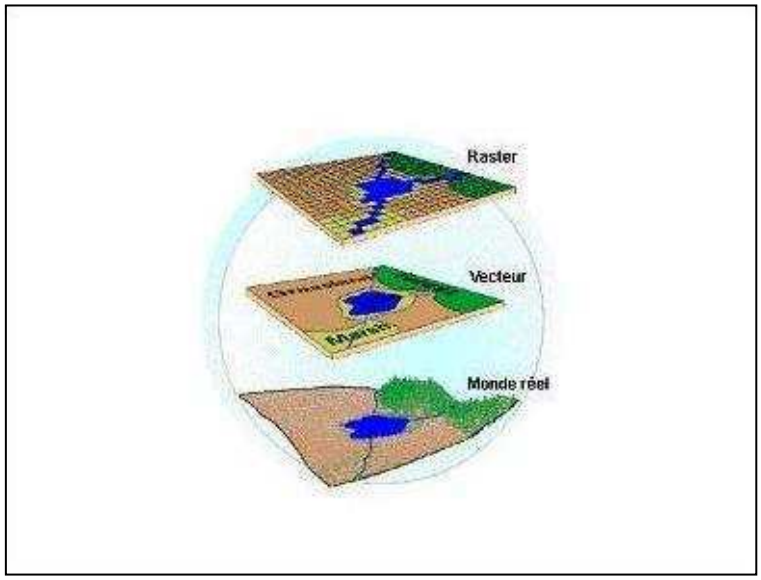

FIG. 6.1 – Représentation du mode vecteur et rasteur

représentation cartographique.

6.2.3 Construction et rafraîchissement de données

Le processus d'extraction-intégration est fondamental dans la conception d'un entrepôt de données, car les données fournies seront stockées à l'intérieur de l'entrepôt. Pour cela, nous pensons qu'il est intéressant de prendre en compte le développement de ce processus, due au fait que nous pouvons mesurer la qualité de l'entrepôt par le biais de la qualité de ses données. La principale problématique est le suivi des changements dans les sources de données et leur propagation vers l'entrepôt pour le mettre à jour. Ce processus requiert une transaction de maintenance. Dans les entrepôts de données, une problématique très importante est la façon d'exécuter les transactions de maintenance sans perturber les requêtes des utilisateurs.

La figure 6.2 montre l'architecture proposée pour le projet WHIPS à l'Université de Stanford [HGMW$^+$95, Wid95].

Adaptateur/Moniteur : Il est associé à chaque source de données et il a deux fonctionnes : transformer le schéma et ses instances en une représentation intermédiaire et détecter automatiquement les changements dans la source pour les propager vers l'intégrateur.

Intégrateur : Il est responsable de la réception des représentations intermédiaires des données sources, en provenance des adaptateurs, et de l'intégration de celles-ci. L'intégration entraîne aussi une phase de transformation, celle-ci consiste à les préparer (mise en correspondance des formats de données), les nettoyer, les filtrer,..., pour les rendre conforme au schéma de l'entrepôt et aux critères de qualité choisis.

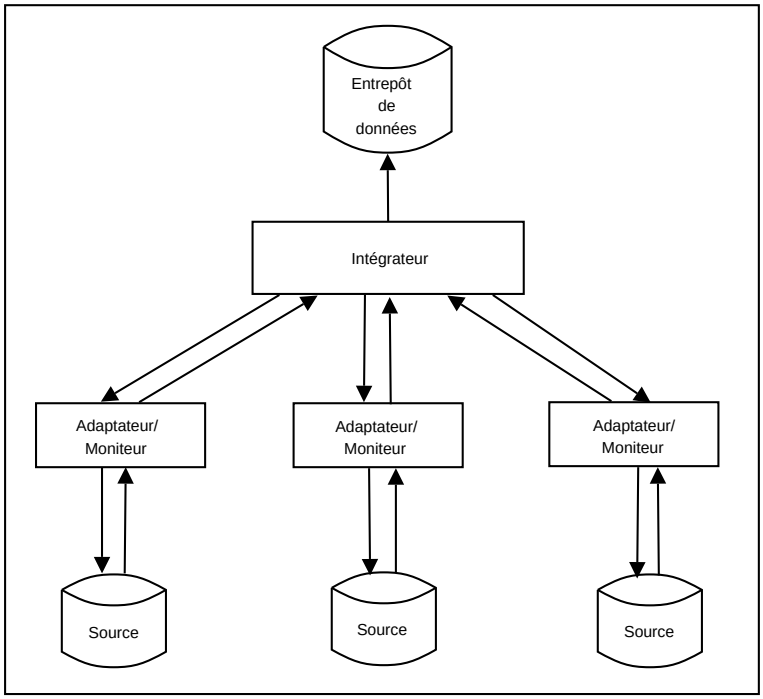

FIG. 6.2 – Architecture des systèmes pour l'intégration de données

Enfin, la nature diverse des sources oblige à programmer un adaptateur/moniteur spécifique pour chaque type de source. Du même pour chaque entrepôt de données, dont l'activité de l'intégrateur peut être spécifique. Ainsi, il est important de proposer le développement de techniques et d'outils pour la génération automatique d'adaptateurs/moniteurs et d'intégrateurs à partir de spécifications déclaratives [Ben02].

6.2.4 Grille de données

La dernière perspective que nous envisagions est la technologie de la Grille. Elle vise à fournir une infrastructure globale qui permet un partage de ressources coordonnées, flexibles, distribuées et sécurisées [Hea05]. Par exemple, dans le milieu médical [Med05], la Grille offre :

- La puissance de calcul pour des algorithmes complexes.

- Une infrastructure distribuée qui permet le partage de ressources aux différents participants médicaux ayant des besoins similaires de traitement de données.

- Une architecture commune pour l'accès des données hétérogènes.

- La possibilité de gérer d'énormes bases de données (par exemple, pour des études épidémiologiques).

En raison de la nature sensible de données médicales, il est important de prendre en compte des problèmes propres à leur confidentialité, tels que : authentification des utilisateurs, spécification de leurs droits d'accès, cryptage ou suppression des données relatives au patient pour la transmission sur le réseau, ...

De cette manière, nous pouvons créer un vaste réseau dans lequel il sera possible de questionner un système de données ayant une vue globale sur toute cette infrastructure.

Le datawarehouse n'est qu'une étoile de plus dans le firmament des technologies [Gog04].

Bibliographie

[AAD+96] Sameet Agarwal, Rakesh Agrawal, Prasad Deshpande, Ashish Gupta, Jeffrey F. Naughton, Raghu Ramakrishnan, and Sunita Sarawagi. On the Computation of Multidimensional Aggregates. In *VLDB*, pages 506–521, 1996.

[ABCM99] Ulf Asklund, Lars Bendix, Henrik Baerbak Christensen, and Boris Magnusson. The Unified Extensional Versioning Model. In *System Configuration Management*, pages 100–122, 1999.

[Adi02] M. Adiba. Entrepôts de données et fouille de données, Introduction, Supports de cours, 2002.

[AGS97] Rakesh Agrawal, A. Gupta, and Sunita Sarawagi. Modeling Multidimensional Databases. In Alex Gray and Per-Åke Larson, editors, *Proc. 13th Int. Conf. Data Engineering, ICDE*, pages 232–243. IEEE Computer Society, 7–11 1997.

[AQ86] Michel E. Adiba and N. Bui Quang. Historical Multi-Media Databases. In *VLDB '86 : Proceedings of the Twelfth International Conference on Very Large Data Bases*, pages 63–70. Morgan Kaufmann Publishers Inc., 1986.

[BB03] X. Baril and Z. Bellahsène. Selection of Materialized Views : A Cost-Based Approach. In *CAiSE*, pages 665–680, Bethesda, Maryland, USA, 2003. J. Eder and M. Missikoff.

[BCA01] E. Benitez, C. Collet, and M. Adiba. Entrepôts de données : caractéristiques et problématique. *Revue TSI*, 20(2), 2001.

[Bel02] A. Belot. Développement d'un logiciel de cartographie de l'offre et de la demande de soins hospitaliers en france. Technical report, 2002.

[Ben02] E. Benitez. *Infrastructure adaptable pour l'évolution des entrepôts de données*. PhD thesis, Université Joseph Fourier, Grenoble, France, Septembre 2002.

[Ber03] Philip Bernstein. Applying Model Management to Classical Meta Data Problems. In *CIDR Conference on Innovative Data Systems Research*, 2003.

[Blo99] Bloor Research - http ://www.bloor-research.com/, 1999.

[BPT97] Elena Baralis, Stefano Paraboschi, and Ernest Teniente. Materialized Views Selection in a Multidimensional Database. *The VLDB Journal*, pages 156–165, 1997.

[BSH98] J.W. Buzydlowski, I-Y. Song, and L. Hassell. A Framework for Object-Oriented On-Line Analytic Processing. In *ACM First International Workshop on Data Warehousing and OLAP*, Bethesda, Maryland, USA, November 1998.

[CD97] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1) :65–74, 1997.

[CGJM01] W. Cellary, S. Gangarski, G. Jomier, and M. Manouvrier. *Les versions, Chapitre 8 du livre : Bases de Données et Internet, Modèles, langages et systèmes.* Editions Hermès, 2001.

[CHJM02] D. Coadic, F. Hertout, Y. Julien, and C. Murgale. PFE : Méthodologie d'Analyse Décisionnelle. EISTI, 2002.

[CHS02] Rada Chirkova, Alon Y. Halevy, and Dan Suciu. A formal perspective on the view selection problem. *The VLDB Journal*, 11(3) :216–237, 2002.

[CLF99] G. Chan, Q. Li, and L. Feng. Design and Selection of Materialized Views in Data Warehousing : A Case Study. In *ACM International Workshop on Data Warehousing and OLAP*, 1999.

[Cod93] E.F. Codd. Providing OLAP (On-Line Analytical Processing) to user-analystes : an IT mandate. Technical report, 1993.

[CT97] Luca Cabibbo and Riccardo Torlone. Querying Multidimensional Databases. In *Workshop on Database Programming Languages*, pages 319–335, 1997.

[CT98] L. Cabibbo and R. Torlone. A Logical Approach to Multidimensional Databases. In *EDBT '98 : Proceedings of the 6th International Conference on Extending Database Technology*, pages 183–197. Springer-Verlag, 1998.

[CW98] R. Conradi and B. Westfechtel. Version Models for Software Conguration Management. *ACM Computing Surveys*, 30(2) :232–282, June 1998.

[DB204] DB2 OLAP Server - http ://www-306.ibm.com/software/data/db2/db2olap/, 2004.

[Dbo97] Mohamed Dbouk. *HyperGéo : Conception et réalisation d'un Système d'Information Géographique Hypermédia.* PhD thesis, Université Paris XI Orsay, Paris, France, Janvier 1997.

[DG01] A. Doucet and S. Gangarski. *Entrepôts de données et Bases de Données Multidimensionnelles, Chapitre 12 du livre : Bases de Données et Internet, Modèles, langages et systèmes.* Editions Hermès, 2001.

[DGS95] C. DeCastro, F. Grandi, and M. R. Scalas. On Schema Versioning in Temporal Databases. In S. Clifford and A. Tuzhilin, editors, *Recent Advances in Temporal Databases*, pages 272–294, Zurich, Switzerland, 1995. Springer Verlag.

[DGW+98] Curtis Dyreson, Fabio Grandi, Wolfgang, Nick Kline, Nikos Lorentzos, Yannis Mitsopoulos, Angelo Montanari, Daniel Nonen, Elisa Peressi, Barbara Pernici, John F. Roddick, Nandlal L. Sarda, Maria Rita Scalas, Arie Segev, Richard T. Snodgrass, Mike D. Soo, Abdullah Tansel, Paolo Tiberio, and Gio Wiederhold. A consensus glossary of temporal database concepts. *SIGMOD Rec.*, 23(1) :52–64, 1998.

[DSBH99] B. Dinter, C. Sapia, M. Blaschka, and G. Höfling. OLAP Market and Research : Initiating the Cooperation. *Computer Science and Information Management*, 2(3), 1999.

[Dum00] M. Dumas. *TEMPOS : une plate-forme pour le développement d'applications temporelles au dessus de SGBD à objets.* PhD thesis, Université Joseph Fourier, Grenoble, France, Juin 2000.

[EJWG03] Jr. E. James Whitehead and Dorrit Gordon. Uniform Comparison of Configuration Management Data Models. In *Software Configuration Management*, pages 70–85, 2003.

[Evo97] Projet Evolution - http ://www.prism.uvsq.fr/recherche/themes/ anciens/teams/dataware/coop/evolution.html, 1997.

[FGM00] E. Franconi, F. Grandi, and F. Mandreoli. Schema Evolution and Versioning : a Logical and Computational Characterisation. In *9th Intl. Workshop on Foundations of Models and Languages for Data and Objects : Database schema evolution and meta-modeling (DEMM'00)*, September 2000.

[Fra97] J-M Franco. *Le Data Warehouse (Le Data Mining).* Editions Eyrolles, Paris, 1997.

[GBLP95] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube : A Relational Aggregation Operator Generalizing Group-By, Cross-Tab and Sub-Totals. Technical Report Microsoft Technical Report No. MSR-TR-95-22., 1995.

[GM95] A. Gupta and I. Mumick. Maintenance of Materialized Views : Problems, Techniques and Applications. *IEEE Quarterly Bulletin on Data Engineering ; Special Issue on Materialized Views and Data Warehousing*, 18(2) :3–18, 1995.

[GM99] Fabio Grandi and Federica Mandreoli. ODMG Language Extensions for Generalised Schema Versioning Support. In *ER Workshops*, pages 36–47, 1999.

[GM00] A. Gutiérrez and A. Marotta. An Overview of Data Warehouse Design Approaches. 2000.

[GM02] Fabio Grandi and Federica Mandreoli. A Formal Model for Temporal Schema Versioning in Object-Oriented Databases. Technical report, 2002.

[GMS00] Fabio Grandi, Federica Mandreoli, and Maria Rita Scalas. A Generalized Modeling Framework for Schema Versioning Support. In *Australasian Database Conference*, pages 33–40, 2000.

[Gog04] Jean-François Goglin. *Le datawarehouse, pivot de la relation client.* Hermès Science, 2004.

[Gup97] Himanshu Gupta. Selection of Views to Materialize in a Data Warehouse. In *ICDT*, pages 98–112, 1997.

[Hea05] Healthgrid - http ://www.healthgrid.org/, 2005.

[HGMW+95] J. Hammer, H. Garcia-Molina, J. Widom, W. Labio, and Y. Zhuge. The Stanford Data Warehousing Project. *IEEE Quarterly Bulletin on Data Engineering. Special Issue on Materialized Views and Data Warehousing*, 18(2) :41–48, 1995.

[HMV99a] Carlos A. Hurtado, Alberto O. Mendelzon, and Alejandro A. Vaisman. Maintaining Data Cubes under Dimension Updates. In *ICDE*, pages 346–355, 1999.

[HMV99b] Carlos A. Hurtado, Alberto O. Mendelzon, and Alejandro A. Vaisman. Updating OLAP dimensions. In *DOLAP '99 : Proceedings of the 2nd ACM international workshop on Data warehousing and OLAP*, pages 60–66. ACM Press, 1999.

[How03] P. Howard. Database Performance, IBM, Oracle and Microsoft, une évaluation. *Bloor Research*, Novembre 2003.

[HRU95] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing Data Cubes Efficiently. A full version. Technical Report IRI-92-23406 and DAAH04-95-1-0192 and F33615-93-1-1339, 1995.

[HRU96] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing Data Cubes Efficiently. In *SIGMOD. ACM 0-89791-7944/96/0006*, pages 205–216, 1996.

[IH94] W. Inmon and R.D. Hackathorn. *Using the Data Warehouse.* 1994.

[Inf02] Informix Metacube - http ://publib.boulder.ibm.com/epubs/pdf/ct1s9na.pdf, 2002.

[Inm92] William Inmon. *Building the Data Warehouse.* QED Technical Publishing Group, Wellesley, Massachusetts, U.S.A., 1992, 1992.

[Inm95] William Inmon. What is a Data Warehouse, White paper., 1995.

[INS05] Institut National de la Statistique et des Etudes Economiques, France - http ://www.insee.fr/fr/home/home_page.asp, 2005.

[JJQV99] M. Jarke, M. Jeusfeld, C. Quix, and P. Vassiliadis. Architecture and Quality in Data Warehouses : An Extended Repository Approach. *Information Systems*, 24(3) :229–253, 1999.

[JLS99] H. Jagadish, L. Lakshmanan, and D. Srivastava. What can Hierarchies do for Data Warehouses ? In *The VLDB Journal*, pages 530–541, 1999.

[JQJ98] M. Jeusfeld, C. Quix, and M. Jarke. Design and Analysis of Quality Information for Data Warehouses. In *International Conference on Conceptual Modeling / the Entity Relationship Approach*, pages 349–362, 1998.

[JV97] M. Jarke and Y. Vassiliou. Data Warehouse Quality : A Review of the DWQ Project. Technical report, 1997.

[KC88] Won Kim and Hong-Tai Chou. Versions of Schema for Object-Oriented Databases. In *Proceedings of the Fourteenth International Conference on Very Large Data Bases*, pages 148–159. Morgan Kaufmann Publishers Inc., 1988.

[KC02] Heum-Geun Kang and Chin-Wan Chung. Exploiting Versions for Online Data Warehouse Maintenance in MOLAP Servers. In *Proceedings of the 28th VLDB Conference*, 2002.

[Kim96] R. Kimball. *Entrepôts de données, Guide pratique du concepteur de data warehouse*. John Wiley and Sons, Inc., 1996.

[KM99] Sachin Kulkarni and Mukesh K. Mohania. Concurrent Maintenance of Views Using Multiple Versions. In *International Database Engineering and Application Symposium*, pages 254–259, 1999.

[KMP02] Panos Kalnis, Nikos Mamoulis, and Dimitris Papadias. View selection using randomized search. *Data Knowl. Eng.*, 42(1) :89–111, 2002.

[KR99] Yannis Kotidis and Nick Roussopoulos. DynaMat : a dynamic view management system for data warehouses. In *SIGMOD '99 : Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 371–382. ACM Press, 1999.

[KR03] R. Kimball and M. Ross. *Entrepôts de données, Guide pratique de modélisation dimensionnelle*. Vuibert, Paris, 2003.

[KS95] Ralph Kimball and Kevin Strehlo. Why Decision Support Fails and How To Fix It. *SIGMOD Record*, 24(3) :92–97, 1995.

[KU95] Arthur M. Keller and Jeffrey D. Ullman. A Version Numbering Scheme with a Useful Lexicographical Order. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 240–248. IEEE Computer Society, 1995.

[Lau96] Sven-Eric Lautemann. An Introduction to Schema Versioning in OODBMS. In *Database and Expert Systems Applications*, pages 132–139, 1996.

[LMSS95] A. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering Queries Using Views. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 95–104, San Jose, Calif., 1995.

[LW96] Chang Li and Xiaoyang Sean Wang. A Data Model for Supporting On-Line Analytical Processing. In *CIKM*, pages 81–88, 1996.

[LZW+97] W. Labio, Y. Zhuge, J. Wiener, H. Gupta, H. Garcia-Molina, and J. Widom. The WHIPS prototype for data warehouse creation and maintenance. In *In Proceedings ACM SIGMOD International Conference on Management of Data*, pages 557–559, May 1997.

[Mar98] P. Marcel. *Manipulations de Données Multidimensionnelles et Langages de Règles*. PhD thesis, Institut National des Sciences Appliques de Lyon, France, 1998.

[MBJ+02] Renata Matos, Adriana Bueno, Anelise Jantsch, Nina Edelweiss, and Clesio Saraiva. Dynamic Schema Evolution Management Using Version in Temporal Object-Oriented Databases. In *Database and Expert Systems Applications*, pages 524–533, 2002.

[ME97] Viviane Pereira Moreira and Nina Edelweiss. Queries to Temporal Databases Supporting Schema Versioning, 1997.

[Med05] Aci project MEDIGRID : medical data storage and processing on the GRID - http ://www.creatis.insa-lyon.fr/medigrid/, 2005.

[MQM97] Inderpal Singh Mumick, Dallan Quass, and Barinderpal Singh Mumick. Maintenance of data cubes and summary tables in a warehouse. In *SIGMOD '97 : Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 100–111. ACM Press, 1997.

[MS90] E. McKenzie and Richard Thomas Snodgrass. Schema evolution and the relational algebra. *Inf. Syst.*, 15(2) :207–232, 1990.

[MS93] Simon Monk and Ian Sommerville. Schema evolution in OODBs using class versioning. *SIGMOD Rec.*, 22(3) :16–22, 1993.

[Mun93] B. P. Munch. *Versioning in a Software Database - The Change Oriented Way*. PhD thesis, Norwegian Institute of Technologie, 1993.

[Odb92] Erik Odberg. A Framework for Managing Schema Versioning in Object-Oriented databases. In *DEXA*, pages 115–120, 1992.

[Ola04] The OLAP Report - http ://www.olapreport.com/fasmi.htm, 2004.

[ORA96] Oracle Express Server - http ://otn.oracle.com/documentation /express_server.html, 1996.

[PCTM02] J. Poole, D. Chang, D. Tolbert, and D. Mellor. *Common Warehouse Metamodel, An Introduction to the Standard for Data Warehouse Integration*. Wiley Publishing, Inc., 2002.

[PCTM03] J. Poole, D. Chang, D. Tolbert, and D. Mellor. *Common Warehouse Metamodel, Developer's Guide*. Wiley Publishing, Inc., 2003.

[PMS04] Programme de Médicalisation des Systèmes d'Information - http ://www.atih.sante.fr/, 2004.

[Qui99] Christoph Quix. Repository Support for Data Warehouse Evolution. In *DMDW*, page 4, 1999.

[QW97] Dallan Quass and Jennifer Widom. On-line warehouse view maintenance. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 393–404. ACM Press, 1997.

[RGMS01] John F. Roddick, Fabio Grandi, Federica Mandreoli, and Maria Rita Scalas. Beyond Schema Versioning : A Flexible Model for Spatio-Temporal Schema Selection. *Geoinformatica*, 5(1) :33–50, 2001.

[RKZ+99] E. A. Rundensteiner, A. Koeller, X. Zhang, A. J. Lee, A. Nica, A. Van Wyk, and Y. Lee. Evolvable view environment (EVE) : non-equivalent view maintenance under schema changes. In *SIGMOD '99 : Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 553–555. ACM Press, 1999.

[RKZ00] Elke A. Rundensteiner, Andreas Koeller, and Xin Zhang. Maintaining data warehouses over changing information sources. *Commun. ACM*, 43(6) :57–62, 2000.

[Rod92] John F. Roddick. Schema evolution in database systems : an annotated bibliography. *SIGMOD Rec.*, 21(4) :35–40, 1992.

[Rod95] John F. Roddick. A survey of schema versioning issues for database systems. *Information and Software Technology*, 37(7) :383–393, 1995.

[RR97] Young-Gook Ra and Elke A. Rundensteiner. A Transparent Schema-Evolution System Based on Object-Oriented View Technology. *IEEE Transactions on Knowledge and Data Engineering*, 9(4) :600–624, 1997.

[SA04] Maria-Trinidad Serna and Michel Adiba. Los Almacenes de Datos como soporte a la decisión médica. In *Proceedings of the Congreso Internacional en Ciencias Computacionales(CICC04)*, Colima, México, Mars 2004.

[SA05a] Maria-Trinidad Serna and Michel Adiba. Conception et optimisation d'un entrepôt de données médicales. *Revue des Nouvelles Technologies de l'Information. RNTI-B-1*, pages 122–140, 2005.

[SA05b] Maria-Trinidad Serna and Michel Adiba. Exploiting bitemporal schema versions for managing an historical medical data warehouse : A case study. In *Proceedings of the Encuentro Nacional de computación (ENC05)*, Puebla, México, September 2005.

[SAS04] SAS OLAP Server - http ://www.sas.com/technologies/dw/storage/mddb/, 2004.

[Sat03] Ulrike Sattler. Data Warehouse Models and OLAP Operations - http ://www.cs.man.ac.uk/ sattler/teaching/cs636.html, 2003.

[Sch01] Michel School. *Bases de données géographiques*, volume Chapitre 6. Editions Hermès, 2001.

[SDJL96] Divesh Srivastava, Shaul Dar, H. V. Jagadish, and Alon Y. Levy. Answering Queries with Aggregation Using Views. In *VLDB '96 : Proceedings of the 22th International Conference on Very Large Data Bases*, pages 318–329. Morgan Kaufmann Publishers Inc., 1996.

[SMKK98] Sunil Samtani, Mukesh K. Mohania, Vijay Kumar, and Yahiko Kambayashi. Recent Advances and Research Problems in Data Warehousing. In *ER Workshops*, pages 81–92, 1998.

[SMS04] Site du Ministère des Solidarités de la Santé et de la Famille - http ://www.sante.gouv.fr/, 2004.

[Sno95] Richard T. Snodgrass. *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers, 1995.

[Sno99] Richard T. Snodgrass. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann, 1999.

[Tes00] O. Teste. *Modélisation et manipulation d'entrepôts de données complexes et historisées*. PhD thesis, Université Paul Sabatier de Toulouse, Décembre 2000.

[TS93] Markus Tresch and Marc H. Scholl. Schema transformation without database reorganization. *SIGMOD Rec.*, 22(1) :21–27, 1993.

[TS99] D. Theodoratos and T. Sellis. Dynamic Data Warehouse Design. In *Data Warehousing and Knowledge Discovery*, pages 1–10, 1999.

[TU98] Michael Teschke and Achim Ulbrich. Concurrent Warehouse Maintenance Without Compromising Session Consistency. In *Database and Expert Systems Applications*, pages 776–785, 1998.

[Var00] G. Vargas. *Service d'évenements flexible pour l'intégration d'applications bases de données réparties*. PhD thesis, Université Joseph Fourier, Grenoble, France, Décembre 2000.

[Vas98] Panos Vassiliadis. Modeling Multidimensional Databases, Cubes and Cube Operations. In *SSDBM '98 : Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, pages 53–62. IEEE Computer Society, 1998.

[VGD00] A. Vavouras, S. Gatziu, and K. Dittrich. Modeling and Executing the Data Warehouse Refreshment Process. Technical Report ifi-2000.01, 11, 2000.

[VQVJ00] Panos Vassiliadis, Christoph Quix, Yannis Vassiliou, and Matthias Jarke. A Model for Data Warehouse Operational Processes. In *Conference on Advanced Information Systems Engineering*, pages 446–461, 2000.

[VS99] Panos Vassiliadis and Timos K. Sellis. A Survey of Logical Models for OLAP Databases. *SIGMOD Record*, 28(4) :64–69, 1999.

[WB97] M-C Wu and A.P. Buchmann. Research Issues in Data Warehousing. In *BTW German Database Conference*, 1997.

[Wid95] Jennifer Widom. Research problems in data warehousing. In *CIKM '95 : Proceedings of the fourth international conference on Information and knowledge management*, pages 25–30, New York, NY, USA, 1995. ACM Press.

[WMC01] Bernhard Westfechtel, Bjorn P. Munch, and Reidar Conradi. A Layered Architecture for Uniform Version Management. *Software Engineering*, 27(12) :1111–1133, 2001.

[YKL97] J. Yang, K. Karlapalem, and Q. Li. Algorithms for Materialized View Design in Data Warehousing Environment. In *The VLDB Journal*, pages 136–145, 1997.

[YW00] J. Yang and J. Widom. Making temporal views self-maintainable for data warehousing. In *ICDE, In Proceedings of the 7th International Conference on Extending Database Technology*, March 2000.

[ZR98] Xin Zhang and Elke A. Rundensteiner. Data Warehouse Maintenance Under Concurrent Schema and Data Updates. Technical report, 1998.

[ZS99] Thomas Zurek and Markus Sinnwell. Datawarehousing Has More Colours Than Just Black & White. In *VLDB '99 : Proceedings of the 25th International Conference on Very Large Data Bases*, pages 726–729. Morgan Kaufmann Publishers Inc., 1999.

[ZYY01] C. Zhang, X. Yao, and J. Yang. An evolutionary approach to materialized view selection in a data warehouse environment. *IEEE Trans. Systems, Man, Cybernetics, PART C*, 31 :282–294, September 2001.

Annexe A

Description des sources de données

Caractéristiques du fichier RSA :

Tout séjour hospitalier, effectué dans la partie court séjour d'un établissement, fait l'objet d'un Résumé de Sortie Standardisé (R.S.S.) constitué d'un ou plusieurs Résumé(s) d'Unité Médicale (R.U.M.). La transmission d'informations médicales à la direction de l'établissement ou à l'extérieur de celui-ci, s'opère sur la base de données agrégées ou de Résumés de Sortie Anonymes (R.S.A.) obtenus par transformation des RSS. La production des RSA est automatisée. A partir du fichier de RSS groupés, le médecin responsable du DIM utilise le logiciel GENRSA (Générateur de RSA), propriété de l'Etat, pour produire le fichier de RSA. A la différence du RSS, le RSA constitue toujours un enregistrement unique par séjour.

Description des variables du fichier :

GHM : Tout Résumé de Sortie Standardisé est classé dans un Groupe Homogène de Malades. La classification en GHM permet un classement exhaustif et unique : tout séjour est obligatoirement classé dans un GHM et dans un seul. A partir des variables médico-administratives contenues dans le RSS, chaque séjour va aboutir dans l'un des groupes de la classification. Celle-ci comporte 512 groupes, soit 462 GHM et 50 groupes autres, dont 46 groupes "ambulatoires" et 4 groupes dans la Catégorie Majeure n90 (erreurs et séjours inclassables).

CMD : Les séjours de moins de 24 heures (séances, décès immédiat, transfert immédiat, pathologies traitées en moins de 24 heures) sont classés dans la Catégorie Majeure n24 (CM24 : séances et séjours de moins de 24 heures), spécificité française absente de la classification américaine qui ne prend pas en compte les séjours de moins de 48 heures. Les séjours de plus de 24 heures sont classés dans l'une des 23 Catégories Majeures de Diagnostic (CMD01 à CMD23), en fonction du diagnostic principal contenu dans le RSS.

RUM : Le RUM contient un nombre limité d'informations qui doivent être systématiquement renseignées. Ces informations sont d'ordre administratif et médical. Pour que les informations médico-administratives contenues dans le RUM puissent

bénéficier d'un traitement automatisé, elles sont codées selon des nomenclatures et des classifications standardisées. Les nomenclatures utilisées pour coder les RUM sont :

Pour le codage des diagnostics, la Classification Internationale des Maladies (C.I. M.)
Pour le codage des actes, le Catalogue des Actes Médicaux (C.d.A.M.)

DP : (concerne les séjours de plus de 24 heures) :

Dans le cas d'un séjour mono-unité, le diagnostic principal contenu dans le RUM devient celui du RSS.
Dans le cas d'un séjour multi-unité, le diagnostic principal est obtenu par l'algorithme suivant : si un seul RUM comporte un acte classant opératoire, le DP de ce RUM est celui du RSS ; si aucun RUM ne mentionne d'acte classant opératoire ou, si plusieurs RUM en mentionnent (au moins) un, le DP du RSS est celui du RUM dont la durée de séjour est la plus longue ; si deux RUM possèdent la même durée de séjour, le DP du RSS est celui du RUM de la dernière unité chronologiquement fréquentée.

Codage du mode d'entrée Le mode d'entrée dans l'unité médicale se code comme suit :

6 : mutation La mutation signifie la provenance d'une autre unité médicale de la même entité juridique.
7 : transfert normal Le transfert normal signifie la provenance d'une autre entité juridique pour une hospitalisation à part entière, à distinguer du cas de la prestation réalisée pour le compte de l'établissement d'origine, codé 0.
8 : domicile Le patient vient de chez lui.
0 : transfert pour ou après réalisation d'un acte Ce type particulier de transfert est réservé à deux cas :

Un établissement "prestataire" reçoit un patient pour une durée de moins de 48 heures dans le seul but de réaliser un acte que l'établissement d'origine ne peut pas réaliser lui-même. Cette "prestation" donnera d'ailleurs lieu à une facturation à l'établissement d'origine.

Un établissement "donneur d'ordre" récupère un patient qu'il avait transféré pour la réalisation d'un acte qu'il ne pouvait réaliser lui-même. Il s'agit donc de la fin de la prestation.

Codage du mode de sortie Le mode d'entrée dans l'unité médicale se code comme suit :

6 : mutation La mutation signifie le départ vers une autre unité médicale de la même entité juridique.

7 : transfert normal Le transfert normal signifie le départ vers une autre entité juridique, à distinguer du cas d'un retour à l'établissement d'origine après la réalisation d'une prestation, codé 0.

8 : domicile Le patient rentre chez lui.

9 : décès Le patient est décédé dans l'unité médicale.

0 : transfert pour ou après réalisation d'un acte Ce type particulier de transfert est réservé à deux cas :

Un établissement "prestataire" retourne le patient dans son établissement d'origine après l'avoir accueilli pour une durée de moins de 48 heures dans le seul but de réaliser un acte qui ne pouvait pas être réalisé sur place. Il facture d'ailleurs cette "prestation" à l'établissement d'origine.

Un établissement "donneur d'ordre" envoie un patient dans un autre établissement pour la réalisation d'un acte qu'il ne peut réaliser lui-même.

Liste Alphabétique des Variables :

Les variables sont classées dans l'ordre alphabétique de leur code et sont décrites dans le tableau de la manière suivante :

- le code
- le type (numérique ou caractère)
- la longueur
- le libellé

Les tableaux A.1 et A.2 décrivent les variables du fichier RSA.

Notation :

* = Pour le privé
+ = Pour le public
La variable Filler pour les RSA publics regroupe les champs 34, 35 et 36 des RSA des établissements privés.

Caractéristiques du fichier FINESS :

Ce fichier recense les structures sanitaires et sociales au niveau national, chaque structure étant renseignée par son numéro FINESS. Il contient trois types de structures publiques ou privées :

Variable	Type	Taille	Libellé
FINESS	Caractère	9	Numéro FINESS de l'entité juridique
Champ 2	Caractère	3	Nř de version du format du RSA (C05)
Champ 3 + numRSA	Caractère	10	Clef de validation
Champ 5	Caractère	3	Numéro de version du format du "RSS-groupé" (ou du RSS) lu
Champ 6	Caractère	3	Nř de version de Genrsa
Champ 7 + cmd_ étab + ghm_étab + Champ 10	Caractère	9	Groupage lu sur le "RSS-groupé" : V, CMD, GHM, code retour
Champ 11 + cmd + ghm + Champ14	Caractère	9	Groupage obtenu par GENRSA : V, CMD, GHM, code retour
Champ 15	Numérique	2	Nombre de RUM composant le RSS d'origine
AGE_ANNEES	Numérique	3	Age en années
AGE_JOURS	Caractère	3	Age en jours (si < 1 an) calculé à l'entrée
sexe	Caractère	1	Sexe
Mode_ent	Caractère	1	Mode d'entrée dans le champ du PMSI
Provenan	Caractère	1	Provenance (si le mode d'entrée est mutation ou transfert)
Mois_sor	Caractère	2	Mois de sortie du champ PMSI
Année_sor	Caractère	4	Année de sortie du champ PMSI
Mode_sor	Caractère	1	Mode de sortie du champ PMSI
Destinat	Caractère	1	Destination (si le mode de sortie est mutation ou transfert)
Type_séjour	Caractère	1	Type de séjour
Nbre_rad	Numérique	2	Nombre d'actes de radiothérapie
Nbre_dia	Numérique	2	Nombre d'actes de dialyse
Durée_se	Numérique	3	Durée totale du séjour dans le champ du PMSI (vide si séances)

TAB. A.1 – Description des variables du fichier RSA

Variable	Type	Taille	Libellé
Champ 29	Caractère	3	Durée de la période sur laquelle s'étalent les séances (si séances)
Code_geo	Caractère	5	Code géographique de résidence
Poids	Numérique	4	Poids à la naissance (en grammes)
Nbre_sea	Numérique	2	Nombre de séances
Igs2	Caractère	3	IGS 2
Filler	Caractère	8	Filler +
Champ 34	Caractère	1	Code de prise en charge *
Champ 35	Caractère	1	Facture associée à 0 franc *
Champ 36	Caractère	6	Zone réservée *
Diag_pri	Caractère	6	Diagnostic principal (DP)
Diag_rel	Caractère	6	Diagnostic relié (DR)
Nbre_di1	Numérique	2	Nombre de diagnostics associés (Nd) dans ce RSA
Nbre_act	Numérique	2	Nombre d'actes (Na) dans ce RSA

TAB. A.2 – Description de variables du fichier RSA

Les structures sanitaires : structures hospitalières, autres centres de soins, laboratoires et pharmacies.

Les structures sociales : personnes âgées, jeunesse handicapée, adultes handicapés, aide sociale à l'enfance, adultes en difficulté sociale.

Les structures de formation des personnels sanitaires et sociaux.

Mise à jour :

Les mises à jour de ce fichier sont faites annuellement par la DRASS (Direction Régionale des Affaires Sanitaires et Sociales) et la DDASS (Direction Départementale des Affaires Sanitaires et Sociales).

La source de données :

Les données proviennent du site Internet du ministère de la santé : http ://www.sante.gouv.fr. Le fichier national des établissements sanitaires et sociaux (FINESS) est géré par la Direction de la Recherche, des Etudes de l'Evaluation et des Statistiques (DREES) - Unité Répertoires - Ministère de l'Emploi et de la Solidarité, en liaison avec les services déconcentrés de l'Etat (DDASS, DRASS).

Description des variables du fichier :

Fichier FINESS : Fichier National des Etablissements Sanitaires et Sociaux.

Une entité juridique : Correspond à la notion de personne morale. Une personne morale exerce son activité dans des établissements. Elle représente juridiquement l'ensemble de ses établissements.

Un établissement : Correspond à une implantation géographique. De plus, quand dans un même lieu, plusieurs activités dépendent de budgets distincts, on identifie autant d'établissements dans le même lieu que de budgets distincts. Exemple : un Centre hospitalier régional C.H.R (entité juridique) peut avoir plusieurs implantations géographiques et dans certaines de ses implantations avoir un service de soins de longue durée qui dépend d'un budget annexe. Il y aura dans FINESS autant d'établissements que d'implantations géographiques et de services dépendant d'un budget annexe. Une association (entité juridique) gérant sur un même lieu un C.A.T. et un foyer d'hébergement aura, dans FINESS, 2 établissements à la même adresse puisque ces 2 structures ont réglementairement des budgets séparés.

Le numéro FINESS : A chaque établissement et à chaque entité juridique est attribué un numéro FINESS à 9 chiffres dont les 2 premiers correspondent au numéro de département d'implantation. Ce numéro est un numéro de référence en particulier pour la facturation hospitalière et la liquidation de sécurité sociale. Rien ne distingue, dans la constitution du numéro, un numéro d'entité juridique d'un numéro d'établissement.

Le type d'établissement : Les établissements sont caractérisés dans FINESS par leur catégorie et les disciplines qu'ils sont autorisés à exercer. Ces informations sont la traduction d'une réglementation complexe et de la possibilité de pluridisciplinarité des établissements sanitaires. Exemple : un établissement d'un C.H.R (lieu géographique) peut à la fois comprendre des services de médecine, de chirurgie, une maternité et une école d'infirmière. Pour faciliter votre recherche, nous avons retenu une présentation par type d'établissement qui résulte du croisement des informations de catégorie et de disciplines. Un établissement pourra ainsi appartenir à plusieurs types en fonction de son niveau de pluridisciplinarité.

La catégorie : Elle caractérise le cadre réglementaire dans lequel s'exerce l'activité de l'établissement. Exemples : centre Hospitalier, Institut médico-éducatif, centre d'aide par le travail.

La discipline : Désigne une activité homogène qui est fonction du type de soins ou de service social. Exemple : médecine générale, pédiatrie, chirurgie, hébergement, accueil temporaire, éducation. Dans le domaine sanitaire, certaines autorisations sont faites par Grands Groupes de Disciplines GGDE qui sont des regroupements de disciplines. Dans le cadre de ce site Internet, les informations relatives à ces disciplines présentes dans le fichier FINESS ne sont pas accessibles dans leur détail. Seules apparaissent sur la fiche des établissements concernés les GGDE des établissements sanitaires et les disciplines d'enseignement des établissements d'enseignements.

Le statut : Caractérise la situation juridique de la personne morale dont dépend l'établissement. Exemple : établissement d'hospitalisation communal, association loi 1901 reconnue d'utilité publique, SARL.

La capacité : Pour les établissements sanitaires, est affichée la capacité en lits par grand groupe de discipline autorisée (par un arrêté) et mise en IJuvre (dont l'installation a été constatée et certifiée conforme). Pour les établissements sociaux, est affichée la capacité totale observée en places par sexe. Pour les établissements d'enseignement, est affiché le nombre de places d'une promotion par discipline d'enseignement.

Les adresses : Pour les établissements sanitaires et sociaux, est affichée, quand elle existe, une deuxième adresse : l'adresse d'accueil d'urgence 24H sur 24 pour le public.

Le tarif : Détermine l'autorité responsable de la fixation du tarif principal de l'établissement et la procédure utilisée.

La participation au service public hospitalier : Pour les établissements privés relevant de la loi hospitalière, il indique si l'établissement participe au service public hospitalier et sous quelle forme.

Les numéros SIREN et SIRET : Sont l'identifiant de l'entité juridique (SIREN) et celui de l'établissement (SIRET) attribué par l'INSEE dans le cadre du système national légal d'identification des entreprises et de leurs établissements.

Liste Alphabétique des Variables :

Variable	Type	Taille	Libellé
Adresse	Char	35	Adresse de l'établissement
Adressepublic	Char	25	Adresse de l'établissement
Capaciteautorisee1	Num	8	capacité autorisée en nombre de lits
Capaciteautorisee2	Num	8	capacité autorisée en nombre de lits
Capaciteautorisee3	Num	8	capacité autorisée en nombre de lits
Capaciteautorisee4	Num	8	capacité autorisée en nombre de lits
Capaciteautorisee5	Num	8	capacité autorisée en nombre de lits
Capacitemiseenoeuvre1	Num	8	capacité observée en nombre de lits
Capacitemiseenoeuvre2	Num	8	capacité observée en nombre de lits

Tab. A.3 – Description de variables du fichier FINESS

Variable	Type	Taille	Libellé
Capacitemiseenoeuvre3	Num	8	capacité observée en nombre de lits
Capacitemiseenoeuvre4	Num	8	capacité observée en nombre de lits
Capacitemiseenoeuvre5	Num	8	capacité observée en nombre de lits
Code1	Num	8	Code de la discipline
Code2	Num	8	Code de la discipline
Code3	Num	8	Code de la discipline
Code4	Num	8	Code de la discipline
Code5	Num	8	Code de la discipline
CodePSPH	Num	8	Code de Part. au Serv. Pub. Hosp.
Codecategorie	Num	8	Code l'activité de l'étab.
Codepostal	Num	8	Code postal
Codepostalpublic	Char	5	Code postal public
Codestatut	Num	8	Code le statut de l'étab.
Codetarif	Num	8	Code de l'autorité fixant le tarif
Compldistrpublic	Char	20	Complément de distribution public
Complementdistribution	Char	35	Complément de distribution
Dateouvert	Char	20	Date d'ouverture
FINESSjuridique	Char	9	Numéro FINESS juridique
Fax	Char	15	Numéro Fax
Faxpublic	Char	15	Numéro Fax public
GGDE1	Char	20	Grands Groupe de Discipline 1
GGDE2	Char	20	Grands Groupe de Discipline 2
GGDE3	Char	20	Grands Groupe de Discipline 3
GGDE4	Char	20	Grands Groupe de Discipline 4
GGDE5	Char	20	Grands Groupe de Discipline 5
LibPSPH	Char	20	Libellé de Part. au Serv. Pub. Hosp.
Libcategorie	Char	25	Libellé de la catégorie
Libelleroutage	Char	30	Libelle du routage
Libstatut	Char	25	Libellé du statut
Libtarif	Char	20	Libellé de l'autorité fixant le tarif
LieuditBP	Char	35	Lieudit BP
LieuditBPpublic	Char	10	Lieudit BP public
NumeroFINESS	Char	9	Numéro FINESS
Raisonsociale	Char	65	Nom de l'établissement
Routagepublic	Char	20	Routage public
SIRET	Char	20	Numéro SIRET
Tel	Char	15	Numéro Téléphone
Telpublic	Char	15	Numéro Téléphone public

Tab. A.4 – Description de variables du fichier FINESS