Infrastructure adaptable pour les entrepots de données

Edgard-Iván Benitez Guerrero

Université Joseph Fourier - Grenoble I

Thèse

pour obtenir le grade de

Docteur de l'Université Joseph Fourier

(arrêtés ministériels du 5 juillet 1984 et du 30 mars 1992)

Discipline : Informatique

présentée et soutenue publiquement par

Edgard Iván Benítez Guerrero

Le 26 septembre 2002

Infrastructure adaptable pour l'évolution des entrepôts de données

Composition du jury

Rapporteurs :	Mme. Zohra Bellahsène
	Mme. Anne Doucet
Examinateurs :	Mme. Marie-France Bruandet
	M. Michel Simonet
Directeurs de thèse :	Mme. Christine Collet
	M. Michel Adiba

Thèse préparée au sein du laboratoire Logiciels, Systèmes, Réseaux - IMAG

À ma mère Rosa Elena,
qui veille toujours sur moi.

Je tiens à remercier :

Zohra Bellahsène, Maître de Conférences à l'Université Montpellier II, et Anne Doucet, Professeur à l'Université Pierre et Marie Curie, les rapporteurs de ma thèse, pour le temps qu'elles ont consacrées à la lecture et à la correction de ce document. À Marie-France Bruandet, Professeur à l'Université Joseph Fourier (Grenoble I) et Michel Simonet, Chargé de Recherche au CNRS, pour avoir accepté de faire partie du jury de cette thèse.

Christine Collet et Michel Adiba, mes directeurs de thèse, pour m'avoir accueilli dans leur équipe. Je les remercie tout particulièrement de leur soutien pendant ces années. Je les remercie également pour leurs conseils, leurs encouragements et leur patience.

Aux membres et collaborateurs de l'équipe STORM-BD. Merci à Elizabeth Pérez-Cortés, Claudia Roncancio, Pierre Habraken et Cyril Labbe, pour la disponibilité qu'ils ont toujours eu pour m'aider pendant le réalisation de ce travail. À mes collègues de bureau Khalid Belhajjame et Trinh Vu pour les discussions qui ont contribuées à la préparation de ma thèse. Un grand merci à tous les autres doctorants de l'equipe : Patricia Serrano-Alvarado, Trinidad Serna, Phuong-Quynh Duong Stéphane Drapeau, Olivier Guyotot, Tanguy Nedelec, Luciano Garcia-Banuelos, Gennaro Bruno, pour leur conseils qui ont beaucoup servi à la préparation de ma soutenance. Merci à tous pour votre amitié.

Stephanie Lonchampt et Adnane Benaissa qui ont beaucoup contribué à la validation expérimentale des idées proposées dans ce travail.

Genoveva Vargas-Solar et Jose-Luis Zechinelli-Martini. Je les remercie du temps qu'ils ont consacrés à de longues discussions sur mon travail, et plus encore, pour leur encouragements et leur amitié.

En général à tous les membres du Laboratoire Logiciels, Systèmes, Réseaux de l'IMAG, en particulier à Paul Jacquet, Professeur à l'Institut National Polytechnique de Grenoble, directeur du laboratoire.

Merci aux membres des équipes du LSR : ADELE, DRAKKAR, PLIAGE , SARDES, SIGMA, VASCO.

Merci aux équipes administratives (Liliane Di Giacomo, Martine Pernice, Solange Roche, Sonia Nogueira, Pascale Poulet) et techniques (Fraçois Challier et Christiane Plumère) pour leur aide et disponibilité.

Aux amis que j'ai eu la chance de rencontrer au LSR : Helena et Alexandre Ribeiro, Marlon Dumas, Rafael Lozano, Olivier Lobry, Antonio Garcia, Raul Garcia, Beatriz Gonzalez, Leopoldo Moran, Sonia Mendoza ...

Consejo Nacional de Ciencia y Tecnologia (CONACyT) du gouvernement du Mexique pour son soutien économique. Je serai honoré de pouvoir contribuer au développement de mon pays.

Merci à Cristina Loyo et aux autres membres de LANIA, en particulier à Christian Lemaitre, Ofelia Cervantes et Antonio Sanchez, pour leurs encouragements et pour avoir contribué à ce que mes études en France soient possibles.

Merci à tous les membres de la "Maestria en Inteligencia Artificial" de Xalapa au Mexique, en particulier à Angelica Garcia, pour leurs encouragements.

Mon père, Dionisio Benitez, merci de ton soutien inconditionnel et pour l'amour que seulement un père peut donner à ses enfants.

Je remercie mes sœurs Edith, Leticia et mon frère Sergio, ainsi que leurs familles respectives, pour toujours avoir confiance en moi. J'espère pouvoir vous rendre tout ce que vous avez fait pour moi. Je pense également à ma tante Victoria et ma cousine Blanca et sa famille, qui m'ont toujours soutenu. Un grand merci à toute ma famille à Xalapa.

À la famille Mezura-Godoy. Merci à mes beaux parents, Miguel et Conception, et à mon beau frère, Carlos Miguel, pour leurs encouragements. J'espère qu'ils seront fiers de leur nouvel élève. Merci aussi aux membres des familles Mezura et Godoy à Xalapa, Guasave, Tijuana ....

Carmen, ma *più bella cosa*. Il n'existe pas de mots pour te remercier tout ce que tu as fait pour moi. Merci de ta compréhension pendant les moments les plus durs lors de la rédaction de cette thèse. Merci pour être toujours là.

Table des matières

1 Introduction **1**

1.1 Les entrepôts de données pour l'aide à la décision 1

1.2 Problématique et objectifs 4

1.2.1 L'évolution du schéma des entrepôts de données 5

1.2.2 Vers une infrastructure adaptable pour l'évolution des entrepôts 7

1.2.3 Objectif de la thèse 8

1.3 Démarche et contribution 8

1.3.1 Modèle multidimensionnel de données 8

1.3.2 Langage de description de données multidimensionnelles . . . 9

1.3.3 Infrastructure adaptable pour l'évolution des entrepôts 9

1.3.4 Validation expérimentale 10

1.4 Organisation du document 10

2 Les entrepôts de données **13**

2.1 Concepts généraux 13

2.1.1 Entrepôts et magasins de données 13

2.1.2 Structure d'un entrepôt de données 15

2.2 Organisation logique 16

2.2.1 Schémas relationnels 17

2.2.2 Schéma multidimensionnel 18

2.3 Le niveau physique des entrepôts 20

2.3.1 Stockage de données dans un système multidimensionnel . . . 20

2.3.2 Méthodes d'accès indexées 22

2.3.3 Partitionnement 24

2.4 Construction et rafraîchissement 24

2.4.1 Adaptateur/moniteur 25

2.4.2 Intégrateur 26
2.4.3 Le chargement de données dans l'entrepôt 28
2.5 Interrogation de l'entrepôt 29
2.5.1 Expression et traitement des requêtes 29
2.5.2 Le précalcul d'agrégats 31
2.5.3 Traitement parallèle de requêtes 32
2.6 Analyse de l'entrepôt 33
2.6.1 Traitement analytique en ligne 33
2.6.2 Fouille de données 35
2.6.3 Visualisation 36
2.7 Évolution de schéma 37
2.7.1 Les précurseurs 37
2.7.2 L'évolution de schéma dans le contexte des entrepôts 40
2.8 Conclusion 44

3 Modèle et langage d'évolution 47
3.1 Le modèle de données MD 48
3.2 Opérateurs d'évolution 52
3.2.1 Évolution de schémas de dimension 52
3.2.2 Évolution de schémas de cube 57
3.3 Le langage MDL 61
3.3.1 Notions de base 62
3.3.2 Création et suppression de dimensions 64
3.3.3 Modification de dimensions 65
3.3.4 Création et suppression de cubes 67
3.3.5 Modification de cubes 68
3.4 Conclusion 70

4 Infrastructure pour la spécification de gestionnaires d'entrepôt 73
4.1 Gestionnaire d'entrepôt 73
4.1.1 Description générale 73
4.1.2 Fonctionnement 76
4.2 Données d'un gestionnaire 77
4.3 Création de l'entrepôt 78
4.3.1 Création du schéma multidimensionnel 79

4.3.2 Création du schéma d'implantation 80
4.3.3 Construction de l'entrepôt 82
4.4 Évolution de l'entrepôt 90
4.4.1 Modification du schéma multidimensionnel 90
4.4.2 Évolution du schéma d'implantation 95
4.4.3 Adaptation des données de l'entrepôt 99
4.5 Conclusion 101

5 Implantation d'un gestionnaire d'entrepôt 103
5.1 Infrastructure technique 104
5.1.1 Le langage XML 104
5.1.2 Le langage d'interrogation XML-QL 106
5.1.3 L'interface de programmation d'applications DOM 107
5.2 Architecture du gestionnaire d'entrepôt implanté 109
5.3 L'adaptateur XML pour sources relationnelles 111
5.3.1 Architecture interne de l'adaptateur XML implanté 111
5.3.2 Le traducteur de requêtes 111
5.3.3 Le composant d'accès à la source 113
5.3.4 Le traducteur de données 114
5.4 Médiateur XML 114
5.4.1 Architecture interne du médiateur XML implanté 115
5.4.2 Le parseur XML-QL 115
5.4.3 Le décomposeur de requêtes 116
5.4.4 Le distributeur de requêtes 117
5.4.5 Le constructeur de résultats 118
5.5 L'outil d'administration 119
5.5.1 L'interpréteur MDL 120
5.5.2 L'interpréteur WCL 121
5.6 L'intégrateur 122
5.7 Adaptateur d'entrepôt relationnel 124
5.7.1 Le traducteur de schémas multidimensionnels 124
5.7.2 Le traducteur de données multidimensionnelles 126
5.8 Le référentiel 127
5.9 Expérimentation 129
5.9.1 Création du schéma de l'entrepôt 130

5.9.2 Construction de l'entrepôt 132
5.9.3 Évolution du schéma de l'entrepôt 134
5.10 Conclusion 134

6 Conclusion **137**
6.1 Bilan du travail effectué 137
6.1.1 Modèle multidimensionnel de données 137
6.1.2 Le langage MDL 138
6.1.3 Infrastructure de spécification des gestionnaires d'entrepôt . . 138
6.1.4 Validation experimentale 138
6.1.5 Principales contributions 139
6.2 Perspectives 139
6.2.1 Langage pour la création, l'évolution et la manipulation d'entrepôts 139
6.2.2 Adaptation de l'entrepôt suite aux évolutions 139
6.2.3 Gestionnaires d'entrepôt et les services NODS 140
6.2.4 D'autres perspectives de recherche 141

Bibliographie **142**

A Quelques formalismes de cube proposés **163**

B Grammaires **171**

Table des figures

1.1 Architecture générale d'un système "entrepôt de données" 4
1.2 Effets de l'évolution du schéma d'un entrepôt 6

2.1 Le cube de données *Ventes* 14
2.2 La structure d'un entrepôt de données 16
2.3 Schéma en étoile 17
2.4 Schéma en flocon de neige 18
2.5 Le schéma et une instance possible du cube *Ventes* 19
2.6 Schéma et instance de la dimension *Magasin* 19
2.7 Vecteur multidimensionnel pour le stockage du cube *Ventes* 21
2.8 Index binaire 22
2.9 Indexation de vecteurs multidimensionnels 23
2.10 Architecture des systèmes pour l'intégration de données 25
2.11 Application de `slice-n-dice` sur le cube *Ventes* 33
2.12 Application de `roll-up` et `drill-down` sur le cube *Ventes* 34
2.13 Application d'une technique de fouille de données 36
2.14 Schéma ME/R 41

3.1 Le schéma et une instance possible de la dimension *Produit* 50
3.2 Le schéma et une instance possible du cube *Ventes* 51
3.3 La dimension *Produit* après avoir ajouté le niveau *marque* 55
3.4 La dimension Produit après avoir supprimé le niveau *catégorie* . . . 56
3.5 Le cube *Ventes* après avoir ajouté l'axe *ville* 59
3.6 Le cube *Ventes* 60

4.1 Interfaces d'interaction d'un gestionnaire d'entrepôt 74
4.2 Les différents niveaux de représentation d'un entrepôt de données . . 75
4.3 Architecture générale des gestionnaires d'entrepôt 76

4.4 Diagramme UML de classes 78
4.5 Schéma multidimensionnel de l'entrepôt d'une chaîne de magasins . . 79
4.6 Expressions MDL pour créer le schéma multidimensionnel de la figure 4.5 80
4.7 Traduction du schéma multidimensionnel en un schéma relationnel en étoile 81
4.8 Schéma global 83
4.9 Schémas OEM des sources 84
4.10 Le schéma global et les requêtes associées 85
4.11 Stockage relationnel de la dimension *Produit* 89
4.12 Schéma de la dimension *Produit* après (a) ajout et (b) suppression d'un niveau 92
4.13 Le schéma du cube *Ventes* (a) avant et (b) après l'ajout de l'axe *ville* 94

5.1 Représentation XML d'un catalogue de pays 105
5.2 Exemple de requête XML-QL 107
5.3 Exemple d'arbre DOM 108
5.4 Architecture du gestionnaire d'entrepôt implanté 109
5.5 (a) Schéma de la relation `pjaunes`. (b) Schéma XML exporté 111
5.6 Architecture de l'adaptateur XML 112
5.7 Exemple de requête SQL générée 113
5.8 Schéma XML global 115
5.9 Architecture du médiateur de données XML 116
5.10 Sous-requêtes générées à partir de la requête de la figure 5.2 117
5.11 Le distributeur de requêtes 118
5.12 Processus de fusion réalisé par le constructeur de résultats 119
5.13 Requête globale et requête de fusion 120
5.14 Les correspondances entre le schéma de la dimension *Magasin* et le schéma XML global de la figure 5.8. 122
5.15 La correspondance entre le schéma du cube *Ventes* et le schéma XML global. 123
5.16 Représentation XML de la dimension *Magasin* 123
5.17 L'arbre DOM du document XML stockant la dimension *Magasin*. . . 127
5.18 Représentation XML d'un schéma de dimension 128
5.19 Représentation XML d'un schéma de cube 128
5.20 Représentation XML d'une correspondance 129

5.21 Schéma multidimensionnel de l'application exemple 130
5.22 Les expressions MDL utilisées 131
5.23 Représentation XML du schéma de la dimension *Produit* 131
5.24 Les correspondances entre le schéma de la *Produit* et le schéma XML global 132
5.25 Représentation XML de l'instance de la dimension *Produit* 133
5.26 La relation *Produit* stockée dans l'entrepôt 134

A.1 L'HyperCube de Agrawal et al. 165
A.2 Le Cube Multidimensionnel de Li et Wang. (a) Cube Multidimensionnel. (b) Relation de regroupement 166
A.3 Table N-Dimensionnelle de Gyssens et Lakshmanan. (a) Table bi-dimensionnelle (b) Instance de la table bi-dimensionnelle 167
A.4 Le Cube de Thomas et al. 168
A.5 Les f-tables de Cabbibo et Torlone. (a) Schéma (b) Instance 168

Liste des tableaux

1.1 Principales différences entre les bases opérationnelles et les entrepôts 3

3.1 Les types de données offerts par MDL 63

4.1 Taxonomie d'opérations d'évolution 91

A.1 Formalismes de cube proposés 164

Chapitre 1

Introduction

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

Un entrepôt de données est une collection de données destinée à être exploitée par les applications d'aide à la décision [Inm92, JLVV00]. En utilisant des applications d'analyse appropriées, un décideur peut découvrir dans un entrepôt les tendances générales significatives d'une activité et prendre des décisions. Par exemple, la direction d'une chaîne de magasins peut interroger l'entrepôt de données de l'entreprise pour connaître les cinq produits les plus vendus par magasin ou pour comparer les ventes de deux mois consécutifs. A partir de ces informations, la direction de l'entreprise peut déterminer le plan d'actions à réaliser afin d'augmenter les bénéfices.

Les entrepôts de données répondent à une véritable besoin en matière d'accès à l'information. Pour prendre les bonnes décisions, les décideurs ont besoin des informations correctes à un moment précis et avec un faible coût. Or, ces informations ne sont pas toujours disponibles. Un entrepôt centralise des données d'intérêt sélectionnées pour un groupe d'utilisateurs, de telle sorte que l'accès devient rapide, moins coûteux et plus efficace. Les données d'un entrepôt possèdent les caractéristiques suivantes :

- **Orientation sujet.** Les données d'un entrepôt s'organisent par sujets. Cette organisation permet de rassembler toutes les données pertinentes à un sujet et nécessaires aux besoins d'analyse, qui se trouvent réparties au travers des structures fonctionnelles d'une entreprise. L'entrepôt de données de la chaîne s'organise autour des sujets les plus importants pour la compagnie, c'est-à-dire les *produits* qui ont été vendus dans les *magasins* au cours du *temps*.

- **Intégration.** Le contenu d'un entrepôt est le résultat de l'intégration de données en provenance de multiples sources ; ainsi, toutes les données nécessaires pour réaliser une analyse particulière se trouvent dans l'entrepôt. Dans notre exemple, l'entrepôt de la chaîne stocke les données intégrées à partir des bases et fichiers utilisés par chaque magasin pour enregistrer les ventes effectuées.

- **Histoire.** Les données d'un entrepôt représentent l'activité d'une entreprise pendant une longue période. Cette caractéristique donne la possibilité de comprendre un phénomène au cours du temps et de faire des prévisions. Dans notre exemple, cette période comprend les dix dernières années, la période intéressant les analystes.

- **Non-volatilité.** Les données chargées dans l'entrepôt sont surtout utilisées en interrogation et ne peuvent pas être modifiées, sauf dans les cas de rafraîchissement. L'entrepôt de la chaîne de magasins peut être rafraîchi, par exemple, à la fin de chaque journée.

L'objectif des entrepôts de données est d'être un support pour les applications de traitement analytique en ligne (OLAP, de l'anglais *On-Line Analytical Processing*) [Cod93]. Ce type d'applications se caractérisent par une vision multidimensionnelle des données de l'entrepôt et l'analyse de ces données au travers de l'interrogation interactive. Il s'agit essentiellement de pouvoir analyser des indicateurs (ou mesures) d'une activité selon différents axes d'analyse (ou dimensions). Par exemple, la direction de la chaîne de magasins peut analyser les ventes par produit et par magasin et ce dans le temps.

Les exigences en termes de fonctionnalités et de performances des applications analytiques sont différentes de celles des applications transactionnelles (OLTP, de l'anglais *On-Line Transaction Processing*) habituellement supportées par les bases de données opérationnelles. Les applications transactionnelles, qui assurent le fonctionnement quotidien d'une entreprise, se caractérisent par un grand nombre de transactions courtes et fréquentes de mise à jour. En revanche, la fonctionnalité des applications OLAP se caractérise par l'interrogation d'un nombre potentiellement important de données historiques.

Le tableau 1.1 résume les principales différences entre les bases opérationnelles et les entrepôts de données [WB97]. Les bases opérationnelles stockent des données courantes alors que les entrepôts stockent des données sur de longues périodes de

Caractéristique	Bases Opérationnelles	Entrepôts de Données
Données	Courantes	Historiques
Usage	Support opérationnel de l'entreprise	Support de l'analyse de l'entreprise
Unité de travail	Transaction	Requête
Nombre de données manipulées	Dizaines	Millions
Mode d'accès	Lecture/Écriture	Lecture (principalement)
Type d'utilisateur	Employé	Décideur
Nombre d'utilisateurs	Mille	Cent

TAB. 1.1 – *Principales différences entre les bases opérationnelles et les entrepôts*

temps. Pour cette raison, la taille d'un entrepôt est souvent de l'ordre du téraoctet. La conception des bases opérationnelles vise à améliorer les performances du traitement de transactions. En revanche, la conception des entrepôts de données vise à améliorer les performances du traitement des requêtes complexes sur des grosses quantités de données.

Les entrepôts de données présentent plusieurs avantages pour l'aide à la décision. Il y a cependant deux problèmes qui se posent : comment construire et rafraîchir ces entrepôts et comment les exploiter (cf. figure 1.1). La construction d'un entrepôt est une activité complexe qui nécessite la réconciliation des données émanant de multiples sources d'information fortement distribuées, hétérogènes et autonomes. Remarquons que la réconciliation de données n'est pas un problème particulier des entrepôts, mais qu'il concerne d'autres domaines comme les systèmes multibases de données et les systèmes médiateurs. Un problème spécifique aux entrepôts est leur rafraîchissement, c'est à dire comment propager les mises à jour des sources vers l'entrepôt. La recherche actuelle vise à proposer des solutions à la détection automatique des mises à jour des sources et à leur propagation vers l'entrepôt.

L'exploitation des entrepôts passe par les applications d'analyse. Il faut souligner que, même si les entrepôts sont construits pour être exploités principalement par les applications d'analyse multidimensionnelle, d'autres techniques sont souvent utilisées. Ces techniques sont très variées et vont des méthodes statistiques classiques jusqu'aux méthodes semi-automatiques d'extraction de connaissances de la fouille de données, en passant par celles relevant de la visualisation. Un problème majeur à résoudre est celui de l'interrogation efficace des gros volumes de données des entre-

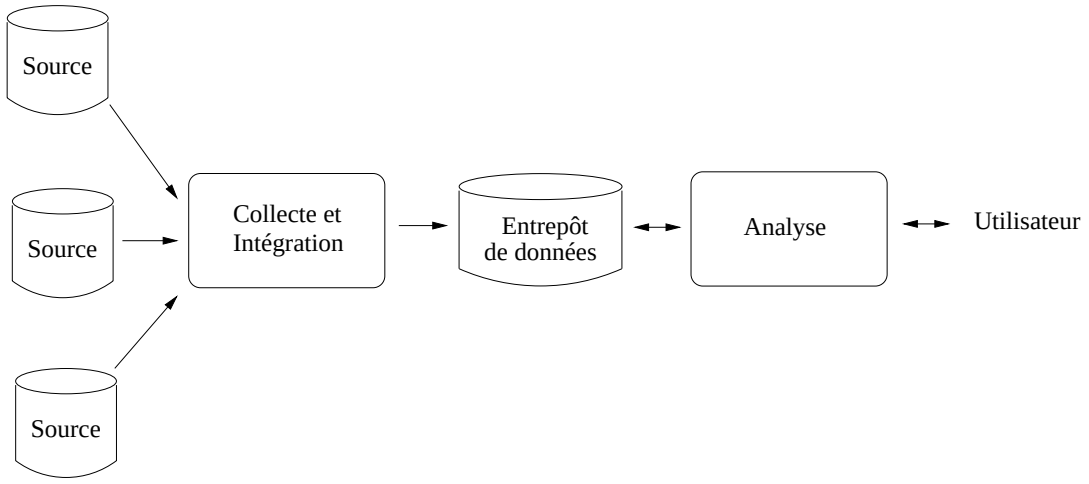


FIG. 1.1 – *Architecture générale d'un système "entrepôt de données"*

pôts. Ce problème a été l'objet d'un nombre important de travaux et on peut trouver des solutions fondées sur la réutilisation de résultats partiels ou bien sur l'exécution parallèle de requêtes.

Le nombre d'entreprises ayant construit des entrepôts a augmenté considérablement ces dernières années. En conséquence, le marché des produits et des services autour des entrepôts a eu une croissance énorme : il est passé de 2 milliards de dollars en 1994 à 8 milliards de dollars en 1998 [CD97]. Des entrepôts ont été mis en place avec succès dans plusieurs industries comme la fabrication, le commerce, les services financiers, le transport, les télécommunications et la médecine [CD97, DWI98].

1.2 Problématique et objectifs

Comme nous l'avons dit précédemment, la grande majorité des contributions dans le domaine des entrepôts ont abordé les problèmes liés à leur maintenance et à leur interrogation. Notre travail s'intéresse plutôt à un sujet qui a été peu abordé mais qui mérite attention : l'évolution du schéma des entrepôts. La problématique de notre travail s'articule plus particulièrement autour de deux axes principaux :

1. l'étude des difficultés inhérentes à l'évolution des entrepôts,
2. la spécification d'une infrastructure et des mécanismes nécessaires pour la gestion de l'évolution.

1.2.1 L'évolution du schéma des entrepôts de données

Un entrepôt de données, comme toute autre base, est la matérialisation informatique des informations d'une partie du monde réel (l'univers du discours). Le schéma d'un entrepôt spécifie les données qui pourront être matérialisées et leurs associations. Il est défini par le concepteur de l'entrepôt, en utilisant les concepts d'un modèle de données.

Le sujet de l'évolution du schéma des bases de données concerne les problèmes liés à la modification dynamique du schéma d'une base [BKKK87]. Les motivations pour permettre l'évolution de schéma des entrepôts sont diverses :

- L'univers du discours est souvent large. Dans ce cas, il est difficile de concevoir le schéma correspondant en une seule fois. La conception s'effectue de manière incrémentale. L'univers du discours n'est alors modélisé qu'à la fin d'un long processus de conception. Cette démarche est en effet adoptée par la plupart des projets de construction d'entrepôts de données.

- Les éléments de l'univers du discours sont souvent complexes et difficiles à comprendre. D'où la nécessité de faire des révisions pour compléter la conception ou corriger des erreurs.

- L'univers du discours évolue. Une évolution peut venir de nouveaux besoins ou d'un changement plus radical de la structure de l'univers du discours. Si les besoins des utilisateurs ne sont pas satisfaits, ils arrêtent d'exploiter l'entrepôt parce que les données requises pour les analyses ne sont pas disponibles. Ceci est reconnu comme une des principales causes d'échec des entrepôts.

Remarquons que l'évolution de schéma concerne tout le cycle de vie (conception, exploitation) d'une base de données. Notons au passage qu'une étude industrielle récente confirme l'importance de l'évolution du schéma des entrepôts [ZS99].

L'évolution du schéma des bases de données est un problème qui a été abordé principalement dans le contexte des bases à objets [BKKK87, Bel95]. Les contributions visent à proposer des mécanismes pour modifier (physique ou logiquement) le schéma de la base et pour adapter ses instances pour les rendre conformes au nouveau schéma. Le choix de la stratégie à utiliser dépend des applications existantes.

Dans le contexte des entrepôts de données, peu de travaux ont abordé la problématique de l'évolution du schéma. Ce problème a été identifié par Widom [Wid95]

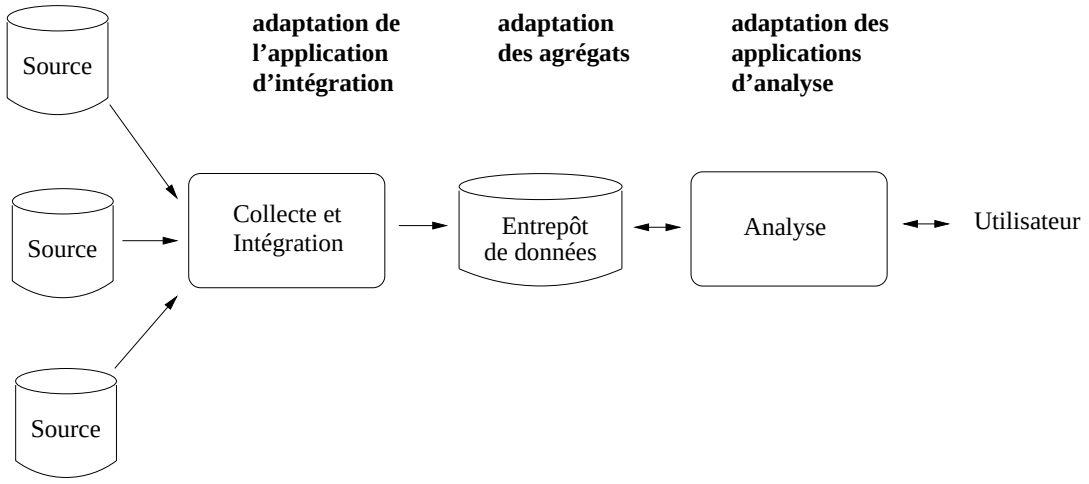


FIG. 1.2 – *Effets de l'évolution du schéma d'un entrepôt*

pour la première fois. Hurtado et al [HMV99a] ont été pionniers dans ce domaine en proposant des opérateurs pour faire évoluer des relations stockant les données des dimensions. Les travaux de Blaschka et al. [BSH99] visent à proposer un outil autorisant l'administrateur à faire évoluer le schéma d'un entrepôt à un niveau conceptuel indépendant des implantations possibles.

Faire évoluer le schéma d'un entrepôt soulève d'autres problèmes : l'adaptation des agrégats pré-calculés et l'adaptation du processus de maintenance. Un entrepôt matérialise des agrégats qui sont souvent utilisés, évitant ainsi de les recalculer à chaque fois. Les problèmes liés à l'adaptation des agrégats pré-calculés ont été explorés principalement dans un contexte relationnel. Dans [Bel98], Bellahsene propose un mécanisme basé sur l'utilisation de vues (virtuelles) pour gérer l'évolution des agrégats matérialisés.

Faire évoluer le schéma d'un entrepôt a également des conséquences sur l'application dédiée à la collecte et intégration de données des sources. En effet, celle-ci peut devenir incomplète ou incohérente vis-à-vis du nouveau schéma de l'entrepôt : incomplète, parce qu'elle doit calculer des instances qui n'étaient pas calculées avant de modifier le schéma (quand on ajoute des nouvelles données dans l'entrepôt), ou incohérente, parce que les instances qu'elle calcule ne correspondent pas au nouveau schéma (quand on modifie une partie du schéma). Dans les deux cas, des nouvelles sources peuvent être impliquées ou des anciennes sources peuvent être ignorées.

En conclusion, nous pouvons dire qu'il devient indispensable d'assurer l'évolution de schéma des entrepôts si l'on veut offrir des technologies visant à être un support solide au cycle de vie des entrepôts de données.

1.2.2 Vers une infrastructure adaptable pour l'évolution des entrepôts

Il existe à l'heure actuelle deux approches principales à l'implantation d'un entrepôt : relationnelle (ROLAP) et multidimensionnelle (MOLAP). Dans l'approche ROLAP, les données sont organisées selon des schémas relationnels spécialisés (étoile, flocon de neige, constellation, etc.). Le langage de requêtes SQL et d'autres fonctionnalités des systèmes relationnels ont été adaptés à cette organisation. Dans l'approche MOLAP, les données sont organisées sous la forme de cubes. Les systèmes de bases de données qui supportent ce modèle fournissent des langages propriétaires d'interrogation pour la manipulation des cubes. Notons que d'autres systèmes (comme les systèmes à objets) sont utilisés pour implanter des entrepôts dédiés à l'OLAP. L'implantation de l'entrepôt détermine alors les mécanismes pour faire évoluer son schéma. Dans le cas d'une implantation ROLAP, il s'agit de modifier le schéma de relations du schéma en étoile. Pour une implantation MOLAP, il s'agit de modifier le schéma de cubes.

Par ailleurs, les sources associées à un entrepôt ont des caractéristiques propres : modèle de données, capacité d'interrogation et capacité de détection/notification de mises à jour. Les modèles de données possibles ne se limitent pas aux modèles structurés, comme le modèle relationnel ou le modèle à objets, mais ils comprennent aussi les modèles semi-structurés et non-structurés. Les capacités d'interrogation varient aussi d'une source à une autre, allant de celles qui autorisent l'expression et l'évaluation de requêtes complexes (le cas de la plupart des SGBD actuels) jusqu'à celles qui ne fournissent pas aucun mécanisme d'interrogation (le cas des sources d'information HTML).

Face à cette diversité, il semble nécessaire d'avoir des systèmes logiciels permettant d'un coté de représenter un entrepôt de manière abstraite afin de cacher son implantation et de l'autre coté d'accéder aux sources de manière transparente. L'évolution du schéma est alors décrite au niveau abstrait et le système est responsable de la propagation des modifications au niveau de l'implantation et de la récupération des données des sources.

Il nous semble impossible d'avoir un système qui puisse donner une solution à toutes les différents combinaisons type d'entrepôt/types (et nombre) de sources. Il nous semble plus viable d'avoir une infrastructure à la base de la création de systèmes adaptés selon les besoins courants. Nous parlons ainsi, par abus du langage, d'une infrastructure adaptable pour l'évolution des entrepôts de données.

1.2.3 Objectif de la thèse

L'objectif de notre travail est la spécification et l'implantation d'une infrastructure autorisant l'évolution d'entrepôts de données. L'infrastructure doit avoir les propriétés suivantes :

- *Modularité* pour faciliter sa construction et maintenance
- *Adaptabilité* pour régler son fonctionnement selon les diverses implantations

1.3 Démarche et contribution

Nous avons réalisé une analyse approfondie de l'état de l'art du domaine [BG98, BGCA01]. Elle nous a permis d'identifier les caractéristiques des entrepôts et les principaux besoins des utilisateurs en matière d'évolution. Prenant comme base cette analyse, nous avons spécifié un modèle de données multidimensionnelles et un langage de description associé, ainsi qu'une infrastructure pour gérer l'évolution du schéma des entrepôts. Nous avons ensuite mis en oeuvre notre infrastructure et nous avons mené une expérimentation de son utilisation.

1.3.1 Modèle multidimensionnel de données

L'étude de l'état de l'art des entrepôts nous a révélé qu'il n'existe pas à l'heure actuelle un modèle multidimensionnel accepté de manière consensuelle par la communauté du domaine [BGCA99]. Pour cette raison, nous avons défini un modèle de référence. Dans notre modèle, le schéma et l'instance d'une dimension ou d'un cube sont clairement séparés. Une autre caractéristique importante est que les dimensions et les cubes peuvent avoir des structures complexes. Associé à ce modèle, nous avons également spécifié un ensemble de primitives d'évolution des schémas multidimensionnels. Cet ensemble comprend la création de nouveaux schémas de dimension et

schémas de cube, la suppression d'anciens schémas et la modification de schémas existants.

1.3.2 Langage de description de données multidimensionnelles

Nous avons spécifié MDL (*Multidimensional Data Definition Language*), un langage de description de données multidimensionnelles. Il est fondé sur le modèle multidimensionnel et les opérateurs d'évolution de schémas multidimensionnels précédemment évoqués. Nous avons conçu ce langage pour être un moyen de communication entre un gestionnaire d'entrepôt et ses utilisateurs. En utilisant des expressions MDL, un utilisateur peut créer et faire évoluer le schéma multidimensionnel de l'entrepôt.

Les principales caractéristiques de MDL sont les suivantes :

- MDL autorise la création de schémas multidimensionnels complexes. Il autorise en particulier la création de schémas de dimension avec plusieurs niveaux et de schémas de cube avec une ou plusieurs mesures.
- MDL autorise l'évolution de ces schémas. Il autorise plus précisément l'ajout, la suppression et la modification de schémas de dimension et de schémas de cube.

1.3.3 Infrastructure adaptable pour l'évolution des entrepôts

Dans ce travail nous proposons une infrastructure adaptable pour le développement de systèmes, appelés *gestionnaires d'entrepôt*, qui gèrent élégamment l'évolution des entrepôts de données. Un gestionnaire autorise l'administrateur à créer et à modifier le schéma d'un entrepôt même en présence de données. Il l'autorise également à définir et à modifier le processus de maintenance à l'exécution.

Un gestionnaire d'entrepôt implante notre modèle multidimensionnel de données qui permet d'un coté la manipulation du schéma en termes abstraits, et d'un autre coté, l'intégration des données, récupérées à partir de sources réparties, qui sont ensuite stockées dans l'entrepôt. L'administrateur décrit un schéma multidimensionnel et c'est le gestionnaire qui le traduit dans un schéma décrit en termes du modèle d'implantation. Un gestionnaire autorise l'administrateur à faire évoluer le schéma multidimensionnel. La réorganisation interne de l'entrepôt est réalisé par le gestionnaire.

Un gestionnaire d'entrepôt est construit comme un ensemble de composants. On définit ainsi une architecture générique représentant le cadre d'intégration de ces

composants. Pour construire un nouveau gestionnaire, il est nécessaire de prendre des composants appropriés et de les composer dans le cadre de l'architecture générique pour définir une configuration particulière. L'activité principale est donc l'intégration de composants et non leur programmation. Un des avantages de cette approche est qu'elle réduit le temps de développement des gestionnaires à travers la réutilisation des composants. Elle apporte également l'adaptabilité nécessaire pour répondre à des besoins particuliers.

1.3.4 Validation expérimentale

Pour valider notre approche, nous avons implanté un prototype de gestionnaire d'entrepôt. Ce prototype assure l'évolution d'un entrepôt de données relationnelles. Il a été développé en utilisant le langage Java pour sa programmation et les technologies XML pour la représentation et l'échange de données.

1.4 Organisation du document

La suite de ce document est organisée de la manière suivante :

- Le chapitre 2 est une analyse de l'état de l'art concernant les entrepôts de données. Nous décrivons les organisations logique et physique des entrepôts. Nous abordons ensuite les aspects liés à l'intégration de données en provenance de sources autonomes et hétérogènes pour la construction et le rafraîchissement des entrepôts, avant de discuter des aspects relatifs à leur interrogation et analyse. Nous analysons également les travaux sur l'évolution du schéma.

- Le chapitre 3 introduit notre modèle multidimensionnel de données et le langage associé. D'abord, nous définissons les notions de schéma et de instance pour les dimensions et les cubes. Puis, nous présentons nos opérateurs d'évolution de schémas multidimensionnels. Ensuite, nous introduisons le langage MDL, en expliquant des exemples d'utilisation. Enfin, nous concluons ce chapitre en discutant les similarités et les différences de notre modèle et de MDL vis-à-vis d'autres travaux existants.

- Le chapitre 4 présente notre infrastructure pour la construction des gestionnaires d'entrepôt. Nous détaillons les principales fonctions des gestionnaires

(création/évolution du schéma de l'entrepôt et construction de l'entrepôt) avant de définir l'infrastructure pour sa construction.

- Le chapitre 5 propose une implantation de notre infrastructure. Nous présentons tout d'abord la plate-forme logicielle adoptée pour la construire. Ensuite, nous présentons les fonctions implantées. Enfin, nous décrivons les expérimentations que nous avons réalisé.

- Le chapitre 6 conclut ce document. Nous présentons un bilan du travail effectué mettant l'accent sur les apports de notre recherche et donnons quelques perspectives pour des recherches futures.

Chapitre 2

Les entrepôts de données

Ce chapitre présente une étude approfondie de l'état de l'art du domaine des entrepôts de données. L'objectif de cette étude est d'analyser les éléments caractérisant les entrepôts de données afin d'établir les mécanismes permettant leur évolution. Tout d'abord, nous introduisons quelques concepts généraux (cf. section 2.1). Nous décrivons ensuite les organisations logique (cf. section 2.2) et physique (cf. section 2.3) des entrepôts. Ensuite, nous abordons les aspects liés à l'intégration de données en provenance de sources autonomes et hétérogènes pour la construction et le rafraîchissement des entrepôts (cf. section 2.4). Nous parlons aussi des aspects relatifs à leur interrogation (cf. section 2.5) et des techniques qui peuvent être utilisées pour analyser les entrepôts (cf. section 2.6). Nous présentons les travaux sur l'évolution de schéma des entrepôts (cf. section 2.7). Enfin, nous concluons (cf. section 2.8).

2.1 Concepts généraux

Dans cette section, nous présentons les notions de base des entrepôts de données : définition et structure interne.

2.1.1 Entrepôts et magasins de données

Un entrepôt de données est une collection de données exploitées par les applications d'aide à la décision [Inm92, Inm95]. Considérons par exemple une chaîne de magasins répartis géographiquement dans plusieurs villes. La direction de l'entreprise voudrait analyser les ventes réalisées pour décider d'activer ou non une nouvelle cam-

pagne publicitaire. Un entrepôt va donc stocker les informations nécessaires pour réaliser de telles analyses.

Les entrepôts de données visent plus précisément à être un support pour les applications OLAP. Ces applications se caractérisent par l'analyse multidimensionnelle de données. Les données sont perçues par l'utilisateur comme étant organisées en cubes, dont les cellules contiennent des objets d'analyse ou mesures, déterminées par une collection d'axes d'analyse ou dimensions. Dans ce contexte, l'interrogation vise la manipulation de cubes à différents niveaux de détail des données (selon le niveau d'agrégation des dimensions). Comme nous verrons plus tard, ce type d'analyses influence les organisations logique et physique ainsi que les capacités des langages d'interrogation des entrepôts.

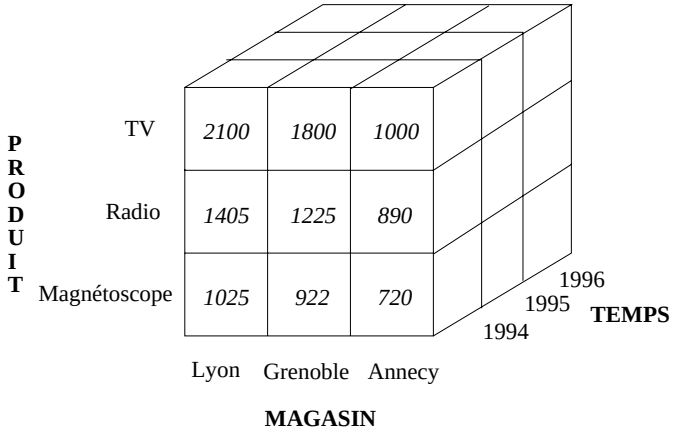


FIG. 2.1 – *Le cube de données Ventes*

Supposons que la direction de la chaîne de magasins décide d'étudier les ventes des produits. En particulier, la direction veut analyser en détails la quantité d'unités vendues. Les dimensions associées à cette mesure sont :

- La dimension *Produit*. Elle fournit les propriétés de chacun des produits : code, nom, etc. Elle fournit également la hiérarchie des marchandises. Classiquement, les produits appartient à des catégories, lesquelles vont dans un rayon déterminé.

- La dimension *Magasin*. Elle constitue une dimension géographique élémentaire de notre activité. On considère que chacun des supermarchés est localisé dans une ville, laquelle est localisée dans une région, laquelle appartient à un pays.

– La dimension *Temps*. Elle fournit les informations sur chacun des jours appartenant à la période d'observation des ventes. Elle associe à chaque jour son nom (lundi, par exemple) et une propriété indiquant s'il s'agit d'un jour férié ou non. Les jours sont rattachés à des mois, et les mois sont rattachés à des années.

La figure 2.1 montre le cube *Ventes* de notre exemple. Dans ce cube, la mesure *quantité* est déterminée par les dimensions *Produit*, *Magasin* et *Temps*. Dans la figure, on peut voir que 1000 téléviseurs ont été vendus en 1994 dans le magasin situé à Annecy. La direction de la chaîne peut utiliser ce cube pour calculer les ventes par jour, par mois ou par année. Elle peut également calculer les ventes par ville, par région ou par pays.

Un concept lié aux entrepôts est celui du magasin de données (*data mart*). Il s'agit d'un entrepôt spécialisé qui se focalise sur un sous-ensemble de sujets particuliers. Les magasins de données sont souvent utilisés comme des éléments supplémentaires à l'entrepôt, pour satisfaire les besoins d'analyse au niveau des départements de l'entreprise. Par exemple, un magasin de la chaîne peut avoir un *data mart* pour analyser ses ventes de produits de manière indépendante du reste des magasins. Il faut souligner que les idées et les concepts sur les entrepôts que nous développons dans ce chapitre sont applicables également aux magasins de données.

2.1.2 Structure d'un entrepôt de données

Les données d'un entrepôt se structurent selon deux axes : synthétique et historique (*cf.* figure 2.2) [Fra97]. L'axe synthétique établit une hiérarchie d'agrégation et comprend les données détaillées (qui représentent les événements les plus récents au bas de la hiérarchie), les données agrégées (qui synthétisent les données détaillées) et les données fortement agrégées (qui synthétisent à un niveau supérieur les données agrégées). L'axe historique comprend les données détaillées qui sont conservées au cours du temps.

Les métadonnées contiennent des informations sur la création, la gestion et l'utilisation de l'entrepôt. Celles-ci peuvent être classées en trois catégories [WB97] : spécifiques à l'application, d'audit et d'administration. La première catégorie concerne les ontologies et les connaissances particulières au domaine d'application. Les métadonnées d'audit comprennent des statistiques sur l'entrepôt.

Les métadonnées d'administration peuvent être encore classifiées en données schématiques, sémantiques, conceptuelles et relatives aux utilisateurs. Les métadonnées

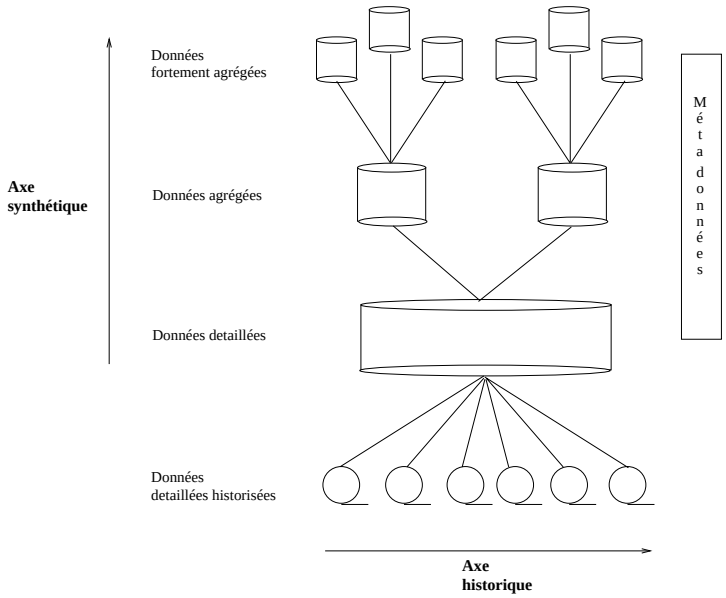


FIG. 2.2 – *La structure d'un entrepôt de données*

schématiques comprennent des informations sur les éléments du schéma de l'entrepôt, les schémas des sources et les règles de correspondance entre ceux-ci et le schéma de l'entrepôt. On trouve également l'information concernant l'extraction de données sources, les règles de transformation et les politiques de rafraîchissement. Les métadonnées sémantiques comprennent les règles de correspondance entre entités similaires en provenance de sources différentes. Les métadonnées conceptuelles contiennent les informations concernant les dimensions de l'entrepôt. Finalement, les métadonnées sur les utilisateurs de l'entrepôt contiennent des informations sur les groupes et les droits d'accès associés.

Remarquons que les métadonnées associées à un entrepôt sont plus nombreuses que celles associées à une base de données classique. Pour cette raison, elles sont souvent gérées par un système (nommé *référentiel*) qui autorise le partage des métadonnées entre les composants des systèmes pour la mise en place et l'utilisation des entrepôts [CD97].

2.2 Organisation logique

Nous présentons ici l'organisation logique des entrepôts. Nous décrivons d'abord les schémas relationnels et puis nous abordons les schémas multidimensionnels.

2.2.1 Schémas relationnels

L'utilisation de schémas de relations fortement normalisées n'est pas adaptée pour la conception du schéma logique d'un entrepôt [KS95b]. Ceci s'explique par le grand nombre de jointures et de restrictions occasionnées pour les traitements des requêtes typiques des systèmes OLAP. Pour éviter ces inconvénients, des schémas dénormalisés sous forme d'étoile, de flocon de neige et de constellation de faits ont été proposés.

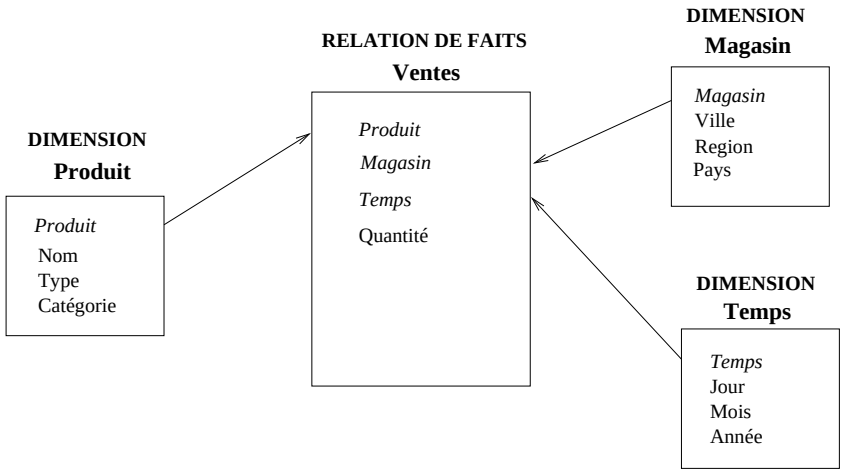


FIG. 2.3 – *Schéma en étoile*

Dans le schéma en étoile [Kim96], les mesures sont représentées par une *relation de faits* et chaque dimension par une *relation de dimension*. La relation de faits référence les relations de dimension, en utilisant une clé étrangère pour chacune d'elles, et stocke les valeurs des mesures pour la combinaison de ces clés étrangères (qui constitue la clé primaire de la relation). Autour de cette relation figurent les relations de dimension qui regroupent les caractéristiques des dimensions. La relation de faits est normalisée et peut atteindre une taille importante en nombre de n-uplets. Les relations de dimension sont dénormalisées et sont en général de petite taille. La figure 2.3 montre le schéma en étoile où la relation de faits **Ventes** stocke la quantité de produits vendus et les relations **Produit**, **Magasin** et **Temps** comportent les informations sur les dimensions *Produit*, *Magasin* et *Temps*, respectivement.

Le schéma en étoile ne reflète pas de manière explicite les hiérarchies associées à une dimension et ces informations sont définies dans les métadonnées conceptuelles (*cf.* section 2.1.2). Le schéma en forme de flocon de neige a été proposé pour rendre ces informations explicites. Il normalise les relations de dimension, réduisant la taille de

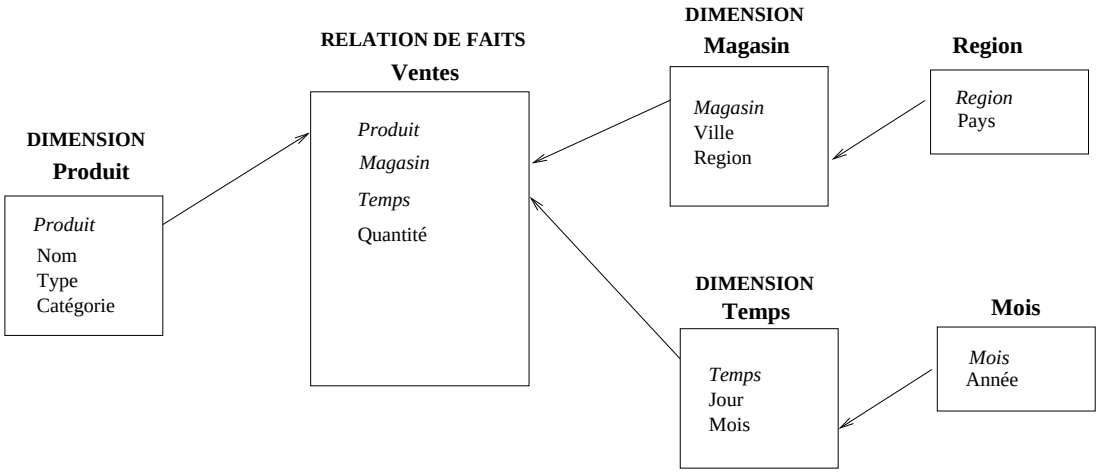


FIG. 2.4 – *Schéma en flocon de neige*

chacune d'elles et permettant ainsi de formaliser la notion de hiérarchie au sein d'une dimension [Fra97]. La figure 2.4 montre les relations nécessaires pour représenter un niveau des hiérarchies de dimensions *Magasin* et *Temps*.

Il est possible d'avoir plusieurs relations de faits pour représenter les situations dans lesquelles les mesures ne sont pas déterminées par exactement le même ensemble de dimensions. Dans ce cas, les relations de faits forment une *famille* [KS95b] qui partage plusieurs relations de dimension mais où chaque membre possède ses dimensions propres. Bien entendu, si les relations de faits partagent une dimension, il faut vérifier que celle-ci est la même. Le schéma résultant s'appelle constellation de faits [CD97].

Les schémas relationnels adaptés aux entrepôts possèdent plusieurs avantages par rapport à ceux qui sont normalisés. L'étoile, le flocon et la constellation autorisent l'expression de mesures, de dimensions et de hiérarchies, d'une manière simple qui permet de les distinguer clairement tout en restant faciles à comprendre. Ces schémas facilitent également l'accès aux mesures, même si la taille d'une relation de faits est souvent importante, parce que le nombre de jointures est plus petit.

2.2.2 Schéma multidimensionnel

Le cube est le concept central du modèle de données des systèmes multidimensionnels [Ken95, App98, Arb98]. Un cube organise les données en une ou plusieurs *dimensions* qui déterminent une *mesure* d'intérêt. Considérons un schéma multidimensionnel pour une chaîne de magasins. Dans ce cas, la mesure d'intérêt est la

quantité d'un *Produit* vendu dans un *Magasin* (représenté par sa ville de localisation) à un instant du *Temps*. La partie gauche de la figure 2.5 montre le schéma du cube *Ventes*. La partie gauche de la figure 2.5 présente une instance de ce cube.

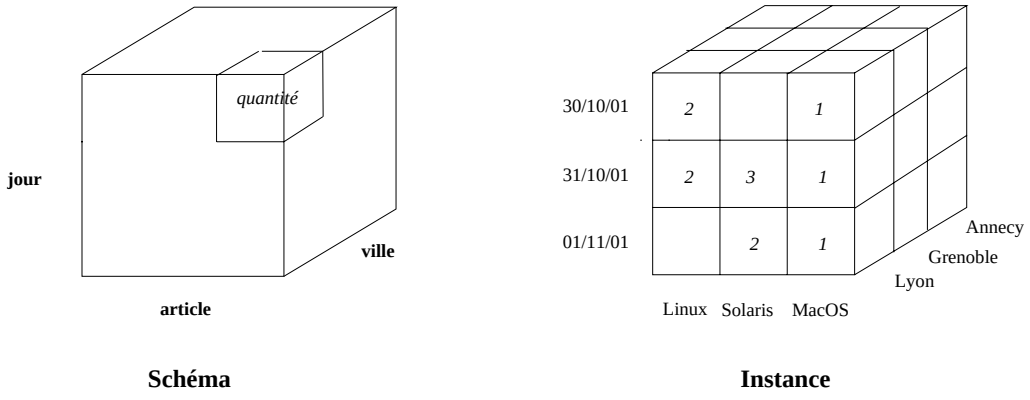


FIG. 2.5 – *Le schéma et une instance possible du cube Ventes*

Associées aux dimensions, on trouve des hiérarchies d'agrégation qui permettent d'observer les données à différents niveaux de granularité. Une dimension peut avoir plusieurs hiérarchies associées, chacune spécifiant une relation d'ordre distincte entre ses attributs. La partie gauche de la figure 2.6 montre le schéma de la dimension *Magasin*. La hiérarchie « ville→région→pays » est associée à cette dimension. La partie droite de la figure 2.6 montre une instance possible de cette hiérarchie, qui regroupe les villes de Paris et Créteil dans la région Ile-de-France et les régions Rhône-Alpes et Ile-de-France dans le pays France.

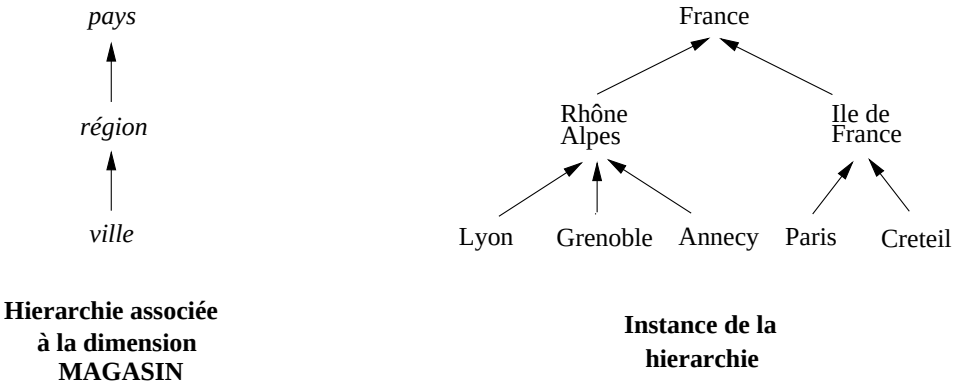


FIG. 2.6 – *Schéma et instance de la dimension Magasin*

Remarquons qu'il existe diverses contributions [AGS95, LW96, GL97, TDV97, CT98, Vas98] ayant proposé des formalismes de cube pouvant être implantés en utilisant divers modèles de bases de données (*cf.* Annexe A). Dans ces travaux, on distingue une tendance générale qui consiste à définir les notions de schéma et d'instance pour les dimensions et les cubes. Le schéma d'une dimension prend la forme d'un ordre partiel entre les niveaux au sein d'une dimension, autorisant la définition des hiérarchies simples ou multiples [Vas98, CT98]. Une instance de dimension est modélisée comme un famille de fonctions déterminant les éléments d'un niveau en rapport avec les éléments d'un niveau supérieur [Vas98]. En ce qui concerne le schéma d'un cube, celui-ci est déterminé par un ensemble de niveaux (chaque niveau appartenant à une dimension différente) et une mesure [CT98, Vas98] ou plusieurs [TDV97]. Une instance de cube est souvent définie comme un ensemble de n-uplets (nommés cellules), où chaque cellule instancie les niveaux et les mesures définis dans le schéma du cube [TDV97, CT98, Vas98].

2.3 Le niveau physique des entrepôts

Le stockage de données et les méthodes d'accès associées ont été étudiés depuis plusieurs années. Cependant, les gros volumes gérés par les entrepôts et les besoins d'accès performant lors de l'interrogation ont remis en cause les techniques traditionnelles. Dans cette section, nous présentons les techniques de stockage et d'accès aux données adaptées aux entrepôts.

2.3.1 Stockage de données dans un système multidimensionnel

Le stockage de données dans un système multidimensionnel vise à être un support pour le modèle multidimensionnel (le cube). Le stockage d'un cube est réduit au problème de stockage d'un vecteur multidimensionnel [Sho97, DSHB98], où chaque dimension du cube est associée à un axe du vecteur et chaque valeur d'une dimension est associée à une position de l'axe correspondant à la dimension. Les cellules du vecteur, c'est-à-dire les intersections entre les positions des axes, contiennent les mesures.

Considérons à titre d'illustration la figure 2.7, où nous pouvons voir les correspondances entre les dimensions *Magasin*, *Produit* et *Temps* et les axes d'un vecteur 3-dimensionnel pour stocker le cube *Ventes*. Nous observons également les correspon-

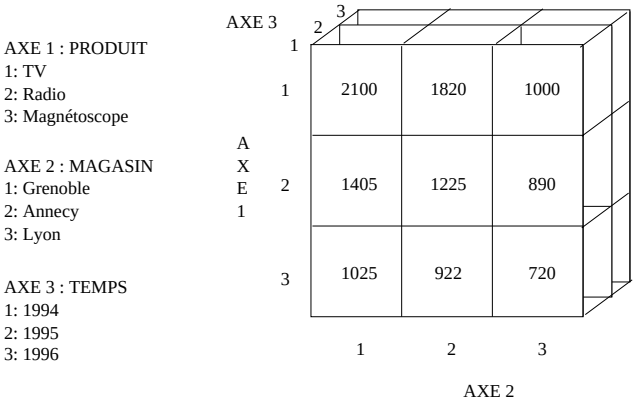


FIG. 2.7 – *Vecteur multidimensionnel pour le stockage du cube Ventes*

dances entre les valeurs de chaque dimension et les positions des axes. La cellule à la position AXE 1=2, AXE 2=1 et AXE 3=1 stocke la valeur 1405, ce qui signifie que 1405 radios ont été vendues en Grenoble en 1994.

Il existe deux approches pour le stockage d'un vecteur en mémoire secondaire [GC97]. La première consiste à stocker les cellules de manière consécutive en parcourant le vecteur à l'aide des indices associés à chaque axe, ce qui suppose l'existence d'un ordre prédéfini entre les axes. Le principal avantage de cette approche est qu'un calcul mathématique simple peut déterminer l'adresse relative d'une cellule spécifique. L'inconvénient est que des cellules adjacentes dans le vecteur peuvent être stockées à des positions très éloignées en mémoire secondaire. Celle-ci devra donc être lue plusieurs fois lors de l'accès à un ensemble spécifique de cellules adjacentes. La deuxième approche vise à résoudre ce problème, en coupant le vecteur en plusieurs blocs de taille égale. L'avantage est alors que la distance entre les positions de mémoire secondaire pour les cellules adjacentes est réduite.

Le problème principal posé par la représentation d'un cube sous la forme d'un vecteur multidimensionnel est que celui-ci est creux. Ceci est dû au fait qu'en général seulement un nombre réduit de cellules d'un cube ont une valeur de mesure associée. Par exemple, si l'on considère que les produits ne sont pas tous vendus dans tous les magasins pendant les mêmes périodes de temps, alors les cellules (du cube *Ventes*) valorisées pour la quantité de produits vendus seront peu nombreuses. Ce sujet a déjà été abordé pour la représentation de vecteurs creux en deux ou trois dimensions. Dans le domaine des systèmes multidimensionnels, les travaux visent à la compression de l'espace de stockage des vecteurs de plus de trois dimensions [ZDN97].

2.3.2 Méthodes d'accès indexées

Les entrepôts de données se prêtent bien à l'utilisation de méthodes d'accès indexées plus complexes que celles utilisées dans les bases de données. En effet, comme la mise à jour des données entraîne celle des index associés, l'utilisation des méthodes décrites ci-après affectent les performances dans le cas de mises à jour fréquentes. Cela n'est pas le cas pour les entrepôts puisque le chargement des données est en général fait de manière périodique par un processus dédié et les mises à jour sont donc peu fréquentes.

Indexation binaire

Un index binaire est une séquence de bits, où chacun d'eux indique si l'enregistrement associé présente une propriété donnée, par exemple qu'un champ prend une valeur particulière. La figure 2.8 montre un index binaire pour la propriété `type = électronique` d'un fichier de produits. Les index binaires ont été introduits il y a plusieurs années [O'N87] et ce n'est que récemment que leur efficacité lors de l'accès à des gros volumes de données a été montrée [OG95]. À l'heure actuelle, plusieurs SGBDR commerciaux fournissent des index binaires [Red96, Ora98b].

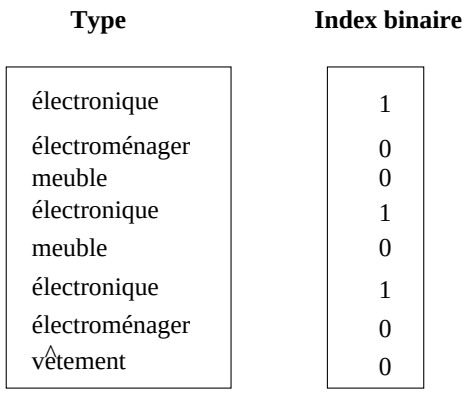


FIG. 2.8 – *Index binaire*

Les index binaires se prêtent bien à la recherche de données. Par exemple, pour chercher les produits électroniques en utilisant l'index binaire de la figure 2.8, il suffit de trouver les bits à 1 et d'accéder ensuite aux enregistrements correspondants. Ils sont utiles également pour le traitement de requêtes dont les réponses peuvent être calculées à partir de l'index. Par exemple, pour calculer le nombre de produits élec-

troniques, il suffit de compter les bits à 1. Il est possible d'exécuter des opérations logiques, comme `ET` et `OU`, entre deux index binaires pour retrouver des enregistrements combinant plusieurs propriétés [OQ97].

Un autre avantage des index binaires est que, pour des attributs à nombre limité de valeurs, l'espace nécessaire pour leur stockage est réduit. Dans ce cas, l'index binaire peut être géré en mémoire, ce qui améliore les performances du système gérant l'entrepôt. Évidemment, en présence d'attributs avec un nombre important de valeurs, l'espace de stockage augmente. Dans ce cas, il faut gérer une quantité importante d'index binaires qui contiennent presque dans leur totalité des bits à zéro. Des techniques de compression de données sont alors utilisées pour éviter ces problèmes [OQ97].

Indexation de vecteurs multidimensionnels

Les méthodes d'accès indexées à un vecteur multidimensionnel entraînent la division de l'ensemble global de dimensions en deux sous-ensembles disjoints : celui des dimensions denses et celui des dimensions clairsemées. Les dimensions denses forment un sous-cube dans lequel chacune des cellules a une valeur de mesure ; les dimensions restantes sont considérées comme clairsemées. Un index dans ce contexte est une structure qui stocke toutes les combinaisons de valeurs des dimensions clairsemées, et chaque combinaison valable a un pointeur vers un vecteur qui représente un cube formé à partir des dimensions denses [Col96].

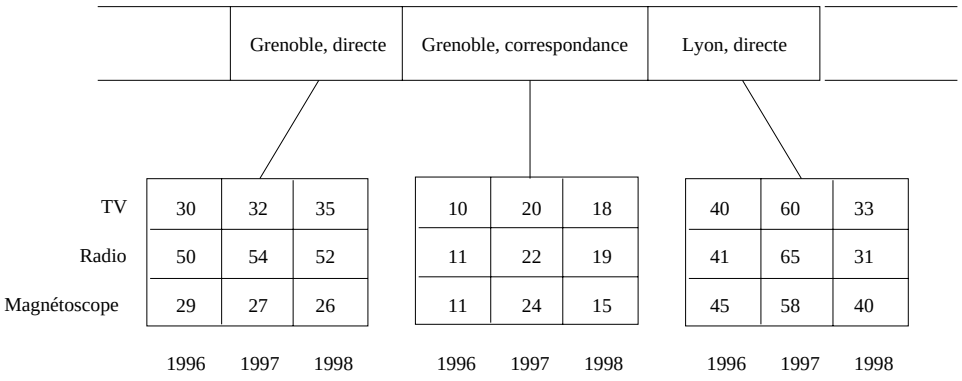


FIG. 2.9 – *Indexation de vecteurs multidimensionnels*

Supposons que certains magasins sélectionnés sont autorisés à vendre des produits en magasin ou par correspondance. Le cube *Ventes* de la figure 2.1 devient un hypercube en quatre dimensions, dont la nouvelle dimension *Distribution* peut prendre les

valeurs *direct* ou *correspondance*. Comme la vente par correspondance est autorisée seulement dans un petit nombre de magasins, on peut considérer les dimensions *Magasin* et *Distribution* comme étant clairsemées. Les dimensions *Produit* et *Temps* sont considérées comme denses. Dans la figure 2.9 nous pouvons voir un exemple d'index de ce type. Dans cet exemple, la combinaison *Magasin* =`grenoble` et *Distribution* = `directe` a un pointeur vers un vecteur dense. Cette méthode d'accès indexée aux données est adéquate lorsque la taille de l'index permet de le gérer en mémoire. Dans le cas contraire, il faudra accéder plusieurs fois à la mémoire secondaire pour rechercher les données [Sar97].

L'exploration de nouvelles méthodes d'indexation pour les données multidimensionnelles est un sujet de recherche intéressant. Les méthodes d'indexation proposées dans les bases de données spatiales [SCR+99] (par exemple, les Quadtrees [FB74], les arbres R [Gut84], les arbres R+ [SRF87] et les arbres R* [BKSS90]) semblent de bonnes candidates pour les entrepôts [Sar97].

2.3.3 Partitionnement

Les entrepôts de données se prêtent bien à l'utilisation des techniques de partitionnement, car des requêtes complexes sur de grandes quantités de données peuvent être exécutées en parallèle améliorant éventuellement les temps de réponse (*cf.* section 2.5). De plus, le partitionnement de l'entrepôt a des avantages par rapport au chargement de données (*cf.* section 2.4.2).

Dans le cas des systèmes relationnels, les techniques de partitionnement de relations ont été étudiées depuis de nombreuses années. Les travaux récents dans le contexte des entrepôts visent au partitionnement vertical de la relation de faits d'un schéma en étoile [VDR97]. Dans les systèmes multidimensionnels, il est possible de distinguer deux types de partitionnement de cubes : celui qui vise à sélectionner des sous-ensembles de données (sous-cubes) du cube original, et celui qui vise à diviser un cube en deux selon des dimensions différentes [SMKK98].

2.4 Construction et rafraîchissement

Les données intégrées dans un entrepôt ont une grande valeur pour une entreprise mais leur intégration n'est pas une tâche simple, car le passage des données sources vers l'entrepôt nécessite un processus complexe de transformation. De plus, les chan-

gements dans les sources doivent être régulièrement propagés vers l'entrepôt pour le mettre à jour. Dans cette section nous abordons ces aspects, prenant comme cadre pour la discussion l'architecture (*cf.* figure 2.10) proposée dans le contexte du projet WHIPS (*Warehouse Information Project at Stanford*) à l'Université de Stanford aux États-Unis [Wid95].

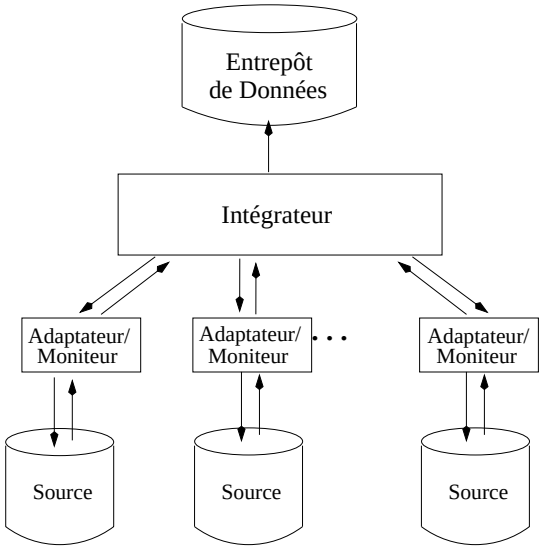


FIG. 2.10 – *Architecture des systèmes pour l'intégration de données*

2.4.1 Adaptateur/moniteur

Associé à chaque source on trouve un adaptateur/moniteur qui a deux fonctions : (1) transformer le schéma source et ses instances en une représentation intermédiaire et (2) détecter automatiquement les changements dans la source pour les propager vers l'intégrateur. Dans la suite, nous détaillons ces activités.

Les sources peuvent avoir des modèles de données et des schémas hétérogènes. La partie adaptateur transforme les données d'une source en une représentation intermédiaire et comprend également une interface pour interroger la source. Par exemple, un adaptateur relationnel associé à une source HTML présente les données que celle-ci stocke selon un format relationnel. L'intégrateur tire profit de cette uniformité de représentation pour faire la fusion des sources (*cf.* section 2.4.2).

La partie moniteur de ce module détecte automatiquement les changements de sa source et les propage vers l'intégrateur. Par rapport à la détection automatique de changements, il est possible de classifier les sources en [Wid95] :

- Sources coopératives : celles qui supportent des mécanismes de règles actives ou de notification [CVSGR00], pour notifier les changements de manière automatique.
- Sources avec journal : elles maintiennent un journal pour enregistrer leur activité, de telle sorte que les changements intéressants peuvent être identifiés.
- Sources permettant des requêtes à la demande : elles autorisent le moniteur à poser des questions de manière périodique pour pouvoir détecter les changements pertinents.
- Sources permettant les photographies : la seule façon de détecter des changements est la comparaison de photographies consécutives.

La nature diverse des sources oblige à programmer un moniteur/adaptateur spécifique pour chaque type de source, ce qui est une solution coûteuse. Il est donc important de proposer des techniques et des outils pour la génération automatique d'adaptateurs/moniteurs à partir de spécifications déclaratives [SA99].

2.4.2 Intégrateur

L'intégrateur est responsable de la réception des représentations intermédiaires des données sources, en provenance des adaptateurs, et de l'intégration (fusion) de celles-ci. L'intégration entraîne plusieurs activités de « nettoyage » liées à la transformation des données sources pour les rendre conformes au schéma de l'entrepôt et aux critères de qualité choisis [Kim96, GFSS00]. Une problématique similaire est abordée dans le domaine des systèmes multibases de données [SL90, BH92], où l'intégration des schémas est décomposée en trois activités : (1) identification des entités ayant une sémantique similaire, (2) identification et résolution des éventuels conflits entre les entités et (3) intégration des entités.

L'identification des entités à sémantique similaire peut être compliquée à cause des problèmes d'hétérogénéité structurelle que présentent les sources. Une même entité peut être représentée de différentes manières dans plusieurs schémas. Des entités

distinctes peuvent aussi être représentées de façon similaire. Les techniques de recherche de motifs sont souvent utilisées pour trouver des patrons utiles pour chercher des instances dupliquées, en dépit de variations et d'attributs manquants [HS98]. En général, l'intervention humaine est indispensable pour résoudre ces conflits.

Une fois identifiées les entités similaires, le problème est alors de résoudre les éventuels conflits qui peuvent apparaître. Ces conflits comprennent des problèmes de domaine, d'hétérogénéité sémantique ou encore d'incohérence. Les problèmes de domaine sont résolus en utilisant des techniques où il s'agit de trouver la valeur correcte d'un attribut dans une liste (domaines ou rangs) de valeurs légales. Il s'agit par exemple de trouver la bonne adresse d'un client dans la liste des adresses valides d'une ville ou de corriger un nom mal écrit en cherchant un nom similaire dans un annuaire.

Les sources peuvent également présenter des problèmes d'hétérogénéité sémantique, comme les synonymes. Les problèmes de synonymie sont traités en spécifiant un ensemble de règles de transformation pour arriver à une représentation uniforme. Par exemple, remplacer le mot « genre » par « sexe », ou bien « Féminin » par « F » et « Masculin » par « M ». Dans le cas simple, celui de la plupart des outils commerciaux, ces règles sont établies directement par l'utilisateur [ETI98, VAL98]. Aujourd'hui, des techniques issues de l'intelligence artificielle sont employées pour découvrir les règles de transformation à partir des données [GMV96]. Les règles découvertes sont ensuite utilisées pour trouver les exceptions, qui sont alors traitées comme des erreurs.

La consolidation des entités vise à les unifier au sein d'un même schéma, celui de l'entrepôt. La consolidation de schémas relationnels avec des sémantiques similaires est possible grâce à un ensemble d'opérateurs qui visent à faire l'union, la jointure ou la généralisation de deux relations, portant à la fois sur les schémas et les données qu'elles stockent [YM98]. Il faut adapter ces techniques au cas des entrepôts.

En conclusion, l'activité de l'intégrateur est complexe et, pour chaque entrepôt, celle-ci peut être spécifique. Le développement de techniques et d'outils pour la génération automatique d'intégrateurs à partir de spécifications déclaratives diminuera l'effort d'implémentation de ceux-ci. Cette problématique est abordée dans le projet *H2O* à l'Université de Colorado aux États-Unis [ZHKF95].

2.4.3 Le chargement de données dans l'entrepôt

Une fois que l'intégrateur a fusionné les données en provenance des adaptateurs, les données intégrées peuvent alors être chargées dans l'entrepôt. Le chargement est fait soit lors de la création de l'entrepôt, soit lors du rafraîchissement. Pour la création de l'entrepôt, les aspects liés à la détection automatique de changements évoqués dans la section 2.4.1 sont ignorés. Dans ce cas, le rôle des moniteurs est d'envoyer des copies complètes des sources vers l'intégrateur, qui va alors remplir l'entrepôt.

Le processus de détection de changements dans les sources et leur propagation vers l'entrepôt est connu sous le nom de rafraîchissement et il en existe deux types [WB97] : reconstruction ou incrémental. Dans le premier, il s'agit d'éliminer l'ancien contenu de l'entrepôt et de le remplir à nouveau périodiquement, avec une période qui dépend des besoins des utilisateurs et de la charge d'accès à l'entrepôt. Dans le rafraîchissement incrémental, l'entrepôt est mis à jour en utilisant les changements dans les sources détectés par le moniteur et traités par l'intégrateur. Ce type de rafraîchissement peut avoir lieu à chaque fois qu'une source change ou bien de manière périodique. Des projets de recherche comme WHIPS [HGMW+95, LZW+97] ou SIRIUS (*Supporting Incremental Refreshment of Information Warehouses*) [VGD99] à l'Université de Zurich en Suisse, étudient les mécanismes de rafraîchissement incrémental.

Les techniques incrémentales de rafraîchissement n'ont pas été suffisamment développées pour les systèmes multidimensionnels et il faut donc reconstruire l'entrepôt périodiquement. À l'heure actuelle, seulement un nombre limité de ces systèmes fournit des mécanismes pour le rafraîchissement incrémental, comme [Hyp99] qui donne la possibilité d'introduire des nouvelles partitions. Dans le cas des systèmes relationnels, le rafraîchissement incrémental de l'entrepôt a été davantage étudié. En particulier, la mise à jour incrémentale de vues matérialisées représentant des agrégats précalculés (*cf.* section 2.5) a donné lieu à une quantité importante de travaux. La plupart des techniques de maintenance de vues utilisent comme heuristique le fait que seulement une partie d'une vue change lors des modifications des relations de base [GM95, MQM97, GLT97]. Il s'agit donc de trouver l'expression qui décrit ces changements et de l'exécuter pour modifier la vue.

Pendant le rafraîchissement, l'entrepôt est souvent indisponible. Ceci est dû au fait que le chargement de données dans l'entrepôt entraîne plusieurs opérations, comme l'ajout de nouvelles données détaillées et la mise à jour des agrégats et des index, qui peuvent devenir très complexes à cause du volume des données. Il est donc important

de minimiser le temps de rafraîchissement, par exemple, en le divisant en plusieurs tâches faites en parallèle [GMLWZ98].

Avant de conclure cette section, il faut remarquer que l'architecture présentée ici est plus générale que celle des produits commerciaux existants. La collecte et l'intégration de données restent encore des processus intuitifs, réalisés à l'aide d'outils semi-automatiques. En général, ces processus sont mal documentés et ils ne peuvent pas être facilement évalués. Des méthodologies, des outils et des formalismes sont nécessaires pour faciliter la collecte et l'intégration de données, de telle façon que les erreurs éventuelles soient détectées et éliminées automatiquement, améliorant ainsi la qualité des données de l'entrepôt. Ceci est justement le but du projet européen DWQ (*Foundations of Data Warehouse Quality*) [JV97, CGL+98].

2.5 Interrogation de l'entrepôt

Dans cette section, nous abordons la problématique de l'interrogation d'entrepôts de données. Les requêtes entraînant en général des opérations coûteuses d'agrégation sur de gros volumes de données, mettent en évidence les limitations de la technologie actuelle pour l'expression et le traitement de requêtes. Nous discutons d'abord les aspects qui concernent l'expression et le traitement de requêtes dans les systèmes relationnels et multidimensionnels. Ensuite, nous expliquons le précalcul d'agrégats et, finalement, nous abordons le traitement parallèle de requêtes.

2.5.1 Expression et traitement des requêtes

Il a été reconnu que SQL, le langage d'interrogation standard des SGBD relationnels, n'est pas bien adapté à l'interrogation des entrepôts [KS95a]. En effet, l'expression des requêtes d'analyse devient très complexe parce qu'il est souvent nécessaire d'utiliser de multiples jointures et des unions. De plus, les fonctions d'agrégat offertes par SQL, telles que `COUNT` (cardinalité), `SUM` (somme) et `AVG` (moyenne), sont souvent insuffisantes pour exprimer des analyses plus complexes.

Plusieurs extensions à SQL ont été proposées pour faciliter l'expression des requêtes d'analyse. Jagadish *et al.* [JLS99] ont proposé une extension au langage SQL pour rendre plus facile l'écriture, la lecture et l'optimisation de requêtes intégrant généralement la notion de dimension et les agrégations nécessaires. Gray *et al.* [GCB+97] ont proposé l'opérateur `CUBE` pour faciliter l'expression de requêtes d'agrégation en-

traînant plusieurs `GROUP-BY`. Considérons par exemple la requête suivante sur la relation **Ventes** du schéma en étoile de la figure 2.3:

```
select         Produit, Magasin, Temps, SUM(quantite)
from           Ventes
group by cube  Produit, Magasin, Temps
```

Le résultat de l'exécution de cette requête est l'union des `GROUP-BY` pour tous les sous-ensembles possibles de $\{Produit, Magasin, Temps\}$. Pour mettre en œuvre cet opérateur, les `GROUP-BY` peuvent être modélisés de manière hiérarchique [AAD+96]. À partir de cette représentation, les opérations communes à plusieurs d'entre eux sont combinées et les calculs effectués sont réutilisés pour en calculer d'autres. L'opérateur `CUBE` est maintenant fourni par des SGBD relationnels commerciaux, par exemple [Ora98b].

En ce qui concerne les fonctions d'agrégat, les systèmes relationnels commerciaux incorporent désormais des fonctions comme `N_TILE` pour classer par pourcentage, `RATIO_TO_TOTAL` pour calculer des proportions, et d'autres de type financier [Red96]. Il est possible également d'exprimer en SQL étendu le calcul des échantillons [Ora98b], très utilisé par les outils statistiques et de fouille de données (*cf.* section 2.6).

Dans le cas des systèmes multidimensionnels, il n'existe pas de langage de requêtes standard. Tous proposent des langages procéduraux (par exemple [Ken95, Pil98]) qui autorisent l'expression de requêtes où les fonctions d'agrégat et d'autres fonctions mathématiques sont communes. Évidemment, l'expressivité de ces langages est très limitée parce qu'il faut spécifier la manière dont une requête doit être exécutée. Le traitement de requêtes dans les systèmes multidimensionnels est très performant car l'accès à une mesure correspondant à n'importe quelle combinaison de valeurs de dimensions se réduit à une seule opération d'entrée/sortie. En effet, étant donné le stockage d'un cube sous forme de vecteur multidimensionnel (*cf.* section 2.3.1), une fois qu'un sous-ensemble particulier des valeurs de dimensions est identifié dans une requête, un calcul mathématique simple détermine rapidement les adresses relatives spécifiques aux cellules appropriées du vecteur. Ainsi, des analyses complexes, comme la comparaison des ventes de cette année par rapport à celles de l'année dernière, peuvent être réalisées de manière performante.

2.5.2 Le précalcul d'agrégats

Comme nous l'avons déjà mentionné, les opérations typiques sur un entrepôt sont des agrégats. Pour éviter de calculer le résultat d'une requête de ce type à partir de zéro, il est possible de calculer par avance des agrégats et de les stocker. Ces données pourront alors être utilisées par le système lors du traitement d'autres requêtes. Il existe trois aspects liés au précalcul d'agrégats : (1) quelles agrégats précalculer ? (2) comment utiliser ces données lors de l'optimisation de requêtes ? (3) comment les maintenir lorsque les données de base ont été mises à jour ? Dans cette section, nous traitons les deux premiers aspects ; nous avons discuté le troisième dans la section 2.4.2.

Il existe deux approches principales pour le précalcul d'agrégats [HRU96] : complet et sélectif. Le précalcul complet vise à matérialiser les agrégats correspondant à toutes les combinaisons de valeurs de toutes les dimensions. Le principal inconvénient de cette approche est que l'espace de stockage croît rapidement lorsque le nombre de dimensions, et donc le nombre possible de combinaisons, augmente. Le précalcul sélectif se base sur l'observation qu'il y a des agrégats qui peuvent servir à en calculer d'autres et le problème se réduit donc à la sélection des bons agrégats.

L'utilisation des agrégats précalculés pour répondre à une requête doit être transparente pour l'utilisateur. Pour cela, il est indispensable que le système gérant l'entrepôt soit capable d'analyser une requête et de la réécrire en termes d'agrégats existants, en utilisant les métadonnées définissant les hiérarchies au sein des dimensions. Par exemple, si l'utilisateur demande les totaux annuels et que les agrégats sur ceux-ci ne sont pas disponibles, les agrégats pour les totaux mensuels peuvent être utilisés pour répondre à la requête.

Dans un SGBD relationnel, les agrégats précalculés sont représentés par des vues matérialisées [Adi81]. Celles-ci ont fait l'objet d'un nombre très important de travaux de recherche depuis plusieurs années et, aujourd'hui, nous assistons à une nouvelle activité de ce domaine motivée par leur utilisation dans les entrepôts [GM95]. Les contributions abordant le problème de la matérialisation sélective de vues proposent des solutions en considérant des contraintes d'espace de stockage et le coût de maintenance des vues sélectionnées [HRU96, BPT97, Gup97, CHS01, KMP02]. De nombreux travaux ont été consacrés également à la réécriture de requêtes en terme de vues matérialisées associées à une relation de base [GHQ95, LMSS95, SDJL96, CNS99].

Les systèmes multidimensionnels effectuent par défaut le précalcul d'agrégats sur tous les niveaux des hiérarchies du schéma [Arb98, App98]. Ceci est fait en cherchant à diminuer les temps de réponse, mais les gros volumes de données générés peuvent provoquer la dégradation des performances [PC98]. Il est évident que les travaux sur la matérialisation de vues relationnelles peuvent être repris dans le contexte de la matérialisation sélective des cubes.

2.5.3 Traitement parallèle de requêtes

Le volume de données d'un entrepôt peut atteindre l'ordre du téraoctet et, dans ce cas, les méthodes traditionnelles de stockage et de traitement de requêtes doivent être parallélisées. Les travaux portant sur l'application de techniques parallèles pour le traitement de requêtes dans les systèmes relationnels sont nombreux [DG92] et, à l'heure actuelle, plusieurs SGBDR commerciaux les utilisent. Dans le cas des travaux récents sur l'interrogation des entrepôts, il s'agit de tirer profit de la structure sous forme d'étoile des schémas relationnels pour répondre aux requêtes de type *jointure en étoile*. Dans ce type de requêtes, on fait la jointure d'une ou plusieurs relations de dimension avec la relation de faits. Par exemple, considérons la requête du type *jointure en étoile* suivante, exprimée sur les relations du schéma en étoile de la figure 2.3 :

```
select    M.ville, SUM(V.quantite)
from      Ventes V, Magasin M, Produit P, Temps T
where     M.magasin = V.magasin AND
          P.produit = V.produit AND
          T.temps = V.temps
group by  M.ville
```

Le résultat de cette requête est la quantité totale de ventes par ville. Étant donné la grande taille de l'entrepôt, l'exécution d'une requête comme celle-ci devient une opération coûteuse. Datta *et al.* [DMT98] proposent un algorithme qui exploite le schéma en étoile en partitionnant les données entre plusieurs processeurs et en exécutant la jointure en parallèle. Aujourd'hui, il est reconnu que l'utilisation effective du parallélisme dans ce cadre ne sera efficace que si des techniques de placement adéquates sont utilisées [SMKK98].

2.6 Analyse de l'entrepôt

L'objectif principal de la création d'un entrepôt est l'analyse des données qu'il contient. Dans cette section, nous présentons les trois techniques d'analyse les plus souvent utilisées : le traitement analytique en ligne, la fouille de données et la visualisation.

2.6.1 Traitement analytique en ligne

Comme il a été dit précédemment, le traitement analytique en ligne vise à effectuer la synthèse et l'analyse des données multidimensionnelles [PC98]. Étant donné la représentation d'un entrepôt sous la forme d'un cube, plusieurs opérations pour sa manipulation, comme `slice-n-dice`, `roll-up` et `drill-down`, ont été proposées par l'industrie [Ken95, App98, Arb98]. L'opérateur `slice-n-dice` permet de restreindre les valeurs dans une ou plusieurs dimensions. La figure 2.11 présente le résultat de l'application de cet opérateur sur le cube *Ventes*, sélectionnant les ventes de téléviseurs et de magnétoscopes réalisées dans les magasins de Lyon ou de Annecy en 1994 et 1996.

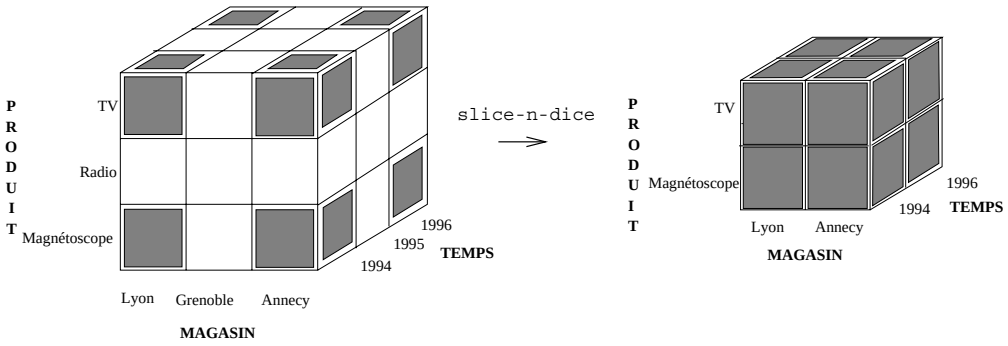


FIG. 2.11 – *Application de* `slice-n-dice` *sur le cube Ventes*

Les opérateurs `roll-up` et `drill-down` autorisent l'analyse de données à différents niveaux d'agrégation en utilisant les hiérarchies associées à chaque dimension. L'opérateur `roll-up` effectue l'agrégation des mesures allant d'un niveau particulier vers un niveau plus général des hiérarchies associées à une dimension. L'opérateur `drill-down` réalise l'opération inverse. La figure 2.12 montre l'application de l'opérateur `roll-up` pour déterminer le total des ventes au niveau de la région, en sommant les totaux

de chaque ville. La figure 2.12 montre également l'application de `drill-down` pour déterminer le détail des ventes par ville.

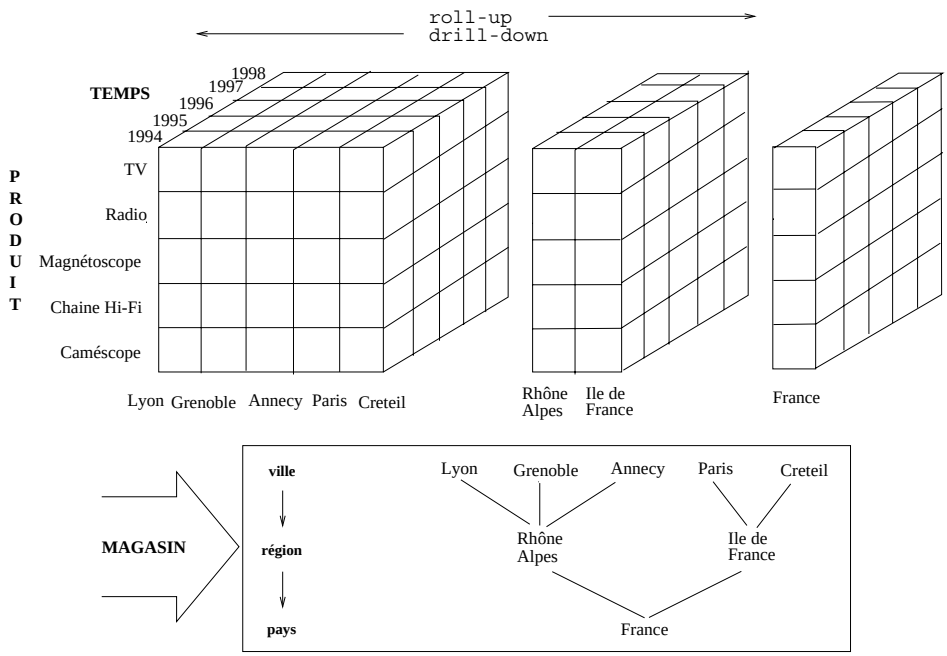


FIG. 2.12 – *Application de* `roll-up` *et* `drill-down` *sur le cube Ventes*

Plusieurs langages OLAP déclaratifs visant à faciliter l'écriture de requêtes complexes ont été proposés (*cf.* Annexe A). La plupart des algèbres [AGS95, LW96, GL97, TDV97] propose des opérateurs unaires comme `slice-n-dice` ou `roll-up` (ce dernier formalisé comme la composition d'opérateurs plus simples de regroupement et d'agrégation [Vas98]). On trouve également la proposition d'opérateurs binaires, semblables aux opérateurs de l'algèbre relationnelle, pour effectuer l'`union`, l'`intersection`, la `différence` et la `jointure` de cubes. Le langage d'interrogation proposé par Cabibbo et Torlone [CT97] se base sur la logique du premier ordre et permet l'incorporation des fonctions d'agrégat mieux adaptées aux besoins du domaine d'application.

Les systèmes de traitement analytique en ligne sont souvent classés en fonction du système utilisé pour le stockage et la gestion de l'entrepôt ; ainsi, il en existe trois types : Relationnel (ROLAP), Multidimensionnel (MOLAP) et Hybride (HOLAP). Les systèmes ROLAP [Mic98, Inf98] héritent des systèmes relationnels les fondements théoriques et la possibilité de stocker des volumes de données de l'ordre du téraoctet. Cependant, leur temps de réponse lors de l'interrogation peut être élevé,

parce qu'il est nécessaire de traduire les résultats relationnels en une représentation multidimensionnelle.

Les systèmes MOLAP [Arb98, Pil98, App98], qui utilisent un SMD, présentent des temps de réponse plus faibles par rapport à ceux d'un système ROLAP, mais les volumes de données stockées sont limités [Col96]. Les systèmes HOLAP [Ora98a, Spe97, Sea98] essaient d'éviter les problèmes des systèmes ROLAP et MOLAP, en stockant les données détaillées dans un système relationnel et les données agrégées d'un entrepôt dans un système multidimensionnel. Ainsi, il est possible de gérer une grande quantité de données et d'avoir un temps de réponse acceptable.

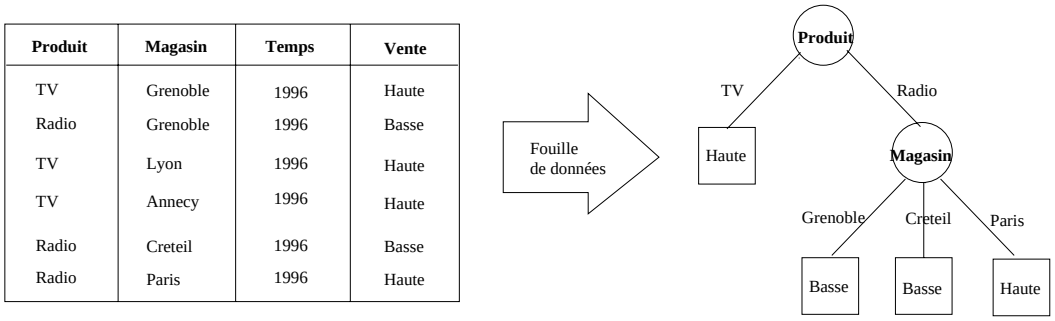
2.6.2 Fouille de données

Lorsque l'utilisateur analyse l'entrepôt en utilisant les techniques de traitement analytique en ligne, il choisit des zones « intéressantes » de l'espace multidimensionnel. Cependant, l'espace à analyser peut être de grande taille et il est évident que l'utilisateur n'est pas capable de tout explorer. Pour cette raison, des mécanismes plus puissants et « intelligents » qui automatisent une partie du processus d'analyse sont nécessaires. À ce propos, les techniques de fouille de données (ou *Data Mining*) sont des outils indispensables.

La fouille de données est la recherche de la connaissance, sous forme de modèles de comportement, cachée dans les données [FPSSU96]. Il s'agit d'un domaine jeune qui résulte du rapprochement des domaines de l'Intelligence Artificielle, des Statistiques et des Bases de Données. Il existe un nombre important de techniques de fouille de données, comme la régression linéaire, l'induction d'arbres de décision, les algorithmes génétiques, les réseaux de neurones, les algorithmes de formation de groupes et la génération de règles d'association [CHY96, HK00].

La figure 2.13 montre l'application d'une technique de fouille à un ensemble de données. Dans ce cas, il s'agit d'induire, à partir des n-uplets en entrée, un modèle de classification sous la forme d'un arbre de décision. L'algorithme appliqué a construit un arbre dont l'attribut le plus important pour distinguer un n-uplet d'un autre est **Produit** et dans un deuxième niveau, **Magasin**. Dans la figure 2.13, il est possible d'observer que, quelle que soit la localisation du magasin et l'année, la vente des TV est élevée. En revanche, la vente de radios dépend de la localisation du magasin.

À l'heure actuelle, la recherche dans ce domaine est très active. Un axe de recherche important est celui de la proposition de techniques et d'algorithmes mieux



Produit	Magasin	Temps	Vente
TV	Grenoble	1996	Haute
Radio	Grenoble	1996	Basse
TV	Lyon	1996	Haute
TV	Annecy	1996	Haute
Radio	Creteil	1996	Basse
Radio	Paris	1996	Haute

FIG. 2.13 – *Application d'une technique de fouille de données*

adaptés au traitement de données multimédia (texte, audio, vidéo) et non-structurées [ZHL+98]. Un autre axe de recherche considère la fouille de données comme un processus d'interrogation de bases de données [IM96] avec une théorie générale qui est en cours de proposition [Man97]. Des extensions à SQL pour exprimer la fouille dans une requête ont également été proposées [MPC96, HFW+96].

2.6.3 Visualisation

La visualisation vise à représenter des données sous forme d'images, de graphiques ou d'animations. Ces représentations visuelles peuvent aider les analystes lors de la recherche des corrélations entre les données, facilitant l'analyse et l'interprétation de celles-ci [KLTW95]. Il existe un nombre important de techniques de visualisation. Les plus simples, comme les camemberts, les graphes à points ou les histogrammes, présentent les données en deux ou trois dimensions. D'autres techniques plus sophistiquées, comme les multidimensionnelles, visent à présenter n dimensions sur un plan à 2 dimensions [BEH+94].

À l'heure actuelle, il est possible d'accéder aux données d'un entrepôt depuis des scènes VRML en trois dimensions. Le langage VRML [VRM97] est un langage standard qui permet de décrire la géométrie et le comportement d'objets éventuellement partagés par plusieurs scènes réparties sur Internet. Les travaux récents sur ce langage proposent d'incruster du code SQL dans la description des scènes pour accéder aux entrepôts relationnels [CJ98].

2.7 Évolution de schéma

Le sujet de l'évolution du schéma des bases de données concerne les problèmes liés à la modification dynamique du schéma d'une base [BKKK87, Rod92]. Les motivations pour permettre l'évolution de schéma des bases de données sont diverses, par exemple autoriser la conception incrémentale ou donner la possibilité de corriger des éventuelles erreurs. L'évolution de schéma est un problème classique des bases de données qui a été principalement étudié pour les bases relationnelles et à objets. Á l'heure actuelle, peu de travaux ont abordé cette problématique dans le contexte des entrepôts de données. Ces travaux visent à adapter les résultats obtenus précédemment aux caractéristiques particulières des entrepôts, telles que la nature multidimensionnelle des données.

La suite de cette section est organisée comme suit. Le paragraphe 2.7.1 présente une synthèse des travaux précurseurs abordant la problématique de l'évolution du schéma des bases de données. Le paragraphe 2.7.2 présente la synthèse des recherches effectuées dans les contexte des entrepôts.

2.7.1 Les précurseurs

Évolution de schémas relationnels et à objets

Un nombre important de contributions abordent le problème de l'évolution de schéma dans les contextes relationnel et à objets.

Dans le cas relationnel, peu de concepts (relations et attributs décrits sur des domaines atomiques) sont utilisés pour décrire le schéma d'une base. Ainsi, le nombre de changements du schéma est limité [MS90] : ajout ou suppression d'un attribut dans un schéma de relation, modification du type d'un attribut dans un schéma de relation, ajout ou suppression d'un schéma de relation, etc. Notons que, à l'heure actuelle, l'ensemble de primitives d'évolution de schéma varie d'un SGBD à un autre SGBD. Certains systèmes (par exemple [Ora02]) offrent la possibilité de supprimer des attributs d'une relation alors que d'autres systèmes ([IBM02]) n'offrent pas cette possibilité.

Dans un contexte à objets le schéma est plus complexe, parce que le modèle est plus riche. Le nombre de changements du schéma autorisé est plus important [BKKK87] : changements dans la définition des classes (ajout, suppression ou modification de variables, de domaines et de méthodes) ou de leur hiérarchie (ajout ou suppression de classes, modification des relations superclasse/sous-classe). Le problème de l'évolution

de schéma dans les bases de données à objets présente une importante activité de recherche et développement. Cela est dû particulièrement à la nature évolutive des applications (génie logiciel, systèmes d'information géographique, etc.) les utilisant.

Les approches pour la gestion de l'évolution de schéma

Faire évoluer le schéma d'une base pose des problèmes à plusieurs niveaux : schéma, instance et applications. Le changement d'une partie du schéma peut provoquer des incohérences avec d'autres parties du schéma. Ceci fait nécessaire la propagation des changements aux parties affectées pour assurer la cohérence du schéma. Au niveau des instances (les données), elles peuvent devenir incohérentes vis-à-vis de leur schéma. Une évolution du schéma doit donc être propagée aux instances du schéma pour assurer le maintien de leur cohérence. Au niveau des applications, les anciennes applications peuvent devenir incompatibles avec le schéma après l'évolution.

Il existe trois approches principales à la gestion de l'évolution de schéma : la modification du schéma, l'utilisation des vues et le versionnement. Dans la suite de cette section nous présentons, pour chacune des approches, son principe, ses avantages et ses inconvénients.

Modification du schéma. Dans cette approche, le schéma de la base de données possède une seule définition, partagée par tous les utilisateurs. Un objet possède donc une seule représentation. Une évolution du schéma est alors traduite par la mise à jour du schéma et l'adaptation des instances associées [BKKK87, PS87, FMZ94, FMZ+95, Hon95].

Pour maintenir la cohérence d'une instance vis-à-vis de son schéma, on utilise la conversion ou l'émulation comme techniques d'adaptation. La conversion [FMZ94] restaure la cohérence d'une instance par la mise à jour physique des représentations des objets. Elle peut introduire une perte d'informations lors de la suppression ou la modification d'éléments du schéma. Cette perte d'informations peut engendrer l'incompatibilité des anciens programmes qui utilisent la base au travers de la partie modifiée du schéma. Ces programmes devront être modifiés.

La conversion peut être immédiate ou différée. Dans la conversion immédiate, la modification du schéma provoque la mise à jour de tous les objets qui ont une représentation non conforme avec la nouvelle structure de leurs classes. Ainsi, la cohérence d'une instance vis-à-vis de son schéma est toujours garantie. Le coût de

la conversion est fonction du nombre d'objets à modifier. Ce coût devient prohibitif dans le cadre d'applications qui manipulent un grand nombre d'objets volumineux et complexes. Dans la conversion différée, la mise à jour n'est effectuée sur un objet que lorsqu'on y accède. Dans ce cas, il est nécessaire de gérer l'historique des mises à jour différées. Ainsi, l'accès aux objets dans les applications où l'évolution du schéma est fréquente, induit une dégradation des performances du système.

Dans l'émulation [BKKK87], les représentations des objets ne sont pas mises à jour mais plutôt interprétées dans un format conforme aux classes au moment de leur utilisation. Cette approche évite les inconvénients de la conversion. Ici, les inconvénients résident dans la complexité du mécanisme d'interprétation et la dégradation des performances, particulièrement quand on accède fréquemment aux objets affectés.

Mécanisme de vues. Certains travaux de recherche (voir par exemple [Ber92, Bra92, TS93, BFK95, RR97, Bel97, BL96, Bel96, LDB97]) suggèrent l'utilisation du mécanisme de vues pour simuler les changements du schéma d'une base. Rappelons qu'une vue est définie par une abstraction sur le schéma réel de la base et que l'instance d'une vue est calculée à partir de celle du schéma réel. L'idée est alors que les applications utilisent la base de données par l'intermédiaire de vues. Une évolution du schéma est ainsi traduite par la création ou la modification d'une vue qui reflète la sémantique de cette évolution.

L'avantage principal de cette approche est d'autoriser les changements de schéma sans avoir à restructurer les instances, parce que la définition d'une vue ne modifie pas le schéma sous-jacent. L'utilisation de vues préserve aussi les programmes d'application des incompatibilités qui peuvent résulter de la perte d'informations. Le mécanisme de vues est cependant insuffisant pour permettre les évolutions dites "additives", comme l'ajout d'un attribut à une classe. Ceci parce que les attributs ajoutés par une vue sont souvent dérivés à partir d'autres attributs qui existent déjà, alors que les informations ajoutées par une évolution de schéma sont en générale nouvelles et indépendantes des informations existantes. Enfin, les performances du SGBD peuvent être dégradées lorsque le nombre de modifications (et donc de vues) est important, parce qu'il doit calculer des vues en utilisant d'autres vues.

Versions. Dans cette approche, le schéma de la base de données peut avoir plusieurs versions. Une évolution de schéma est traduite ainsi par la dérivation d'une nouvelle version [Rod95, Lau96]. La motivation principale pour l'utilisation du versionnement

pour l'évolution du schéma est d'autoriser les applications à accéder aux données de la base à partir de plusieurs versions du schéma. Les anciennes applications restent exécutables après l'évolution du schéma car l'ancienne version du schéma existe toujours. Cela est particulièrement important quand le nombre d'applications affectées est grand ou qu'il est impossible de les modifier.

Les travaux sur le versionnement du schéma ont été réalisés principalement dans un contexte à objets. Certains contributions [SZ90, Mon93, MS93, Cla94] préconisent le versionnement au niveau de la classe. Une évolution apportée à une classe crée une nouvelle version de cette classe et de ses sous-classes ainsi que de toutes les classes qui lui sont liées par agrégation. L'inconvénient de cette approche est que les programmeurs des applications doivent s'assurer qu'ils utilisent une configuration cohérente des versions de classes, compliquant ainsi la tâche de développement. Pour cette raison, d'autres travaux [KC88, Odb92, BM93, Lau96] proposent le versionnement au niveau du schéma. Une évolution de schéma d'une base de données provoque la dérivation d'une nouvelle version du schéma complet de la base. Il est possible ainsi de vérifier la cohérence du schéma par avance et les programmeurs doivent seulement choisir une version particulière. L'inconvénient est qu'un nombre important de schémas peut être généré, notamment lorsque les modifications qui sont effectuées ne concernent qu'une infime partie du schéma.

Pour maintenir la cohérence des données vis-à-vis de leur schéma, le versionnement des objets est utilisé comme technique d'adaptation. Une version d'un objet peut être vue comme un instantané de l'objet pris à un certain moment. Plusieurs versions peuvent coexister, représentant différents états de l'objet. L'évolution d'une classe se traduit par la dérivation d'une nouvelle version de la classe et la dérivation d'une nouvelle version de chacun de ses objets. La perte d'informations est ainsi évitée. Par contre, le nombre de versions d'objets peut être important quand le nombre de versions de la classe augmente. De plus, les coûts de stockage des versions et ceux de la maintenance des relations qui les lient augmentent avec le nombre de versions.

2.7.2 L'évolution de schéma dans le contexte des entrepôts

Un entrepôt de données est construit pour satisfaire les besoins d'information d'un groupe d'utilisateurs. Il est important alors que l'entrepôt puisse évoluer lorsque ces besoins changent. L'administrateur de l'entrepôt doit pouvoir faire évoluer le schéma pour intégrer des nouveaux besoins résultant des processus d'analyse ou pour

prendre en compte l'évolution du schéma des sources. Dans la suite, nous présentons les travaux qui ont été consacrés à l'évolution du schéma des entrepôts de données.

Travaux existants

Il existe à l'heure actuelle peu de contributions qui abordent la problématique de l'évolution du schéma des entrepôts des données. Nous citons les travaux de Hurtado et al. [HMV99a, HMV99b] à l'Université de Toronto au Canada et de Blaschka et al. [Bla99, BSH99, HSB00] au sein du projet BabelFish du laboratoire FORWISS en Allemagne.

Hurtado et al. [HMV99a, HMV99b] abordent le problème de l'évolution des relations de dimension des schémas en étoile et en flocon de neige. Ils introduisent un modèle abstrait pour les dimensions et un ensemble d'opérateurs d'évolution associés à ce modèle. Dans le modèle qu'ils proposent, une dimension peut être formée par plusieurs niveaux formant des hiérarchies d'agrégation alternatives. Les opérateurs d'évolution autorisent l'ajout et la suppression de niveaux. Les opérateurs sont implantés de manière à réaliser la modification du schéma relationnel. L'étude de l'évolution de cubes n'est pas considérée dans ce travail.

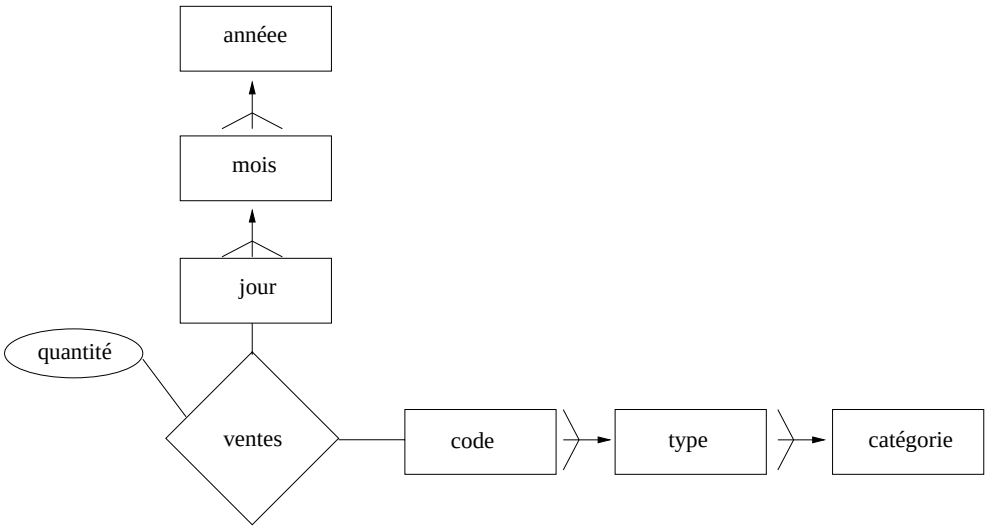


FIG. 2.14 – *Schéma ME/R*

Les travaux de Blaschka et al. [Bla99, BSH99, HSB00] se situent dans le contexte du projet BabelFish. Ce projet propose un outil d'aide à la conception et à l'évolution du schéma des entrepôts de données. L'outil offre à l'administrateur la possibi-

lité de créer et de faire évoluer le schéma d'un entrepôt à partir d'une représentation conceptuel graphique, nommée ME/R (*Multidimensional Entity-Relationship Model*), qui adapte le modèle entité-association pour prendre en compte les notions de dimension et de faits (mesures). La figure 2.14 montre le schéma ME/R pour notre exemple de la chaîne de magasins. Les modifications au schéma sont alors réalisés au niveau conceptuel et l'outil propage ensuite ces modifications au niveau (relationnel ou multidimensionnel) d'implantation.

Notons que les travaux décrits ci-dessus considèrent que le schéma de l'entrepôt est conçu pour satisfaire des besoins d'information des utilisateurs et que ses données sont récupérées à partir des sources qui peuvent contribuer à satisfaire ces besoins. La principale motivation de ces travaux est d'autoriser l'évolution du schéma pour intégrer des nouveaux besoins des utilisateurs. Nous avons adopté cette optique pour la réalisation de notre travail.

Remarquons cependant que d'autres contributions considèrent qu'un entrepôt est construit pour permettre l'accès intégré à un ensemble spécifique de sources. Dans ce cas, le schéma de l'entrepôt est souvent dérivé à partir des schémas des sources d'information. Le problème est alors de faire évoluer le schéma de l'entrepôt suite à l'évolution des schémas des sources. Le projet EVE (*Evolvable View Environment*) à l'Institut Polytechnique Worcester [NLR98, RKZ+99, RKZ00] aborde ce problème. Dans la solution proposée, l'administrateur indique les informations qui peuvent être remplacées par d'autres informations similaires pouvant être récupérées d'une autre source. Lorsqu'un changement de schéma affectant le schéma de l'entrepôt est détecté, il est possible de le redéfinir en trouvant les remplacements appropriés. L'intervention de l'administrateur est ainsi minimisée.

Effets de l'évolution du schéma des entrepôts

Nous avons précédemment vu que faire évoluer le schéma d'une base de données peut provoquer des incompatibilités avec les données et les applications qui dépendent de lui. Ceci reste particulièrement vrai dans le contexte des entrepôts de données, où une évolution peut avoir des effets sur les agrégats matérialisés, les index, les applications d'analyse et l'application dédiée à la construction de l'entrepôt. Dans la suite nous discutons chacun de ces aspects.

Agrégats matérialisés. La définition d'un agrégat matérialisé doit évoluer en fonction des modifications survenant au niveau du schéma, afin de préserver la cohérence structurelle de l'entrepôt. Ce problème est abordé dans [Bel98], où Bellahsène propose une solution pour le cas des vues relationnelles. L'objectif de ce travail est de gérer les modifications du schéma des vues matérialisées sans provoquer des incompatibilités pour les applications existantes. Pour cela, il propose un mécanisme de vues pour simuler les changements au schéma d'une vue matérialisée. L'idée générale de l'approche est la suivante : si un changement met en péril la compatibilité des applications existantes, alors une vue simulant le changement est générée; dans le cas contraire, le changement est réalisé, modifiant physiquement la vue matérialisée.

Index. L'évolution du schéma entraîne la création de nouveaux index ou la suppression des index obsolètes. Par exemple, l'ajout d'un attribut a dans une relation peut conduire à la création d'un index sur cet attribut. Ceci parce qu'il est attendu que a sera utilisé lors des interrogations futures. De la même manière, la suppression d'un attribut peut conduire à la suppression des index définis sur celui-ci, afin d'assurer la cohérence de l'entrepôt.

Applications d'analyse. Les changements du schéma peuvent provoquer des incompatibilités avec les applications d'analyse existantes. La suppression d'une partie du schéma, par exemple, peut entraîner l'élimination de données nécessaires pour le fonctionnement correct d'une application.

Application de collecte et d'intégration de données. L'évolution du schéma peut provoquer également des incompatibilités avec l'application dédiée à la collecte et à l'intégration des données stockées dans l'entrepôt. En effet, elle peut devenir incomplète ou incohérente vis-à-vis du nouveau schéma de l'entrepôt : incomplète, parce qu'elle doit calculer des instances qui n'étaient pas calculées avant de modifier le schéma (quand on ajoute des nouvelles données dans l'entrepôt), ou incohérente, parce que les instances qu'elle calcule ne correspondent pas au nouveau schéma (quand on modifie une partie du schéma). Dans les deux cas, de nouvelles sources peuvent être impliquées ou d'anciennes sources ignorées.

Remarquons finalement qu'il est possible d'introduire des erreurs lors de l'évolution de l'entrepôt, mettant ainsi en péril sa qualité. Le projet DWQ [Qui99] propose

l'utilisation de métadonnées pour tracer l'histoire des changements. De cette manière, lorsqu'un problème arrive, ces métadonnées peuvent être utilisées pour trouver sa cause.

2.8 Conclusion

Dans ce chapitre, nous avons étudié les entrepôts de données selon le point de vue de la recherche en bases de données. Nous avons constaté que les systèmes multidimensionnels ou relationnels utilisés pour gérer les entrepôts ont une grande influence sur l'organisation logique et physique de données, ainsi que sur les aspects liés à l'interrogation. Le modèle multidimensionnel de données, qui organise les données en cubes, est aujourd'hui à la base de plusieurs systèmes dans lesquels les notions de schéma et d'instance ne sont pas bien définies. À l'heure actuelle, il n'existe pas de consensus pour un modèle multidimensionnel, même si plusieurs propositions ont été faites.

Les organisations logique et physique de données autorisent l'interrogation efficace de gros volumes de données. À la différence des schémas relationnels fortement normalisés des bases de données traditionnelles, les schémas relationnels pour les entrepôts sont dénormalisés, ce qui diminue le nombre de jointures nécessaires pour accéder aux données. En ce qui concerne le niveau physique, nous avons vu que le stockage d'un cube dans un système multidimensionnel est réduit au problème du stockage d'un vecteur. Par rapport aux méthodes d'accès indexées, des techniques comme les index binaires sont à nouveau utilisées.

Nous avons présenté une architecture de base pour la collecte et l'intégration de données dans les entrepôts et pour leur utilisation. Associé à chaque source, on trouve un adaptateur/moniteur, qui détecte les changements dans la source et transforme ces données en une représentation intermédiaire homogène. Cette uniformité de représentation est exploitée par l'intégrateur, qui fusionne les données en provenance des adaptateurs/moniteurs en éliminant les éventuels problèmes d'hétérogénéité sémantique et structurelle. Les données intégrées sont finalement chargées dans l'entrepôt.

En ce qui concerne l'interrogation des entrepôts, le précalcul d'agrégats et l'utilisation du parallélisme permettent d'améliorer les temps de réponse des SGBD relationnels ou multidimensionnels. Dans un SGBDR, les vues matérialisées, déjà étudiées depuis plusieurs années dans d'autres contextes, sont maintenant utilisées pour des

agrégats. Dans le cas des systèmes multidimensionnels, les mécanismes pour la matérialisation sélective de cubes sont encore en phase d'exploration.

Par rapport à l'analyse des données de l'entrepôt, nous avons étudié les techniques les plus couramment utilisées. Le traitement analytique en ligne, qui vise à l'agrégation de données, a été bien accueilli par l'industrie et fait maintenant l'objet de nombreux travaux de recherche. La fouille de données, qui vise à l'extraction de connaissances sous la forme de modèles de comportement des données, constitue aujourd'hui une activité importante au niveau de la recherche. Les techniques de visualisation, qui produisent des représentations graphiques des données, sont également utiles lors de l'analyse de l'entrepôt.

L'évolution du schéma des entrepôts de données est un sujet important qui a reçu peu d'attention. Les solutions proposées jusqu'à maintenant se focalisent sur l'évolution des schémas de dimension et l'adaptation des instances associées. Des contributions plus récentes ont étendu ces résultats pour prendre en compte l'évolution du schéma des cubes. Ces travaux visent à garder la sémantique des dimensions et de cubes lors des évolutions.

Chapitre 3

Modèle et langage d'évolution

Nous présentons dans ce chapitre notre modèle multidimensionnel de données. Une de caractéristiques principales de ce modèle est la claire séparation entre le schéma et l'instance d'une dimension ou d'un cube. Une autre caractéristique importante est la complexité des dimensions et des cubes. Dans notre modèle, les dimensions peuvent avoir de multiples hiérarchies alternatives et les cubes peuvent avoir une ou plusieurs mesures.

Nous présentons également un ensemble de primitives pour faire évoluer des schémas multidimensionnels. Cet ensemble comprend la création de nouveaux schémas de dimension et schémas de cube, la suppression d'anciens schémas et la modification de schémas existants. Nous avons défini la sémantique de nos opérateurs de manière à assurer la cohérence du schéma entre des évolutions successives.

Nous introduisons aussi MDL (*Multidimensional Data Definition Language*), un langage de description de données multidimensionnelles associé au modèle. Il autorise la définition de schémas multidimensionnels. Il autorise également l'évolution d'un schéma multidimensionnel à travers l'ajout, la suppression et la modification de schémas de dimension et de schémas de cube.

Le plan de ce chapitre est le suivant. D'abord, nous présentons notre modèle multidimensionnel de données (cf. section 3.1). Puis nous introduisons nos primitives d'évolution de schémas multidimensionnels (cf. section 3.2). Ensuite, nous présentons MDL (cf. section 3.3). Enfin, nous concluons en discutant les ressemblances et les différences de notre proposition vis-à-vis d'autres travaux existants (cf. section 3.4).

3.1 Le modèle de données MD

Nous supposons dans la suite l'existence des ensembles suivants :

- un ensemble de valeurs $\mathcal{DOM} =$ dom(char) $\cup$ dom(integer) $\cup$ dom(float) $\cup$ dom(decimal) $\cup$ dom(date) $\cup \{t\}$ contenant le domaine de chaque type atomique plus la valeur distinguée t.
- un ensemble infini $\mathcal{N}$ de noms
- un ensemble $\mathcal{L} \subseteq \mathcal{N}$ de noms de niveaux
- un ensemble $\mathcal{P} \subseteq \mathcal{N}$ de noms de propriétés
- un ensemble $\mathcal{D} \subseteq \mathcal{N}$ de noms de dimension
- un ensemble $\mathcal{C} \subseteq \mathcal{N}$ de noms de cube
- un ensemble $\mathcal{M} \subseteq \mathcal{N}$ de noms de mesure.

Associé à un niveau $l \in \mathcal{L}$ on trouve un ensemble de valeurs $\{e_1, \ldots, e_n\}$ tel que $\forall_{i=1,\ldots,n} e_i \in \mathcal{DOM}$. Les éléments de cet ensemble sont nommés *membres*. Nous distinguons un niveau nommé $\top$ qui a comme seul membre la valeur t. De manière similaire, associé à une propriété $p \in \mathcal{P}$ on trouve un ensemble de valeurs $\{u_1, \ldots, u_n\}$ tel que $\forall_{i=1,\ldots,n} u_i \in \mathcal{DOM}$. Enfin, associé à une mesure $m \in \mathcal{M}$ on trouve également un ensemble de valeurs $\{v_1, \ldots, v_n\}, \forall_{i=1,\ldots,n} v_i \in \mathcal{DOM}$.

Nous supposons aussi l'existence des fonctions suivantes :

- Une fonction $mset$ de signature $mset : \mathcal{L} \rightarrow \mathcal{E}(\mathcal{DOM})$[1], qui retourne l'ensemble des membres associés à un niveau.
- Une fonction $pdom$ de signature $pdom : \mathcal{P} \rightarrow \mathcal{E}(\mathcal{DOM})$, qui retourne l'ensemble des valeurs associées à une propriété.
- Une fonction $mdom$ de signature $mdom : \mathcal{M} \rightarrow \mathcal{E}(\mathcal{DOM})$, qui retourne l'ensemble des valeurs associées à une mesure.

1. L'ensemble $\mathcal{E}(\mathcal{A})$ est appelé l'ensemble de parties de $\mathcal{A}$ et il est défini comme $\mathcal{E} = \{X | X \subset \mathcal{A}\}$

Définissons d'abord les notions *schéma de dimension* et *instance de dimension*. Un schéma de dimension est un graphe acyclique dirigé (GAD) représentant les hiérarchies des niveaux. Chaque noeud du graphe contient un niveau et un arc entre deux noeuds représente une association entre les niveaux contenus par les noeuds. Il existe un seul noeud, nommé *base*, à partir duquel tous les autres noeuds peuvent être atteints directement ou par transitivité. Il existe aussi un noeud, nommé *sommet*, qui contient le niveau $\top$ et qui est atteint (directement ou par transitivité) par tous les autres noeuds. Il existe, associé à chaque niveau, un ensemble éventuellement vide de noms de propriété.

Définition 1 (Schéma de dimension)
Un schéma de dimension est un n-uplet $(d_n, L, P, \prec, \rho)$*, où* $d_n \in \mathcal{D}$ *est le nom de la dimension,* $L \subseteq \mathcal{L}$ *est un ensemble fini de niveaux contenant* $\top$*,* $P \subseteq \mathcal{P}$ *est un ensemble fini de propriétés,* $\prec$ *est une relation définie sur* L*, et* ρ *est une fonction de* P *à* L*. La relation* $\prec$ *est telle que sa fermeture transitive et reflexive* $\preceq^*$ *est un ordre partiel dans* L*. Il existe un seul niveau* $l_\perp$ *tel que* $\forall_{l \in L} l_\perp \preceq^* l$*. En plus,* $\forall_{l \in L} l \preceq^* \top$*. On dit qu'un niveau* $l_j \in L$ *est un niveau immédiat supérieur (ou père) de* $l_i \in L$ *si* $l_i \prec l_j$*.* □

Une *instance de dimension* organise les membres des niveaux participant aux hiérarchies d'agrégation. Pour chaque arc dans un schéma de dimension, il existe un fonction de *rollup* associée. Une caractéristique importante de cette famille de fonctions est que tout membre d'un niveau est mis en relation avec au moins un membre des niveaux immédiatement supérieurs.

Définition 2 (Instance de dimension)
Étant donné un schéma de dimension $(d_n, L, P, \prec, \rho)$*, où* $d_n \in \mathcal{D}$*,* $L \subseteq \mathcal{L}$*,* $P \subseteq \mathcal{P}$*,* $\prec$ *est une relation définie sur* L *et* ρ *est une fonction de* P *à* L*, une* instance de dimension *est un n-uplet* $(\bigcup_{l_i \in L} mset(l_i), RUP, DESC)$*, où* RUP *est un ensemble de fonctions* $rup_{l_i}^{l_j}$ *tel que (i) pour chaque paire de niveaux* $l_1, l_2 \in L$ *tels que* $l_1 \prec l_2$*, il existe une fonction* $rup_{l_1}^{l_2} : mset(l_1) \rightarrow mset(l_2)$ *et (ii) pour chaque triplet de niveaux* $l_1, l_2, l_3 \in L$ *tels que* $l_1 \prec l_2$ *et* $l_2 \prec l_3, ran(rup_{l_1}^{l_2}) \subseteq dom(rup_{l_2}^{l_3})$*. Enfin,* $DESC$ *est un ensemble de fonctions tel que pour chaque* $(p, l) \in \rho$*, il existe une fonction* $desc_p^l : mset(l) \rightarrow pdom(p)$*.* □

Considérons à titre d'illustration la dimension *Produit*. Le schéma de cette dimension (cf. partie gauche de la figure 3.1) est défini de la manière suivante : le nom de

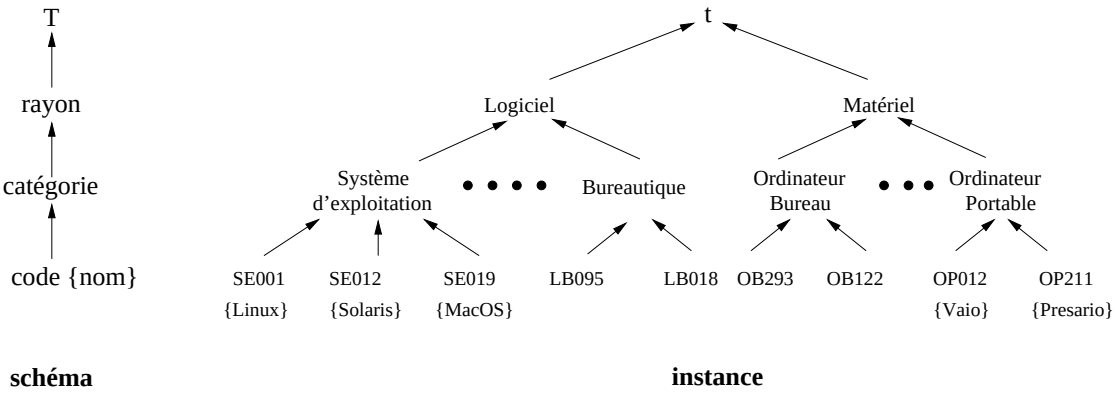


FIG. 3.1 – *Le schéma et une instance possible de la dimension Produit*

la dimension d_n est *Produit*, l'ensemble de niveaux L est $\{code, catégorie, rayon, \top\}$, l'ensemble de propriétés P est $\{nom\}$, la relation $\prec$ est $\{ (code, catégorie), (catégorie, rayon), (rayon, \top) \}$, et la fonction ρ est $\{(nom, code)\}$. La partie droite de la figure 3.1 montre une instance possible de ce schéma. Nous pouvons voir que la fonction $rup_{code}^{categorie}$ met en relation les membres *SE001*, *SE012* et *SE019* du niveau *code* avec le membre *Système d'exploitation* du niveau *catégorie*. Notons que les membres du niveau *rayon* sont mis en relation avec le membre unique t du niveau $\top$. Enfin, nous pouvons voir que le nom *Linux* est associé au membre *SE001* du niveau *code*.

Définissons maintenant les notions *schéma de cube* et *instance de cube*. Un schéma de cube est un n-uplet composé d'un nom, d'un ensemble d'axes et d'un ensemble de noms de mesure. Un *axe* est un niveau d'une dimension. Nous supposons l'existence d'une fonction *dimension* de signature $dimension : \mathcal{L} \rightarrow \mathcal{D}$. Cette fonction prend en entrée un niveau l et donne comme résultat le nom d_n de la dimension dont le niveau fait partie.

Définition 3 (Schéma de cube)
Un schéma de cube *est un triplet* (c_n, A, M)*, où* $c_n \in \mathcal{C}$ *est le nom du cube,* $A \subseteq \mathcal{L}$ *est un ensemble de niveaux, et* $M \subseteq \mathcal{M}$ *est un ensemble de noms de mesure.* $\forall a_i \in A, \exists d_n \in \mathcal{D}$ *telle que* $d_n = dimension(a_i)$*.* □

Une instance de cube est un ensemble de *cellules*. Une cellule est un n-uplet qui associe un ensemble de membres (un membre pour chacun des axes définis dans le schéma du cube) à un ensemble de valeurs de mesure.

Définition 4 (Instance de cube)
Étant donné un schéma de cube $(c_n, \{a_1, \ldots, a_s\}, \{m_1, \ldots, m_t\})$*, où* $c_n \in \mathcal{C}$*,* $a_i \in \mathcal{L}$ *et* $m_i \in \mathcal{M}$*, une* instance de cube *est défini comme étant un ensemble de n-uplets de la forme* $(e_1, \ldots, e_s, v_1, \ldots, v_t)$*, où* $\forall_{k=1,\ldots,s} e_k \in mset(a_k)$ *et* $\forall_{j=1,\ldots,t} v_j \in mdom(m_j)$ □

Considérons par exemple le cube *Ventes*. La partie gauche de la figure 3.2 montre le schéma de ce cube. Il est défini de la manière suivante : le nom c_n du cube est *Ventes*, les éléments de l'ensemble {*code*, *ville*, *jour*} sont les axes du cube et la mesure est *quantité*. La partie droite de la figure 3.2 montre une instance possible de ce cube.

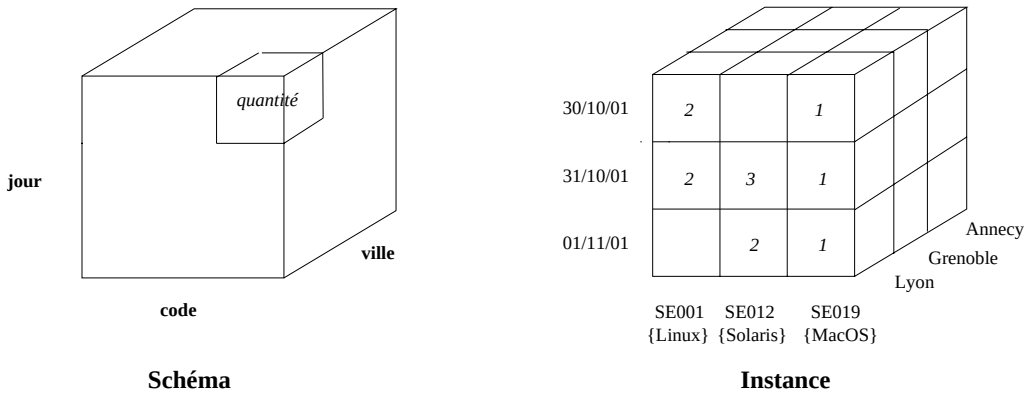


FIG. 3.2 – *Le schéma et une instance possible du cube Ventes*

Nous pouvons à présent définir la notion de *base de données multidimensionnelle*. Une base de ce type est composée d'un ensemble de dimensions et d'un ensemble de cubes.

Définition 5 (Base de données multidimensionnelle)
Le schéma d'une base de données multidimensionnelle est un n-uplet $\mathcal{S} = (D_s, C_s)$*, où* D_s *est un ensemble de schémas de dimension et* C_s *est un ensemble de schémas de cubes. L'instance d'une base multidimensionnelle est un n-uplet* $\mathcal{I} = (D_i, C_i)$*, où* D_i *est un ensemble d'instances de dimension et* C_i *est un ensemble d'instances de cubes. Pour chacune des instances de dimension* $d_i \in D_i$*, il existe un schéma* $d_s \in D_s$ *tel que* d_i *est conforme à* d_s*. De même, pour chacune des instances de cube* $c_i \in C_i$*, il existe un schéma* $c_s \in C_s$ *tel que* c_i *est conforme à* c_s*.* □

3.2 Opérateurs d'évolution

Nous dédions cette section à la présentation de nos 16 opérateurs d'évolution de schémas multidimensionnels. Chacun de ces opérateurs transforme une base multidimensionnelle B en une base B'. Pour définir leur sémantique, nous avons identifié un ensemble de propriétés (nommées *invariants*) d'un schéma multidimensionnel lesquelles doivent être préservées lors des évolutions :

- **Dimension valide.** Un schéma de dimension est un graphe acyclique dirigé. Le graphe a un seul nœud base $l_\perp$ et un seul nœud sommet $\top$.
- **Cube valide.** Dans un schéma de cube, la cardinalité de l'ensemble d'axes est supérieur ou égale à 1. De même, la cardinalié de l'ensemble de mesures est supérieur ou égale à 1.
- **Noms uniques.** Les dimensions et les cubes ont des noms uniques. De même, tous les niveaux et toutes les mesures ont des noms différents.

Nous avons défini la sémantique de nos opérateurs de façon à préserver ces invariants. Nous présentons dans la suite les 8 opérateurs permettant de faire évoluer des schémas de dimension (cf. paragraphe 3.2.1) et les 8 opérateurs permettant de faire évoluer des schémas de cube (cf. paragraphe 3.2.2).

3.2.1 Évolution de schémas de dimension

L'opérateur *Create Dimension* crée un nouveau schéma de dimension. Cet opérateur prend en entrée le nom de la dimension, un ensemble de niveaux, un ensemble (éventuellement vide) de propriétés, une relation entre les niveaux et une fonction associant les propriétés aux niveaux.

Opérateur 1 (Create Dimension)
Étant donnés une base multidimensionnelle B de schéma $\mathcal{S} = (D_s, C_s)$ et d'instance $\mathcal{I} = (D_i, C_i)$, un nom $d_n \in \mathcal{D}$, un ensemble de niveaux $L \subseteq \mathcal{L}$, un ensemble de propriétés $P \subseteq \mathcal{P}$, une relation $\prec$ définie sur L et une fonction ρ de P à L, le résultat de $CreateDim(B, d_n, L, P, \prec, \rho)$ est une base B' dont le schéma est $\mathcal{S}' = (D_s \cup \{d_s\}, C_s)$, où d_s est un schéma de dimension $(d_n, L, P, \prec, \rho)$, et l'instance est $\mathcal{I}' = \mathcal{I}$. □

Considérons par exemple la création du schéma de la dimension *Produit*. Dans ce cas, l'ensemble de niveaux L est {*code*, *catégorie*, *rayon*,$\top$} et l'ensemble P de propriétés est $\{nom\}$. La relation $\prec$ entre les niveaux est l'ensemble de paires {(*code*, *catégorie*), (*catégorie*, *rayon*), (*rayon*, $\top$)}. Enfin, la fonction ρ est {(*nom*, *code*)}. L'opération $CreateDim(B, Produit, L, P, \prec, \rho)$ donne le résultat souhaité.

L'opérateur *Drop Dimension* élimine un schéma de dimension d'une base multidimensionnelle B. Afin d'assurer la cohérence de B vis-à-vis du schéma modifié, l'instance de la dimension est aussi supprimée. Un schéma de dimension ne peut pas être éliminé s'il existe un schéma de cube utilisant un niveau de la dimension en tant qu'axe. Nous supposons l'existence d'une fonction β qui, étant donné un nom de dimension, retourne une valeur booléenne indiquant si le schéma de la dimension peut être supprimé (vrai) ou non (faux).

Opérateur 2 (Drop Dimension)
Étant donnés une base multidimensionnelle B de schéma $\mathcal{S} = (D_s, C_s)$ et d'instance $\mathcal{I} = (D_i, C_i)$, et un nom de dimension $d_n \in \mathcal{D}$ tel que $\beta(d_n) = vrai$, $DropDim(B, d_n)$ est une base B' dont le schéma est $\mathcal{S}' = (D'_s, C_s)$, où $D'_s = D_s \backslash \{(d_n, L, P, \prec, \rho)\}$. L'instance de B' est $\mathcal{I}' = (D'_i, C_i)$, où $D'_i = D_i \backslash \{(\bigcup_{l_i \in L} mset(l_i), RUP, DESC)\}$. □

Supposons que nous voulons supprimer la dimension *Produit*. Cette opération se définit comme $DropDim(B, Produit)$. Si la condition de suppression est vérifiée, alors le schéma de la dimension est retiré du schéma de la base. L'instance de la dimension est également ôtée de la base.

Examinons maintenant les opérateurs permettant de modifier les schémas de dimension. L'opérateur *Rename Dimension* remplace l'ancien nom d_n d'une dimension par un nouveau nom d'_n. Cet opérateur ne modifie pas l'instance de la dimension.

Opérateur 3 (Rename Dimension)
Étant donné une base multidimensionnelle B de schéma $\mathcal{S} = (D_s, C_s)$ et d'instance $\mathcal{I} = (D_i, C_i)$, et deux noms de dimension $d_n, d'_n \in \mathcal{D}$, $RenameDim(B, d_n, d'_n)$ est une base B' dont le schéma est $\mathcal{S}' = (D'_s, C_s)$, où $D'_s = D_s \backslash \{(d_n, L, P, \prec, \rho)\} \cup \{(d'_n, L, P, \prec\ , \rho)\}$, et l'instance est $\mathcal{I}' = \mathcal{I}$. □

Considérons par exemple que nous voulons changer le nom de la dimension *Produit* par *Product*. L'opération $RenameDim(B, Produit, Product)$ modifie le schéma de la dimension en remplaçant l'ancien nom par le nouveau nom.

L'opérateur *Rename Level* change le nom d'un niveau. Cet opérateur prend en entrée le nom d_n de la dimension dont le niveau à renommer fait partie, l'ancien nom du niveau l et le nouveau nom l'.

Opérateur 4 (Rename Level)
Étant donnés une base multidimensionnelle B de schéma $\mathcal{S} = (D_s, C_s)$ et d'instance $\mathcal{I} = (D_i, C_i)$, et deux noms de niveau $l, l' \in \mathcal{L}$, le résultat de $RenameLevel(B, d_n, l, l')$ est une base B' dont le schéma est $\mathcal{S}' = (D'_s, C_s)$, où $D'_s = D_s \backslash \{(d_n, L, P, \prec, \rho)\} \cup \{(d_n, L \backslash \{l\} \cup \{l'\}, P, \prec, \rho)\}$, et l'instance est $\mathcal{I}' = \mathcal{I}$. □

Considérons à titre d'illustration le changement de nom du niveau *code* à *codeProduit*. Dans ce cas, l'opération $RenameLevel(B, Produit, code, codeProduit)$ modifie le schéma de la dimension *Produit* pour prendre en compte cette modification. L'instance n'est pas modifiée.

L'opérateur *Add Level* modifie le schéma de dimension d_s en établissant une relation entre un niveau l_1 faisant partie de d_s et un nouveau niveau l_2. Cette opérateur introduit l'ordre entre l_2 et $\top$ pour assurer que le schéma de dimension résultant soit conforme à la définition précédemment présentée. L'instance de la dimension est adaptée au nouveau schéma. Nous supposons l'existence d'une fonction γ qui, étant donné un nom de dimension n et deux niveau l_i et l_j, retourne une valeur booléenne b telle que (i) $b = vrai$, si $l_i \prec l_j$ dans le schéma de la dimension de nom n (ii) $b = faux$, dans le cas contraire.

Opérateur 5 (Add Level)
Étant donnés une base multidimensionnelle B de schéma $\mathcal{S} = (D_s, C_s)$ et d'instance $\mathcal{I} = (D_i, C_i)$, un nom de dimension $d_n \in \mathcal{D}$, deux niveaux $l_1, l_2 \in \mathcal{L}$ tels que $\gamma(d_n, l_1, l_2) = vrai$, et un membre $x \in mset(l_2)$, $AddLevel(B, d_n, l_1, l_2, x)$ est une base B' dont le schéma est $\mathcal{S}' = (D'_s, C_s)$ tel que $D'_s = D_s \backslash \{d_s\} \cup \{d'_s\}$, où $d_s = (d_n, L, P, \prec, \rho)$ et $d'_s = (d_n, L \cup \{l_2\}, P, \prec \cup \{(l_1, l_2), (l_2, \top)\}, \rho)$. L'instance de B' est $\mathcal{I}' = (D'_i, C_i)$ telle que $D'_i = D_i \backslash \{d_i\} \cup \{d'_i\}$ où $d_i = (\bigcup_{l_i \in L} mset(l_i), RUP, DESC)$ et $d'_i = (\bigcup_{l_i \in L \cup \{l_2\}} mset(l_i), RUP', DESC)$. L'ensemble RUP' contient les fonctions $rup_{l_i}^{\prime l_j}$ telles que $rup_{l_1}^{\prime l_2} = \{(e, x) | \forall e \in mset(l_1)\}$, $rup_{l_2}^{\prime \top} = \{(x, t)\}$ et $rup_{l_i}^{\prime l_j} = rup_{l_i}^{l_j}$ pour les autres niveaux l_i et l_j. □

Considérons la dimension *Produit* montrée par la figure 3.1. Supposons que nous voulons ajouter le niveau *marque*. L'opérateur $AddLevel(B, Produit, code, marque, 'm1')$ modifie le schéma de la dimension en associant le niveau *code* au niveau *marque*

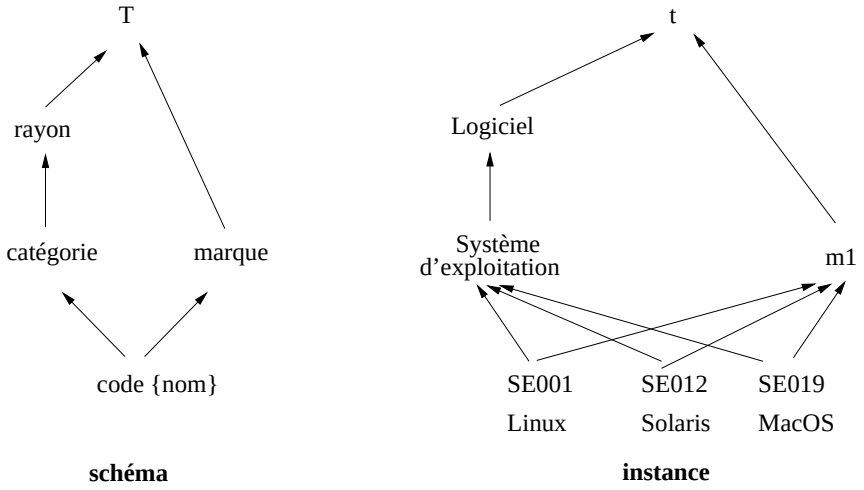


FIG. 3.3 – *La dimension Produit après avoir ajouté le niveau marque*

(cf. partie gauche de la figure 3.3). L'instance de cette dimension est également modifiée (cf. partie droite de la figure 3.3).

L'opérateur *Delete Level* élimine un niveau l d'une dimension. Plus précisément, il supprime le niveau du schéma et de l'instance de la dimension. Les propriétés associées au niveau sont également supprimées. Les niveaux $\top$ et $l_\perp$ du schéma de la dimension ne peuvent pas être supprimés.

Opérateur 6 (Delete Level)
Étant donnés une base multidimensionnelle B de schéma $\mathcal{S} = (D_s, C_s)$ et d'instance $\mathcal{I} = (D_i, C_i)$, un nom de dimension $d_n \in \mathcal{D}$ et un niveau $l \in \mathcal{L}$, tel que $l \neq \top$ et $l \neq l_\perp$, $DelLevel(B, d_n, l)$ est une base B' dont le schéma est $\mathcal{S}' = (D'_s, C_s)$, où $D'_s = D_s \backslash \{(d_n, L, P, \prec, \rho)\} \cup \{(d_n, L', P', \prec', \rho')\}$, où $L' = L \backslash \{l\}$, $P' = P \backslash \{p_i | (p_i, l) \in \rho\}$, $\prec' = \prec \backslash \{(l_1, l_2) | l = l_1 \vee l = l_2)\}) \cup \{(l_1, l_2) | l_1 \prec l \wedge l \prec l_2\}$ et $\rho' = \rho \backslash \{(p_i, l_1) | l = l_1\}$. L'instance de B' est $\mathcal{I}' = (D_i \backslash \{d_i\} \cup \{d'_i\}, C_i)$, où $d_i = (\bigcup_{l_i \in L} mset(l_i), RUP, DESC)$ et $d'_i = (\bigcup_{l_i \in L'} mset(l_i), RUP', DESC')$. L'ensemble RUP' contient les fonctions $rup_{l_i}^{\prime l_j}$ telles que (i) $rup_{l_1}^{\prime l_2} = rup_{l_1}^{l} \circ rup_{l}^{l_2}$, si $l_1 \prec l \wedge l \prec l_2$, (ii) $rup_{l_1}^{\prime l_2} = rup_{l_1}^{l_2}$, dans les autre cas. L'ensemble DESC' contient les fonctions $desc_p^{\prime l}$ telles que $desc_{p_i}^{\prime l_j} = desc_{p_i}^{l_j}$, si $l_j \neq l$. □

Imaginons que nous voulons supprimer le niveau *catégorie* de la dimension *Produit* que nous venons de modifier (cf. figure 3.3). Le résultat de l'application de l'opération $DelLevel(B, Produit, categorie)$ est le schéma de dimension montré dans la partie

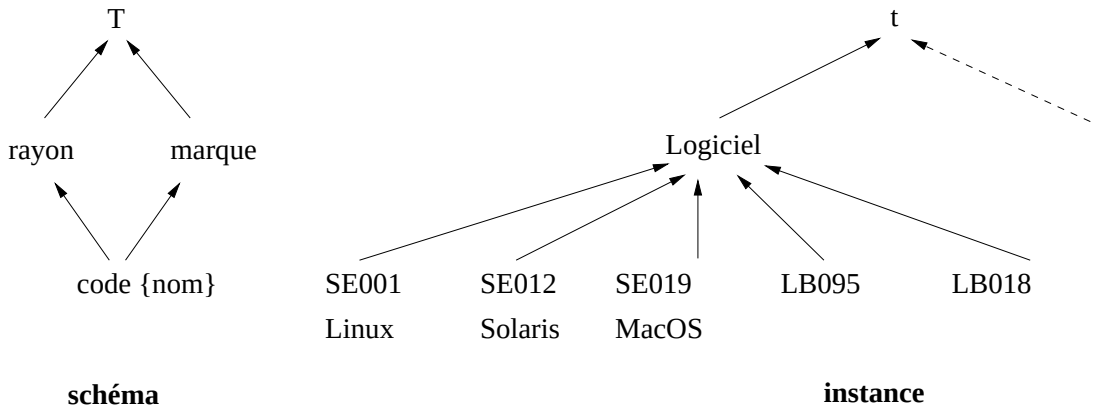


FIG. 3.4 – *La dimension Produit après avoir supprimé le niveau catégorie*

gauche de la figure 3.4. Le niveau *catégorie* a été supprimé et les relations entre les niveaux ont été adaptées : les relations (*code*,*catégorie*) et (*catégorie*,*rayon*) ont été supprimées alors que la relation (*code*,*rayon*) a été ajoutée. Quant à l'instance, elle a subi aussi des adaptations pour être en conformité avec le nouveau schéma. Les membres du niveau *code* sont maintenant mis en relation avec les membres du niveau *rayon*.

L'opérateur *Add Property* associe une nouvelle propriété à un niveau d'une dimension. Cet opérateur prend en entrée, outre la nouvelle propriété p, le nom d_n de la dimension à modifier et le niveau l auquel p sera associée.

Opérateur 7 (Add Property)
Étant donnés une base multidimensionnelle B de schéma $\mathcal{S} = (D_s, C_s)$ et d'instance $\mathcal{I} = (D_i, C_i)$, un nom de dimension $d_n \in \mathcal{D}$, un niveau $l \in \mathcal{L}$ et un nom de propriéte $p \in \mathcal{P}$, $AddProp(B, d_n, l, p)$ est une base B' dont le schéma est $\mathcal{S}' = (D'_s, C_s)$, où $D'_s = D_s \backslash \{(d_n, L, P, \prec, \rho)\} \cup \{(d_n, L, P \cup \{p\}, \prec, \rho \cup \{(p, l)\})\}$, et l'instance est $\mathcal{I}' = \mathcal{I}$.
□

Supposons que nous voulons ajouter la propriété *description* au niveau *code* de la dimension *Produit*. L'opération $AddProp(B, Produit, code, description)$ donne le résultat souhaité.

L'opérateur *Delete Property* élimine une propriété associée à un niveau. Cet opérateur prend en entrée, outre la propriété p à supprimer, le nom d_n de la dimension à modifier et le niveau l auquel p est associée.

Opérateur 8 (Delete Property)
Étant donnés une base multidimensionnelle B de schéma $\mathcal{S} = (D_s, C_s)$ et d'instance $\mathcal{I} = (D_i, C_i)$, un nom de dimension $d_n \in \mathcal{D}$, un nom de niveau $l \in \mathcal{L}$ et un nom de propriété $p \in \mathcal{P}$, le résultat de $DelProp(B, d_n, l, p)$ est une base B' dont le schéma est $\mathcal{S}' = (D'_s, C_s)$, où $D'_s = D_s \backslash \{(d_n, L, P, \prec, \rho)\} \cup \{(d_n, L, P \backslash \{p\}, \prec, \rho \backslash \{(p, l)\})\}$. L'instance de B' est $\mathcal{I}' = (D'_i, C_i)$ où $D'_i = D_i \backslash \{d_i\} \cup \{d'_i\}$, où $d_i = (\bigcup_{l_i \in L} mset(l_i), RUP, DESC)$ et $d'_i = (\bigcup_{l_i \in L} mset(l_i), RUP, DESC')$. L'ensemble DESC' contient les fonctions $desc_{p_i}^{\prime l_j}$ telles que $desc_{p_i}^{\prime l_j} = desc_{p_i}^{l_j}$, si $p_i \neq p$. □

Considérons par exemple la suppression de la propriété *description* associée au niveau *code* de la dimension *Produit*. L'opérateur $DelProp(B, Produit, code, description)$ donne le résultat souhaité.

3.2.2 Évolution de schémas de cube

L'opérateur *Create cube* crée un nouveau schéma de cube. Cet opérateur prend en entrée un nom de cube, un ensemble non vide de niveaux représentant les axes du cube et un ensemble non vide de mesures.

Opérateur 9 (Create Cube)
Étant donnés une base multidimensionnelle B de schéma $\mathcal{S} = (D_s, C_s)$ et d'instance $\mathcal{I} = (D_i, C_i)$, un nom de cube $c_n \in \mathcal{C}$, un ensemble d'axes $A \subseteq \mathcal{L}$ tel que $\|A\| \geq 1$ et un ensemble de mesures $M \subseteq \mathcal{M}$ tel que $\|M\| \geq 1$, le résultat de $CreateCube(B, c_n, A, M)$ est une base B' dont le schéma est $\mathcal{S}' = (D_s, C_s \cup \{c_s\})$, où $c_s = (c_n, A, M)$, et l'instance est $\mathcal{I}' = \mathcal{I}$. □

Considérons par exemple la création d'un cube de nom *Ventes*. Ce cube est déterminé par l'axe *code* (niveau de la dimension *Produit*) et l'axe *jour* (niveau de la dimension *Temps*). La mesure d'intérêt est la quantité d'articles vendus par jour. L'opération $CreateCube(B, Ventes, \{code, jour\}, \{quantite\})$ donne le résultat souhaité.

L'opérateur *Drop cube* supprime un schéma de cube d'une base multidimensionnelle. Aucune restriction est imposée pour effectuer son élimination. Précisons que l'instance du cube est également éliminée de la base afin de préserver sa cohérence vis à vis du schéma modifié.

Opérateur 10 (Drop Cube)
Étant donnés une base multidimensionnelle B de schéma $\mathcal{S} = (D_s, C_s)$ et d'ins-

tance $\mathcal{I} = (D_i, C_i)$*, et un nom de cube* $c_n \in \mathcal{C}$*,* $DropCube(B, c_n)$ *est une base B' dont le schéma est* $\mathcal{S}' = (D_s, C'_s)$*, où* $C'_s = C_s \backslash \{(c_n, a_1, \ldots, a_s, m_1, \ldots, m_t)\}$*, et l'instance est* $\mathcal{I}' = (D_i, C_i \backslash \{c_i\})$*, où* $c_i = \{(e_1, \ldots, e_s, v_1, \ldots, v_t) | \forall_i e_i \in mset(a_i) \wedge \forall_j v_j \in mdom(m_j)\}$*.* □

Imaginons que nous souhaitons supprimer le cube *Ventes*. Dans ce cas, l'opération $DropCube(B, Ventes)$ retire le schéma du cube de la base. Elle supprime également l'instance du cube.

Nous présentons dans la suite les opérateurs permettant de modifier les schémas de cube existants dans une base multidimensionnelle. L'opérateur *Rename Cube* change le nom d'un cube. Il prend en entrée l'ancien nom c_n du cube et le nouveau nom c'_n.

Opérateur 11 (Rename Cube)
Étant donnés une base multidimensionnelle B de schéma $\mathcal{S} = (D_s, C_s)$ *et d'instance* $\mathcal{I} = (D_i, C_i)$*, et deux noms de cube* $c_n, c'_n \in \mathcal{C}$*, le résultat de* $RenameCube(B, c_n, c'_n)$ *est une nouvelle base B' dont le schéma est* $\mathcal{S}' = (D_s, C'_s)$*, où* $C'_s = C_s \backslash \{(c_n, A, M)\} \cup \{(c'_n, A, M)\}$*, et l'instance est* $\mathcal{I}' = \mathcal{I}$*.* □

Supposons que nous voulons changer le nom du cube *Ventes* par *Cumul_ Ventes*. L'opération $RenameCube(B, Ventes, Cumul\_Ventes)$ effectue cette modification.

L'opérateur *Rename Measure* change le nom d'une mesure. Il prend en entrée l'ancien nom m d'une mesure et le nouveau nom m'.

Opérateur 12 (Rename Measure)
Étant donnés une base multidimensionnelle B de schéma $\mathcal{S} = (D_s, C_s)$ *et d'instance* $\mathcal{I} = (D_i, C_i)$*, et deux noms de mesure* $m, m' \in \mathcal{M}$*,* $RenameMeasure(B, m, m')$ *est une nouvelle base B' dont le schéma est* $\mathcal{S}' = (D_s, C'_s)$*, où* $C'_s = C_s \backslash \{(c_n, A, M)\} \cup \{(c'_n, A, M \backslash \{m\} \cup \{m'\})\}$*, et l'instance est* $\mathcal{I}' = \mathcal{I}$*.* □

Considérons par exemple le changement du nom de la mesure *quantité* du cube *Ventes* par *unités_ vendues*. Pour effectuer cette modification, il faut appliquer l'opération $RenameMeasure(B, Ventes, quantite, unites\_vendues)$.

L'opérateur *Add Axis* ajoute un axe à un cube. Il prend en entrée le nom c_n du cube à modifier, l'axe l à ajouter et un membre e de l.

Opérateur 13 (Add Axis)
Étant donnés une base multidimensionnelle B de schéma $\mathcal{S} = (D_s, C_s)$ *et d'instance* $\mathcal{I} = (D_i, C_i)$*, un nom de cube* $c_n \in \mathcal{C}$*, un niveau* $l \in \mathcal{L}$ *et un membre* $e \in mset(l)$*, le résultat de* $AddAxis(B, c_n, l, e)$ *est une base B' dont le schéma est*

$\mathcal{S}' = (D_s, C'_s)$, *où* $C'_s = C_s \backslash \{(c_n, A, M)\} \cup \{(c_n, A \cup \{l\}, M)\}$. *L'instance de* B' *est* $\mathcal{I}' = (D_i, C'_i)$, *où* $C'_i = C_i \backslash \{c_i\} \cup \{c'_i\}$, *où* $c_i = \{(e_1, \ldots, e_s, v_1, \ldots, v_t)\}$ *et* $c'_i = \{(e_1, \ldots, e_s, e, v_1, \ldots, v_t) | (e_1, \ldots, e_s, v_1, \ldots, v_t) \in c_i\}$. □

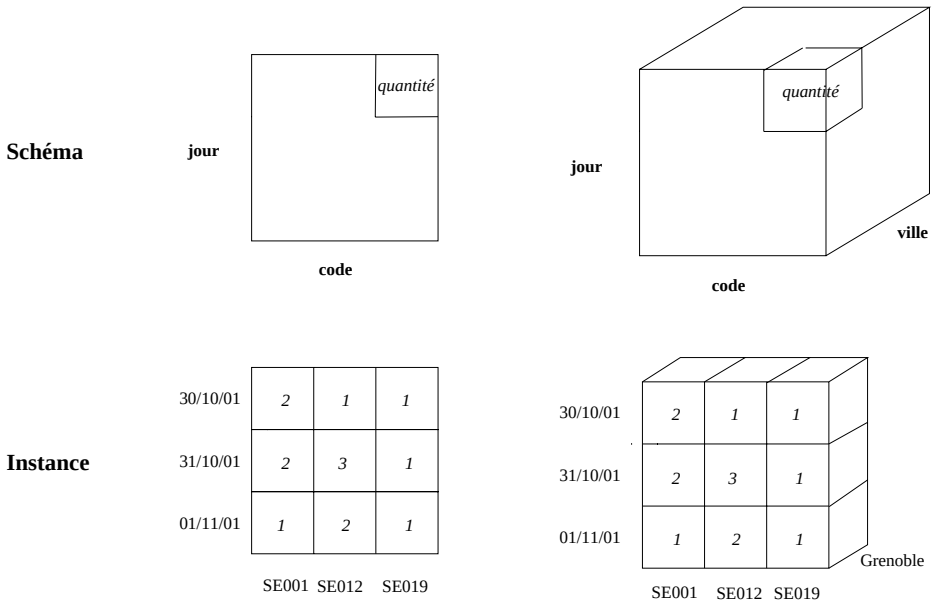


FIG. 3.5 – *Le cube Ventes après avoir ajouté l'axe ville*

Supposons que nous voulons ajouter l'axe *ville* au cube *Ventes*. Rappelons que le schéma de ce cube est (*Ventes*,{*code*, *jour*}, {*quantité*}). Dans ce cas, l'opération $AddAxis(B, Ventes, ville, Grenoble)$ modifie le schéma et l'instance du cube. Le schéma du cube devient ainsi (*Ventes*,{*code*, *jour*, *ville*},{*quantité*}). La partie supérieure de la figure 3.5 montre cette modification. La nouvelle instance du cube est calculée à partir de l'instance originale. Chacune des cellules de l'instance originale est modifiée en augmentant la valeur *Grenoble*. Cette valeur correspond à la valeur par défaut de l'axe *ville*. La partie inférieure de la figure 3.5 montre cette modification.

L'opérateur *Delete Axis* élimine un axe d'un schéma de cube. Pour pouvoir effectuer cette opération, le nombre d'axes du cube à modifier doit être supérieur ou égal à deux[2]. Cet opérateur prend en entrée le nom du cube à modifier, l'axe à supprimer et un ensemble F de fonctions d'agrégat. Chacune des fonctions f_k de F est utilisée pour agréger les valeurs de la mesure m_k du cube. Nous supposons l'existence

2. Dans le cas contraire, lorsque le cube a seulement un axe, on peut utiliser l'opérateur *DropCube* pour éliminer le cube

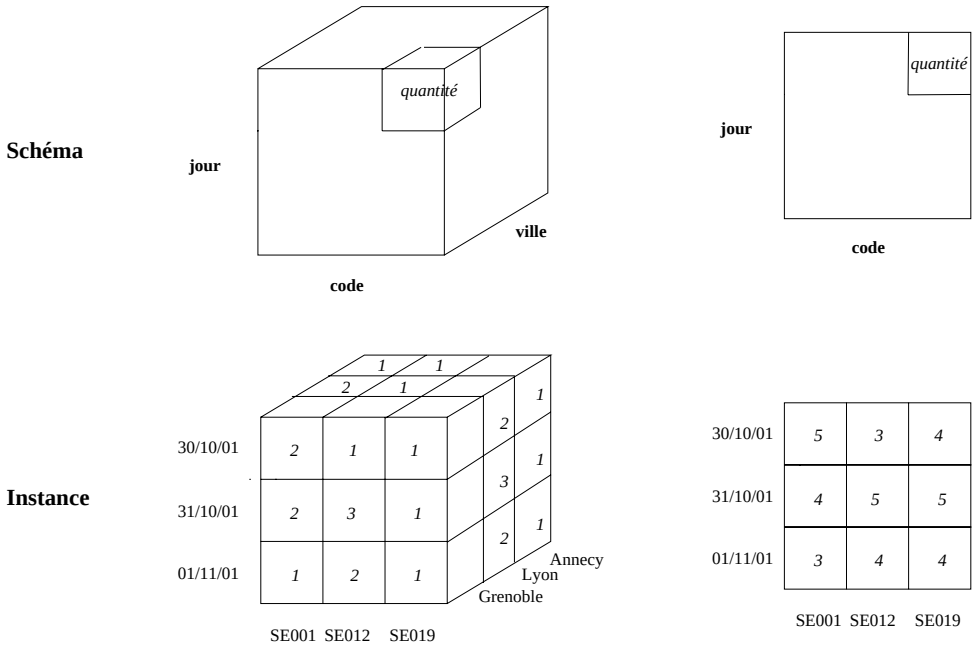


FIG. 3.6 – *Le cube Ventes*

d'une fonction σ permettant de sélectionner les cellules d'un cube qui satisfont une condition sur les axes.

Opérateur 14 (Delete Axis)
Étant donnés une base multidimensionnelle B de schéma $\mathcal{S} = (D_s, C_s)$ et d'instance $\mathcal{I} = (D_i, C_i)$, un nom de cube $c_n \in \mathcal{C}$, un axe $a_s \in \mathcal{L}$ et un ensemble F de fonctions d'agrégat $f_1 \dots f_t$, le résultat de $DeleteAxis(B, c_n, a_s, F)$ est une base B' dont le schéma est $\mathcal{S}' = (D_s, C'_s)$, où $C'_s = C_s \backslash \{(c_n, A, M)\} \cup \{(c_n, A \backslash \{a_s\}, M)\}$. L'instance de la base B' est $\mathcal{I}' = (D_i, C_i \backslash \{c_i\} \cup \{c'_i\})$, où $c_i = \{(e_1, \dots, e_s, v_1, \dots, v_t)\}$ et $c'_i = \{(e_1, \dots, e_{s-1}, f_1(\sigma_{a_1=e_1 \wedge \dots \wedge a_{s-1}=e_{s-1}}(c_i)), \dots, f_t(\sigma_{a_1=e_1 \wedge \dots \wedge a_{s-1}=e_{s-1}}(c_i))) \mid (e_1, \dots, e_s, v_1, \dots, v_t) \in c_i\}$. □

Considérons à titre d'illustration l'élimination de l'axe *ville* du cube *Ventes*. Après avoir appliqué l'opération $DeleteAxis(B, Ventes, ville, \{sum\})$, le schéma de ce cube devient $(Ventes, \{code, jour\}, \{quantité\})$. La partie supérieure de la figure 3.6 montre cette modification. La partie inférieure de la figure 3.6 montre l'adaptation des cellules du cube.

L'opérateur *Add Measure* ajoute une mesure à un schéma de cube. Il prend en entrée, outre le nom du cube à modifier, la mesure à ajouter et une valeur par défaut.

Opérateur 15 (Add Measure)
Étant donnés une base multidimensionnelle B de schéma $\mathcal{S} = (D_s, C_s)$ et d'instance $\mathcal{I} = (D_i, C_i)$, un nom de cube $c_n \in \mathcal{C}$, une mesure $m \in \mathcal{M}$ et une valeur $v \in mdom(m)$, $AddMeasure(B, c_n, m, v)$ est une base B' dont le schéma est $\mathcal{S}' = (D_s, C'_s)$, où $C'_s = C_s \backslash \{(c_n, A, M)\} \cup \{(c_n, A, M \cup \{m\})\}$ et l'instance est $\mathcal{I}' = (D_i, C'_i)$, tel que $C'_i = C_i \backslash \{c_i\} \cup \{c'_i\}$, où $c_i = \{(e_1, \ldots, e_s, v_1, \ldots, v_t)\}$ et $c'_i = \{(e_1, \ldots, e_s, v_1, \ldots, v_t, v) | (e_1, \ldots, e_s, v_1, \ldots, v_t) \in c_i\}$. □

Imaginons que nous voulons ajouter la mesure prix au schéma du cube *Ventes*. Le schéma du cube après avoir appliqué l'opérateur $AddMeasure(B, Ventes, prix, 0.0)$ devient (*Ventes*,{*code*, *jour*},{*quantité*, *prix*}). Chacune des cellules de l'instance du cube est modifiée pour stocker le prix par défaut.

L'opérateur *Delete Measure* élimine une mesure d'un schéma de cube. Cet opérateur prend en entrée le nom du cube à modifier et la mesure à supprimer. Pour pouvoir effectuer cette opération, il est nécessaire que le nombre de mesures du cube en entrée soit supérieur ou égal à deux. Chacune des cellules de l'instance du cube est modifiée en éliminant la valeur de la mesure supprimée.

Opérateur 16 (Delete Measure)
Étant donnés une base multidimensionnelle B de schéma $\mathcal{S} = (D_s, C_s)$ et d'instance $\mathcal{I} = (D_i, C_i)$, un nom de cube $c_n \in \mathcal{C}$, une mesure $m_t \in \mathcal{M}$, le résultat de $DeleteMeasure(B, c_n, m_t)$ est une base B' dont le schéma est $\mathcal{S}' = (D_s, C'_s)$, où $C'_s = C_s \backslash \{(c_n, A, M)\} \cup \{(c_n, A, M \backslash \{m_t\})\}$, et l'instance est $\mathcal{I}' = (D_i, C_i \backslash \{c_i\} \cup \{c'_i\})$, où $c_i = \{(e_1, \ldots, e_s, v_1, \ldots, v_t)\}$ et $c'_i = \{(e_1, \ldots, e_s, v_1, \ldots, v_{t-1}) | (e_1, \ldots, e_s, v_1, \ldots, v_t) \in c_i\}$. □

Imaginons maintenant que nous voulons supprimer la mesure *prix* du cube *Ventes*. Après avoir effectué $DeleteMeasure(B, Ventes, prix)$, le nouveau schéma du cube est (*Ventes*,{*code*,*jour*},{*quantité*}).

3.3 Le langage MDL

MDL (*Multidimensional Data Definition Language*) est le langage de description de données associé à notre modèle. En utilisant des expressions MDL, un utilisateur

peut créer et faire évoluer le schéma multidimensionnel de l'entrepôt. Les caractéristiques de MDL sont les suivantes :

- MDL autorise la création de schémas multidimensionnels complexes. Il autorise en particulier la création de schémas de dimension avec plusieurs niveaux et de schémas de cube avec une ou plusieurs mesures.
- MDL autorise l'évolution de ces schémas. Il autorise plus précisément l'ajout, la suppression et la modification de schémas de dimension et de schémas de cube.

Ces caractéristiques sont détaillées dans les paragraphes suivants. Nous présentons d'abord quelques notions de base, comme les règles d'écriture des noms et les types de données offerts par le langage. Nous abordons ensuite les expressions MDL autorisant l'évolution des schémas de dimension. Nous expliquons enfin les expressions du langage permettant l'évolution de schémas de cube.

3.3.1 Notions de base

MDL est destiné à être utilisé dans des situations concrètes. Pour cette raison, il prend en compte des aspects non inclus dans le modèle multidimensionnel et les opérateurs d'évolution abstraits. Ces aspects comprennent les noms des divers éléments du schéma et les types de données.

Noms

Chacun des éléments d'un schéma MDL a un nom qui l'identifie de manière univoque. Un nom doit satisfaire un ensemble de règles pour être valide :

- Un nom est un chaîne de caractères.
- Un nom doit commencer par un caractère alphanumérique.
- Un nom ne peut contenir que des caractères alphanumériques et le caractère "_".
- Un nom ne peut pas être dans la liste de mots réservés : create, alter, cube, dimension, level, measure, axe.
- Un nom doit être unique dans le schéma.

Les noms `Ventes`, `Ventes_par_article`, et `Produit123` sont des exemples de noms valides. En revanche, les noms `10prix` et `dimension` sont des exemples de noms non valides.

Types de données

MDL offre un ensemble de types de données élémentaires (cf. tableau 3.1). Ces types de données déterminent les domaines de valeurs que les niveaux, les propriétés et les mesures peuvent prendre. Les types char(n) et char varying(n) spécifient respectivement des chaînes de caractères alphanumériques de longueur fixe de n caractères et de longueur variable de n caractères au maximum. Les types numériques comprennent decimal(n,[d]), pour les nombres de n chiffres avec optionnellement d chiffres après la virgule, integer pour les entiers et float pour les nombres à virgule flottante. Le type date représente des dates sous la forme *jj/mm/aaaa*.

Type de donnée	Syntaxe	Description
Alphanumérique	char(n)	Chaîne de caractères de longueur fixe n
Alphanumérique	char varying(n)	Chaîne de caractères de n caractères maximum
Numérique	decimal(n,[d])	Nombre de n chiffres [optionnellement d chiffres après la virgule]
Numérique	integer	Entier signé
Numérique	float	Nombre à virgule flottante
Horaire	date	Date sous la forme *jj/mm/aaaa*

TAB. 3.1 – *Les types de données offerts par MDL*

La raison pour laquelle nous avons adopté les types de données décrits ci-dessus est la suivante. Comme nous le verrons dans le chapitre 4, un schéma MDL est traduit en un schéma décrit en utilisant le modèle de données (relationnel, par exemple) de l'entrepôt. Le schéma dérivé est créé en utilisant le langage de description (LDD) offert par le SGBD qui gère l'entrepôt. En définissant une correspondance simple entre les types de MDL et ceux des LDDs, la traduction est facile à réaliser. C'est ainsi que nous avons décidé d'inclure dans MDL les types ANSI. Bien entendu, si le langage de description cible ne contient pas un type MDL alors il faut considérer ce cas dans la traduction.

3.3.2 Création et suppression de dimensions

Nous consacrons ce paragraphe à la présentation de la partie de MDL dédiée à la création et la suppression de schémas de dimension.

Création d'une dimension. Soit les noms $\mathcal{D}$, $\mathcal{L}_i$, $\mathcal{L}_s$, $\mathcal{L}_t$, $\mathcal{L}_u$, $\mathcal{P}_j$ et $\mathcal{P}_v$. Une expression de la forme

create dimension $\mathcal{D}$
  (level $\mathcal{L}_i$: $\mathcal{T}_i$)$^+$
  (property $\mathcal{P}_j$: $\mathcal{T}_j$)*
  (rollup $\mathcal{L}_s$, $\mathcal{L}_t$)*
  (described_by $\mathcal{L}_u$, $\mathcal{P}_v$)*

permet de créer un schéma de dimension. Cette expression est composée de cinq parties. La première partie est la définition du nom $\mathcal{D}$ de la dimension. La deuxième partie est la définition d'un ou plusieurs niveaux. Chacun des niveaux $\mathcal{L}_i$ a un type $\mathcal{T}_i$. La troisième partie est la définition de zéro ou plusieurs propriétés. Chacune des propriétés $\mathcal{P}_j$ a un type $\mathcal{T}_j$. La quatrième partie est la définition de l'ordre existant entre paires de niveaux $\mathcal{L}_s$ et $\mathcal{L}_t$. Enfin, la cinquième partie associe les propriétés aux niveaux. Considérons par exemple l'expression suivante :

```
create dimension Produit
  level code : char(10)
  level categorie : char(15)
  level rayon : char(15)
  property nom : char(15)
  rollup code, categorie
  rollup categorie, rayon
  described_by code, nom
```

La sémantique associée à cette expression est la création du schéma de la dimension *Produit* dont les niveaux sont code, *catégorie* et *rayon*. La propriété nom est associée au niveau code.

Suppression d'une dimension. Soit $\mathcal{D}$ un nom. La sémantique associé à une expression de la forme

drop dimension $\mathcal{D}$

est la suppression du schéma (et de l'instance) de la dimension de nom $\mathcal{D}$. L'expression ci-dessous, par exemple, est utilisée pour supprimer le schéma et l'instance de la dimension *Produit*.

```
drop dimension Produit
```

3.3.3 Modification de dimensions

Nous consacrons ce paragraphe à la présentation de la partie de MDL dédiée à la modification de schémas de dimension.

Changement du nom d'une dimension. Soit les noms $\mathcal{D}$ et $\mathcal{D}'$. Une expression de la forme :

rename dimension $\mathcal{D}$ to $\mathcal{D}'$

autorise le changement de nom d'une dimension $\mathcal{D}$ par le nouveau nom $\mathcal{D}'$. Par exemple, l'expression suivante remplace le nom *Produit* d'une dimension par *Product*.

```
rename dimension Produit to Product
```

Changement du nom d'un niveau. Soit $\mathcal{L}$ et $\mathcal{L}'$ deux noms. Une expression de la forme :

rename level $\mathcal{L}$ to $\mathcal{L}'$

permet de changer l'ancien nom $\mathcal{L}$ d'un niveau de dimension par le nouveau nom $\mathcal{L}'$. L'expression suivante par exemple change le nom du niveau *code* à *codeProduit* :

```
rename level code to codeProduit
```

Ajout d'un niveau. Étant donnés trois noms $\mathcal{D}$, $\mathcal{L}_1$ et $\mathcal{L}_2$, une expression de la forme :

alter dimension $\mathcal{D}$ add level $\mathcal{L}_1 : \mathcal{T}$ rollup $\mathcal{L}_1$, $\mathcal{L}_2$ default $\mathcal{V}$

autorise l'ajout du niveau $\mathcal{L}_1$ de type $\mathcal{T}$ à la dimension de nom $\mathcal{D}$. Le niveau $\mathcal{L}_1$ devient le père de $\mathcal{L}_2$ dans le schéma de $\mathcal{D}$. Le membre par défaut du niveau $\mathcal{L}_1$

est $\mathcal{V}$. Ce membre doit être de type $\mathcal{T}$. Considérons à titre d'illustration l'expression suivante :

```
alter dimension Produit
add level marque : char(20)
rollup code, marque
default 'm1'
```

Elle autorise l'ajout du niveau *marque* au schéma de la dimension *Produit*. Ce niveau devient le père du niveau *code*. Le membre par défaut de ce niveau est 'm1'.

Suppression d'un niveau. Soit $\mathcal{D}$ et $\mathcal{L}$ deux noms. Une expression de la forme

alter dimension $\mathcal{D}$ delete level $\mathcal{L}$

dénote la suppression du niveau de nom $\mathcal{L}$ de la dimension de nom $\mathcal{D}$. Considérons à titre d'illustration l'expression :

```
alter dimension Produit
delete level categorie
```

Elle autorise la suppression du niveau *catégorie* de la dimension *Produit*.

Ajout d'une propriété. Étant donnés trois noms $\mathcal{D}$, $\mathcal{L}$ et $\mathcal{P}$, une expression de la forme :

alter dimension $\mathcal{D}$ add property $\mathcal{P}$: $\mathcal{T}$ described_by $\mathcal{L}$, $\mathcal{P}$ default $\mathcal{V}$

autorise l'ajout de la propriété $\mathcal{P}$ au niveau $\mathcal{L}$ de la dimension $\mathcal{D}$. La valeur par défaut de $\mathcal{P}$ est $\mathcal{V}$. Le type de cette valeur doit être $\mathcal{T}$. Par exemple, l'expression suivante :

```
alter dimension Produit
add property description : char(50)
described_by code, description
default 'd'
```

autorise l'ajout de la propriété *description* au niveau *code* de la dimension *Produit*. La valeur par défaut de la propriété *description* est 'd'.

Suppression d'une propriété. Étant donnés deux noms $\mathcal{D}$ et $\mathcal{P}$, une expression de la forme :

$$\text{alter dimension } \mathcal{D} \text{ delete property } \mathcal{P}$$

autorise la suppression de la propriété de nom $\mathcal{P}$ de la dimension de nom $\mathcal{D}$. L'expression suivante par exemple autorise la suppression de la propriété *description* de la dimension *Produit* :

```
alter dimension Produit
delete property description
```

3.3.4 Création et suppression de cubes

Création d'un cube. Soit $\mathcal{C}$, $\mathcal{L}_i$ et $\mathcal{M}_j$ des noms. Une expression de la forme :

$$\text{create cube } \mathcal{C} \text{ (axis } \mathcal{L}_i)^+ \text{ (measure } \mathcal{M}_j : \mathcal{T}_j)^+$$

permet de créer un schéma de cube. Cette expression est composée de trois parties. La première partie définit le nom $\mathcal{C}$ du cube. La deuxième partie définit un ou plusieurs axes. Chacun des axes $\mathcal{L}_i$ est un niveau d'une dimension. La troisième partie définit les mesures du cube. Il peut en avoir une ou plusieurs. Chacune des mesures $\mathcal{M}_j$ a un type $\mathcal{T}_j$. Considérons par exemple l'expression suivante

```
create cube Ventes
  axis code
  axis jour
  measure quantite : integer
```

Elle crée le cube *Ventes* dont les axes sont code et *jour*. La mesure de ce cube est *quantité* dont le type est entier (`integer`).

Suppression d'un cube. Étant donné un nom $\mathcal{C}$, une expression de la forme

$$\text{drop cube } \mathcal{C},$$

supprime le schéma (et l'instance) du cube de nom $\mathcal{C}$. Par exemple, l'expression suivante élimine le cube *Ventes* :

```
drop cube Ventes
```

3.3.5 Modification de cubes

MDL fournit des expressions pour modifier le schéma d'un cube. Il est possible ainsi d'exprimer l'ajout et la suppression d'axes et l'ajout et la suppression de mesures. Dans la suite nous présentons chacune de ces possibilités.

Changement du nom d'un cube. Soit $\mathcal{C}$ et $\mathcal{C}'$ deux noms. Une expression de la forme :

rename cube $\mathcal{C}$ to $\mathcal{C}'$

permet de changer l'ancien nom $\mathcal{C}$ d'un cube par le nouveau nom $\mathcal{C}'$. Supposons que l'on veut remplacer le nom *Ventes* d'un cube par *Cumul_Ventes*. L'expression appropriée est

```
rename cube Ventes to Cumul_Ventes
```

Changement du nom d'une mesure. Soit $\mathcal{M}$ et $\mathcal{M}'$ deux noms. Une expression de la forme :

rename measure $\mathcal{M}$ to $\mathcal{M}'$

permet de changer l'ancien nom $\mathcal{M}$ d'une mesure par le nouveau nom $\mathcal{M}'$. Par exemple, l'expression MDL suivante autorise le remplacement du nom de la mesure *quantité* par *unités_vendues*.

```
rename measure quantite to  unites_vendues
```

Ajout d'un axe. Soit les noms $\mathcal{C}$ et $\mathcal{L}$. Une expression de la forme :

alter cube $\mathcal{C}$ add axis $\mathcal{L}$ default $\mathcal{V}$

autorise l'ajout de l'axe $\mathcal{L}$ au cube $\mathcal{C}$. Le membre par défaut de cet axe est $\mathcal{V}$. Ce membre doit être du même type que celui de $\mathcal{L}$. Par exemple, l'expression suivante :

```
alter cube Ventes
add axis ville
default 'Grenoble'
```

permet d'ajouter l'axe *ville* au cube *Ventes*. Le membre par défaut de cet axe est 'Grenoble'.

Suppression d'un axe. Étant donnés les noms $\mathcal{C}$, $\mathcal{L}$ et $\mathcal{M}_i$, une expression de la forme :

$$\text{alter cube } \mathcal{C} \text{ delete axis } \mathcal{L} \text{ (aggregate\_with } \mathcal{M}_i, \mathcal{F}_i)^+$$

permet de supprimer l'axe de nom $\mathcal{L}$ du cube de nom $\mathcal{C}$. Chacune des $\mathcal{F}_i$ est une fonction qui est utilisée pour agréger les valeurs de la mesure $\mathcal{M}_i$ du cube. L'expression suivante supprime l'axe *ville* du cube *Ventes* :

```
alter cube Ventes
delete axis ville
aggregate_with quantite, sum
```

Ajout d'une mesure. Soit $\mathcal{C}$ et $\mathcal{M}$ deux noms. Une expression de la forme :

$$\text{alter cube } \mathcal{C} \text{ add measure } \mathcal{M} : \mathcal{T} \text{ default } \mathcal{V}$$

permet d'ajouter la mesure $\mathcal{M}$ de type $\mathcal{T}$ au schéma du cube de nom $\mathcal{C}$. La valeur par défaut de la mesure $\mathcal{M}$ pour chacune des cellules du cube est $\mathcal{V}$. Cette valeur doit être du type $\mathcal{T}$. Considérons à titre d'illustration l'expression suivante :

```
alter cube Ventes
add measure prix: float
default 0.0
```

Elle autorise l'ajout de la mesure *prix* (de type `float`) au cube *Ventes*. La valeur par défaut pour cette mesure est 0.0.

Suppression d'une mesure. Étant donnés deux noms $\mathcal{C}$ et $\mathcal{M}$, une expression de la forme :

$$\text{alter cube } \mathcal{C} \text{ delete measure } \mathcal{M}$$

permet de supprimer la mesure de nom $\mathcal{M}$ du schéma du cube de nom $\mathcal{C}$. Par exemple, l'expression MDL suivante élimine la mesure *prix* du cube *Ventes* :

```
alter cube Ventes
delete measure prix
```

3.4 Conclusion

Nous avons présenté dans ce chapitre notre modèle multidimensionnel de données. Il s'inspire principalement de la proposition faite par Cabibbo et Torlone [CT97]. Notre modèle, comme [CT97], distingue les notions de schéma de dimension et d'instance de dimension, ainsi que celles de schéma de cube et instance de cube (nommé *f-table* en [CT97]). Notre proposition diffère essentiellement sur deux points. D'une part, elle enrichit les schémas de dimension en associant des propriétés aux niveaux. D'autre part, elle généralise la notion de *f-table* en autorisant une ou plusieurs mesures par cube.

Nous avons également présenté dans ce chapitre 16 opérateurs d'évolution de schémas multidimensionnels. Ils autorisent l'ajout, la suppression, et la modification de schémas de dimension, ainsi que l'ajout, la suppression et la modification de schémas de cube. Nos opérateurs prennent en entrée une base multidimensionnelle et produisent en sortie une nouvelle base multidimensionnelle. Il est possible ainsi de composer ces opérateurs pour former des séquences complexes. L'étude des caractéristiques de notre ensemble d'opérateurs, telles que la minimalité (c'est à dire, qu'une opération ne peut pas être remplacée par une combinaison d'autres opérations) et la complétude (c'est à dire, la possibilité d'exprimer toutes les évolutions possibles d'un schéma), reste une perspective de recherche.

Nos travaux sur l'évolution de schémas multidimensionnels s'inspirent et partagent des buts communs avec d'autres travaux existants. La recherche de Hurtado et al. [HMV99a, HMV99b] est consacrée à l'étude des dimensions et des changements possibles que celles-ci peuvent subir. Ils abordent plus précisément le problème de l'évolution de schémas de dimension ainsi que celui de la mise à jour d'instances des dimensions. Nous nous sommes inspirés des opérateurs d'évolution de schémas de dimension qu'ils ont introduits dans [HMV99a] pour proposer les nôtres.

La proposition de Blaschka et al. [Bla99, BSH99] est la plus proche de la nôtre. Ils proposent un modèle multidimensionnel de données et un ensemble d'opérateurs d'évolution de schémas multidimensionnels. La principale différence de ce travail vis-à-vis du nôtre est la "granularité" à laquelle les opérateurs d'évolution sont définis. On peut considérer les opérateurs proposés dans [BSH99] comme étant de "bas niveau", dans le sens où chacun d'entre eux ajoute ou supprime des éléments spécifiques d'un schéma multidimensionnel, à savoir un niveau, un lien entre deux niveaux, une propriété, une mesure ou un lien entre un niveau et une mesure.

L'inconvénient principal de cette approche est que la cohérence du schéma peut être perdue entre deux évolutions successives. En effet, il est tout à fait possible de créer des schémas avec (i) des niveaux de dimension et des attributs isolés, (ii) des cubes sans axes et sans mesures et (iii) des relations cycliques entre les niveaux. La manière de passer d'un état cohérent à un autre est à travers la composition des opérateurs d'évolution.

Afin d'éviter ces inconvénients, nous avons préféré définir des opérateurs de "haut niveau". Nous avons défini la sémantique de chacun d'entre eux de façon à assurer le passage du schéma d'un état cohérent à un autre. De cette manière, une propriété est toujours associée à un niveau, un niveau est toujours lié à un autre niveau, la relation entre les niveaux est toujours acyclique et les cubes ont toujours au moins un axe et une mesure.

Prenant comme base notre modèle multidimensionnel et les opérateurs d'évolution de schémas multidimensionnels associés, nous avons défini MDL (*Multidimensional Data Definition Language*). C'est un langage de description de données multidimensionnelles qui autorise la définition de schémas multidimensionnels. Il autorise également l'évolution d'un schéma multidimensionnel à travers l'ajout, la suppression et la modification de schémas de dimension et de schémas de cube.

MDL presente des similarités syntaxiques avec d'autres langages de description de données multidimensionnelles. Le SGBD relationnel Oracle9i [Ora01] étend le langage SQL avec des expressions autorisant la création, la suppression et la modification de "schémas de dimension" (`create dimension`, `drop dimension`, `alter dimension`, respectivement). Ces "schémas de dimension" sont en réalité des informations supplémentaires qui informent le SGBD qu'un ensemble de relations doivent être considérées comme étant une seule et unique dimension. Le langage MDX (*MultiDimensional eXpressions*), fourni par le système multidimensionnel *Analysis Services* [Jac00], offre la commande `create cube` qui autorise la définition de schémas de cube. La définition d'un schéma de cube comprend la définition des schémas de ses dimensions ainsi que la définition de ses mesures.

Remarquons que, même si MDL a des similarités syntaxiques avec ces langages, les objectifs d'utilisation sont différents. Les "schémas de dimension" d'Oracle 9i sont utilisés pour améliorer les performances du SGBD lors de l'interrogation de l'entrepôt. Plus précisément, ils permettent à l'optimiseur de déterminer la meilleure stratégie de réécriture des requêtes en termes des vues matérialisées existantes. Soulignons que

dans ce cas la déclaration de ces "schémas de dimension" n'est pas obligatoire et que leur modification ou leur suppression n'ont pas d'impact sur les relations existantes.

En revanche, MDL est un "méta" langage de définition de données. Il peut servir de moyen de communication entre l'administrateur et un gestionnaire d'entrepôt. L'administrateur utilise MDL pour exprimer ses besoins en matière d'évolution du schéma et le gestionnaire génère, à partir du schéma modifié, des expressions dans le langage de description supporté par le SGBD gérant l'entrepôt. Nous détaillons ces aspects dans le chapitre 4.

Chapitre 4

Infrastructure pour la spécification de gestionnaires d'entrepôt

Dans ce chapitre nous présentons notre infrastructure pour la spécification de gestionnaires d'entrepôt. Un gestionnaire est un logiciel de support au cycle de vie d'un entrepôt de données. Il autorise la création du schéma de l'entrepôt et fournit les moyens pour le faire évoluer. Il fournit aussi des outils pour créer les données de l'entrepôt. Il est basé sur un modèle multidimensionnel pivot qui permet d'une part la manipulation du schéma à un niveau abstrait, et d'autre part, l'intégration des données, récupérées à partir de sources réparties, qui sont ensuite stockées dans l'entrepôt.

La section 4.1 présente notre approche pour gérer des entrepôts au travers des gestionnaires d'entrepôt. La section 4.2 décrit les données utilisées par les gestionnaires. Les sections 4.3 et 4.4 expliquent les fonctions principales des gestionnaires : création et évolution de l'entrepôt. Enfin, la section 4.5 conclut ce chapitre en discutant les bénéfices et les inconvénients de notre approche.

4.1 Gestionnaire d'entrepôt

4.1.1 Description générale

Un gestionnaire d'entrepôt (GE) est un logiciel de support au cycle de vie d'un entrepôt de données. Il fournit des interfaces d'interaction avec l'administrateur, avec le système gérant l'entrepôt et avec les sources de données (cf. figure 4.1). Il autorise ainsi l'administrateur de l'entrepôt à créer le schéma de l'entrepôt et à le modifier

même en présence de données. Il fournit aussi des outils pour construire l'entrepôt à partir de données stockées dans des sources multiples. Ces activités sont réalisées de manière transparente, c'est à dire sans considérer l'implantation (relationnelle, multidimensionnelle, à objets) de l'entrepôt, ni l'hétérogénéité des sources auxquelles il faut accéder.

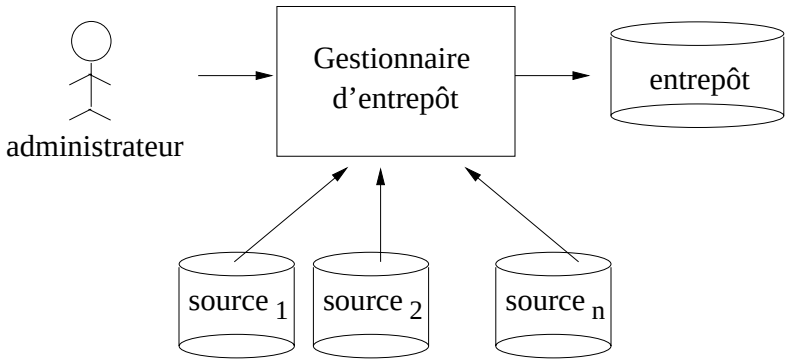


FIG. 4.1 – *Interfaces d'interaction d'un gestionnaire d'entrepôt*

Un GE établit donc une couche logicielle qui permet d'observer un entrepôt à plusieurs niveaux de représentation : logique, d'implantation et externe (cf. figure 4.2). Au *niveau logique*, un entrepôt est vu de manière abstraite en termes multidimensionnels. Le schéma de l'entrepôt à ce niveau est décrit en termes de notre modèle de données multidimensionnelles (cf. chapitre 3). Au *niveau d'implantation*, le schéma est décrit en termes du modèle de données utilisé pour implanter l'entrepôt (par exemple, le modèle relationnel, à objets, etc.). Enfin, le *niveau externe* est constitué des vues multidimensionnelles spécifiques aux différents utilisateurs.

Un GE implante notre modèle multidimensionnel et les règles de correspondance entre ce modèle et le modèle d'implantation. L'administrateur décrit un schéma multidimensionnel (formé par un ensemble de schémas de dimension et un ensemble de schémas de cube) et c'est le gestionnaire qui le traduit dans un schéma décrit en termes du modèle d'implantation. Considérons notre exemple de la chaîne de magasins. Il s'agit d'analyser les ventes de produits au cours du temps. L'administrateur définit alors les schémas des dimensions *Produit* et *Temps*. Il définit également le schéma du cube *Ventes* dont la mesure est *quantité*. Ce cube est déterminé par le *code* du produit et par le *jour* de vente. C'est à partir de ce schéma multidimensionnel que le

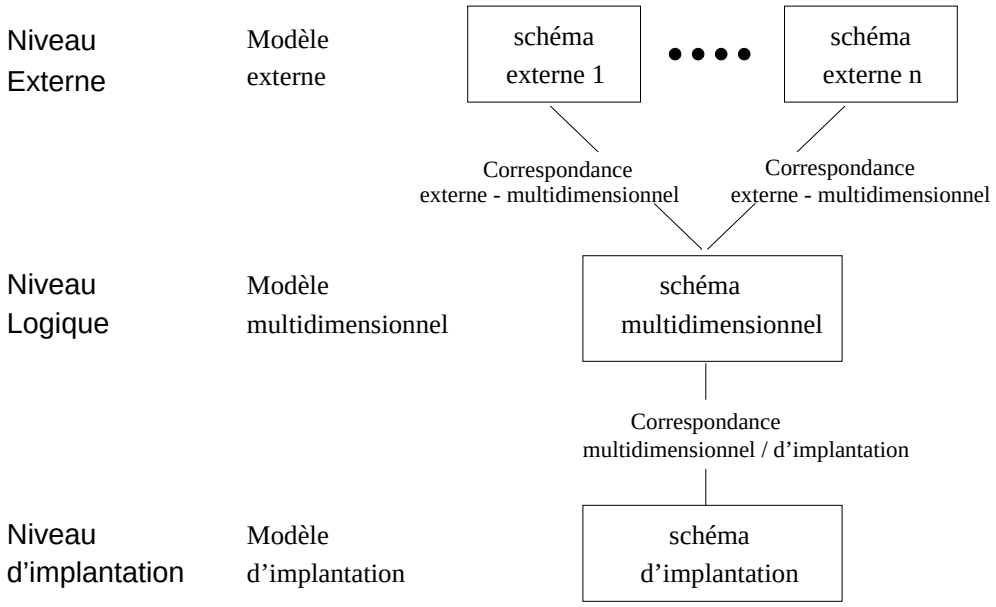


FIG. 4.2 – *Les différents niveaux de représentation d'un entrepôt de données*

gestionnaire peut générer par exemple un schéma relationnel en étoile formé par les relations de dimension Produit et Temps et par la relation de faits Ventes.

Un GE autorise l'administrateur à faire évoluer le schéma multidimensionnel. Ces évolutions entraînent la réorganisation de l'implantation de l'entrepôt : (i) la modification du schéma d'implantation et (ii) l'adaptation des données associées. Par exemple, lorsque la chaîne de magasins s'agrandit, l'analyse des ventes par ville devient intéressante. Dans ce cas, l'administrateur fait évoluer le schéma multidimensionnel en ajoutant la nouvelle dimension *Magasin* et en ajoutant le niveau *ville* de cette dimension comme axe du cube *Ventes*. Au niveau d'implantation, ces évolutions se traduisent par exemple par la création d'un nouvelle relation de nom Magasin et par l'ajout à la relation Ventes d'un attribut, en l'occurrence la clé de la relation Magasin.

Un GE effectue aussi la construction de l'entrepôt. Il réalise pour cela trois activités : intégration de données, structuration des données intégrées et stockage des données structurées. L'intégration permet d'interroger les sources contenant les données nécessaires à la construction de l'entrepôt comme s'il s'agissait d'une seule source globale avec un seul schéma et un seul langage d'interrogation. Cette source globale est virtuelle, c'est à dire que les données restent stockées dans les sources individuelles et sont extraites uniquement au moment de la construction de l'entrepôt. Nous avons décidé de décrire la source globale avec un modèle de données semi-structurées, jus-

tifiant ce choix par le fait que ce modèle s'est révélé bien adapté pour l'intégration de sources hétérogènes. La structuration a comme objectif le calcul de dimensions et de cubes dont les schémas ont été préalablement spécifiés par l'administrateur. Les données nécessaires pour calculer les dimensions et les cubes sont extraites des sources grâce à un ensemble de correspondances qui associent le schéma multidimensionnel au schéma de la source globale. Le GE transforme finalement les cubes et les dimensions en données du modèle d'implantation de l'entrepôt pour pouvoir les stocker.

4.1.2 Fonctionnement

La figure 4.3 présente l'architecture générale d'un gestionnaire d'entrepôt qui réalise trois fonctions principales : création du schéma, construction de l'entrepôt et évolution du schéma.

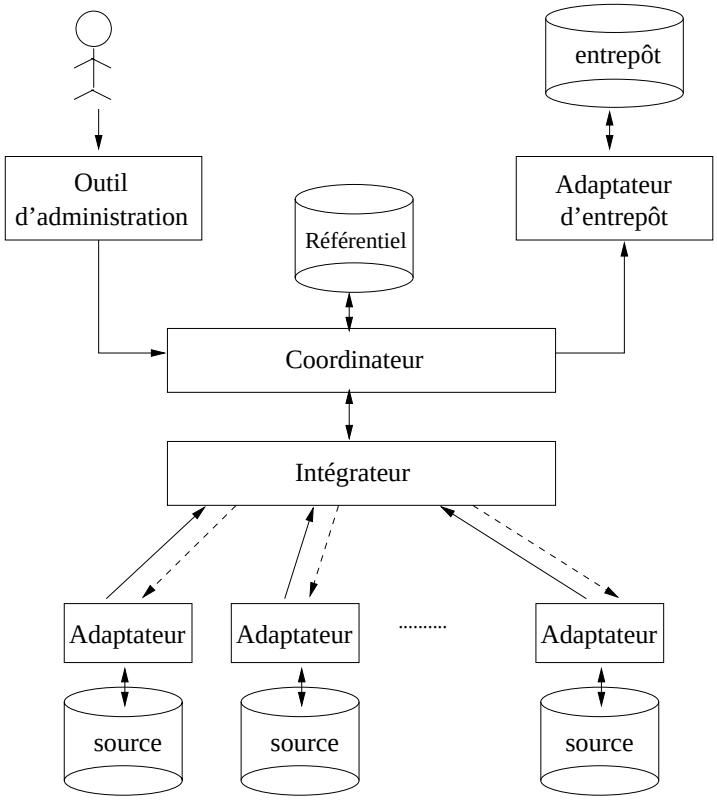


FIG. 4.3 – *Architecture générale des gestionnaires d'entrepôt*

– **Création du schéma.** L'administrateur décrit le schéma multidimensionnel de l'entrepôt à l'aide du langage MDL. L'outil d'administration vérifie que les

expressions MDL en entrée soient valides, c'est à dire qu'elles soient écrites en respectant les règles d'écriture du langage. Le coordinateur vérifie ensuite la cohérence du schéma multidimensionnel décrit. Si la vérification est positive, le schéma est stocké dans le référentiel pour une utilisation ultérieure. L'adaptateur d'entrepôt interagit avec le SGBD sous-jacent afin de créer le schéma d'implantation à partir du schéma multidimensionnel.

- **Construction de l'entrepôt.** Lorsque le début de la construction est signalé, le coordinateur récupère du référentiel les correspondances entre le schéma multidimensionnel et le schéma de la source globale. L'intégrateur utilise ces correspondances pour récupérer des sources les données nécessaires à la construction. L'adaptateur d'entrepôt interagit avec le SGBD sous-jacent afin de stocker les données calculées dans l'entrepôt.

- **Évolution du schéma.** L'administrateur exprime ses besoins en matière d'évolution de schéma à l'aide du langage MDL. L'outil d'administration accepte les expressions MDL fournies par l'administrateur et vérifie leur validité. Le coordinateur récupère du référentiel le schéma de dimension ou le schéma du cube à faire évoluer. Il vérifie ensuite si l'évolution demandée ne met pas en péril la cohérence du schéma multidimensionnel. Si ce n'est pas le cas, le schéma de dimension ou de cube modifié remplace l'ancien dans le référentiel. Le coordinateur demande alors à l'adaptateur d'entrepôt de faire évoluer le schéma d'implantation et d'adapter les données associées.

4.2 Données d'un gestionnaire

La figure 4.4 montre le diagramme UML de classes du modèle multidimensionnel implanté par tout gestionnaire. On y trouve les classes représentant les schémas de dimension et les schémas de cube.

Décrivons d'abord les classes correspondant aux schémas de dimension. La classe DimSchema modélise les schémas de dimension. Elle a comme attribut le nom de la dimension (attribut Dname). Un objet de cette classe est composé d'un ou plusieurs objets de la classe Ordre. Un objet de la classe Ordre met en relation deux objets source et target de la classe Level qui représente les niveaux de la dimension. Cette relation indique que le niveau source est le père du niveau target dans le schéma d'une dimension. Un objet de la classe Level a un nom (attribut Lname) et un type (attribut

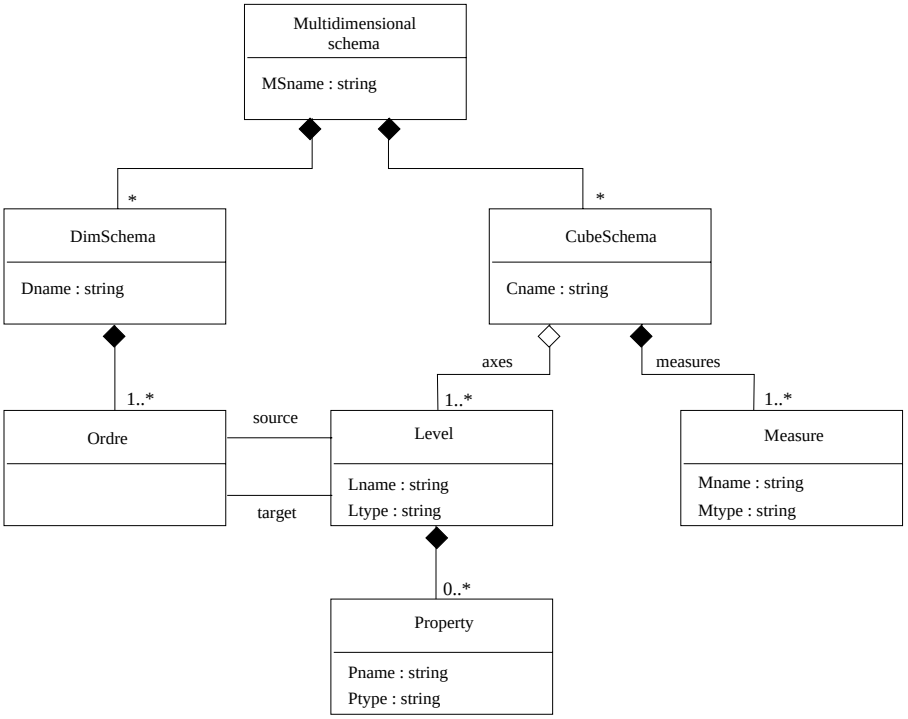


FIG. 4.4 – *Diagramme UML de classes*

Ltype). Associés à un objet de cette classe, on trouve zéro ou plusieurs objets de la classe Property, représentant les propriétés associées au niveau. Un objet de la classe Property possède également un nom (attribut Pname) et un type (attribut Ptype).

Décrivons maintenant les classes représentant les informations sur les schémas de cube. La classe CubeSchema modélise le schéma d'un cube. Cette classe a comme attribut le nom (Cname) du cube. Un objet de cette classe est composé d'un ou plusieurs objets de la classe Level (axes) et un ou plusieurs objets de la classe Measure (measures). Les objets de la classe Measure ont deux attributs : le nom de la mesure (attribut Mname) et le type associé (attribut Mtype).

4.3 Création de l'entrepôt

La création de l'entrepôt est un processus qui consiste en trois phases : création du schéma multidimensionnel, création du schéma d'implantation et construction de l'entrepôt. Nous détaillons dans la suite chacune de ces phases.

4.3.1 Création du schéma multidimensionnel

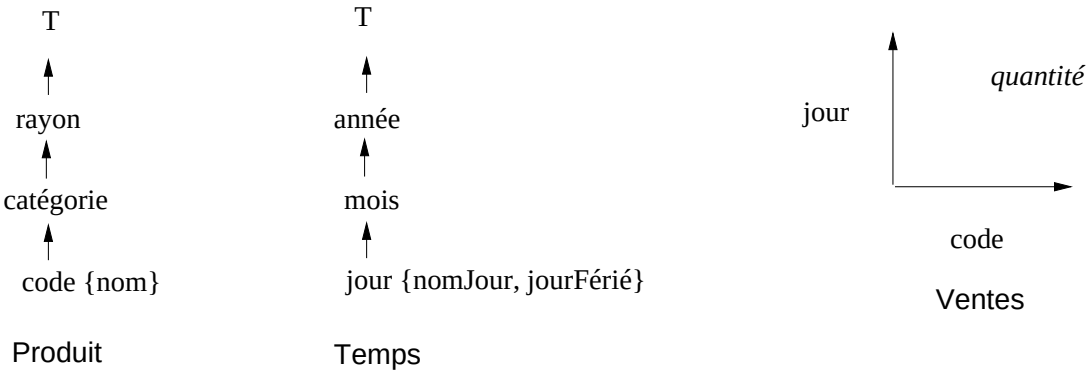


FIG. 4.5 – *Schéma multidimensionnel de l'entrepôt d'une chaîne de magasins*

L'administrateur décrit le schéma multidimensionnel de l'entrepôt à l'aide du langage MDL. Supposons que l'administrateur de l'entrepôt souhaite créer le schéma multidimensionnel montré par la figure 4.5, qui est le schéma multidimensionnel initial de l'entrepôt de la chaîne de magasins. Il est formé par les schémas des dimensions *Produit* et *Temps* et par le schéma du cube *Ventes* :

- La dimension *Produit* a trois niveaux : *code*, *catégorie* et *rayon*. Le niveau *code* caractérise les produits proposés par notre magasin. Associé à ce niveau on trouve la propriété *nom* représentant le nom du produit. Le niveau *catégorie* représente les catégories possibles dans lesquelles les produits peuvent être classés.

- La dimension *Temps* a trois niveaux : *jour*, *mois* et *année*. Associés au niveau *jour*, on trouve les propriétés *nomJour*, le nom du jour (lundi, par exemple) et *jourFérié*, une propriété indiquant s'il s'agit d'un jour férié ou non.

- Le cube *Ventes* a comme mesure *quantité* et il est déterminé par les axes *code* (niveau de la dimension *Produit*) et *jour* (niveau de la dimension *Temps*).

Les expressions MDL décrivant ce schéma sont montrées dans la figure 4.6. Chacune de ces expressions donnée en entrée à l'outil d'administration est analysée pour vérifier sa validité. Les informations sur l'opération demandée (dans ce cas la création d'un schéma de dimension ou d'un schéma de cube) et ses arguments sont envoyés ensuite au coordinateur. Le coordinateur vérifie que le schéma de dimension ou de cube

```
create dimension produit
level code: char(10)
level categorie: char(40)
level rayon: char(40)
property nom : char(30)
rollup code, categorie
rollup categorie, rayon
described_by code, nom
```

```
create dimension temps
level jour: date
level mois: char(10)
level année: char(4)
property nomJour : char(10)
property jourFérié : boolean
rollup jour, mois
rollup mois, année
described_by jour, nomJour
described_by jour, jourFérié
```

```
create cube ventes
axis code
axis jour
mesure quantité : integer
```

FIG. 4.6 – *Expressions MDL pour créer le schéma multidimensionnel de la figure 4.5*

décrit est valide. L'adaptateur d'entrepôt traduit alors le schéma multidimensionnel en un schéma d'implantation. Nous détaillons ce dernier aspect dans le paragraphe suivant.

4.3.2 Création du schéma d'implantation

Le schéma d'implantation de l'entrepôt est créé à partir du schéma multidimensionnel défini. La création est faite grâce à un ensemble de règles de traduction implanté par l'adaptateur d'entrepôt. Ces règles établissent la manière de traduire les schémas de dimension et les schémas de cubes en des schémas d'implantation cible.

Considérons par exemple que le modèle d'implantation est le relationnel et que le schéma cible est un schéma en étoile. Les règles pour traduire un schéma de dimension d_s en un schéma de relation S_d sont les suivantes :

- R1: Le nom de la relation est le nom de la dimension
- R2: Il existe un attribut id_l dans S_d, tel que l est le nom du niveau base de d_s. Cet attribut devient la clé primaire de la relation.
- R3: Il existe un attribut l dans S_d pour chaque niveau l de d_s.
- R4: Il existe un attribut p_i dans S_d pour chacune des propriétés p_i associées aux niveaux de d_s.

Les règles pour traduire un schéma de cube c_s en un schéma de relation (de faits) S_f sont les suivantes :

- R5: Le nom de la relation est le nom du cube

– R6: Il existe dans S_f un attribut id_l pour chaque axe de nom l dans le schéma de cube c_s.

– R7: Il existe un attribut m_i dans S_f pour chacune des mesures m_i dans le schéma de cube c_s.

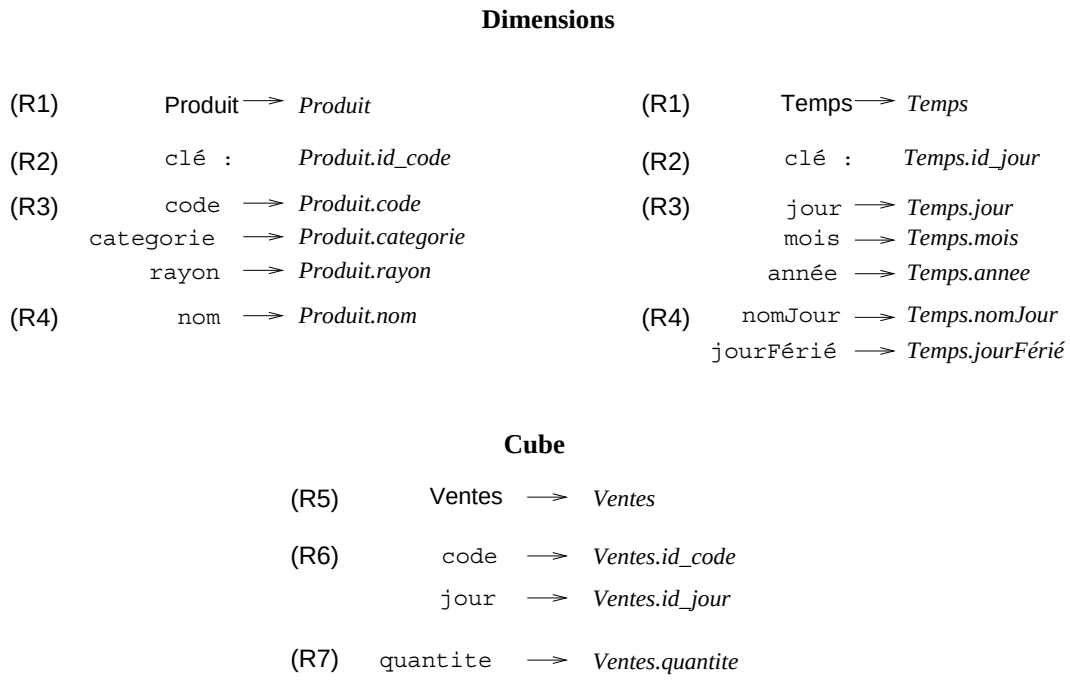


FIG. 4.7 – *Traduction du schéma multidimensionnel en un schéma relationnel en étoile*

Considérons à titre d'illustration la figure 4.7. Elle présente la traduction du schéma multidimensionnel montré dans la figure 4.5. Le schéma de la dimension *Produit* est traduit en un schéma de relation dont le nom est *Produit* (règle R1), les attributs sont *id_code*, la clé de la relation (règle R2), *code*, le code du produit (règle R3), *categorie*, la catégorie du produit (règle R3), et *nom*, le nom du produit (règle R4). Le schéma de la dimension *Temps* est traduit d'une façon similaire. Le schéma du cube *Ventes* est traduit en un schéma de relation dont le nom est *Ventes* (règle R5) et ses attributs sont *id_code* (règle R6), *id_jour* (règle R6) et *quantité* (règle R7).

Remarquons que le modèle multidimensionnel que nous avons adopté prend en compte des concepts que certains systèmes multidimensionnels n'offrent pas. Par exemple, les propriétés associées aux niveaux ne sont prises en compte que par

quelques systèmes ([Jac00], par exemple). Les règles de traduction doivent alors établir comment traiter ce type de situations. Dans le cas particulier des propriétés, une solution simple consiste à ignorer celles-ci lors de la traduction.

4.3.3 Construction de l'entrepôt

La dernière étape du processus de création de l'entrepôt est sa construction. Elle consiste en trois phases : (1) intégration des données des sources, (2) structuration des données intégrées et (3) stockage des données structurées. Nous détaillons dans la suite chacune de ces phases.

Intégration de données

L'objectif de la phase d'intégration est de fournir un accès uniforme aux sources qui contiennent les données nécessaires à la construction de l'entrepôt, en rendant transparentes leur localisation et leur hétérogénéité. Les sources peuvent ainsi être interrogées comme s'il s'agissait d'une seule source globale avec un seul schéma et un seul langage d'interrogation. Comme nous le verrons plus tard, cela est nécessaire pour faciliter la phase de structuration.

La phase d'intégration consiste alors en trois activités :

- Création du schéma global (SG).
- Spécification des schémas exportés des sources.
- Spécification des correspondances associant le schéma global aux schémas exportés des sources.

Ces activités s'appuyent sur l'existence d'un modèle commun et d'un langage associé. Nous avons choisi d'utiliser comme modèle commun un modèle de données semi-structurées. Nous adoptons en particulier le modèle OEM (*Object Exchange Model*) [PGMW95] proposé à l'Université de Stanford[1]. Nous représentons un schéma OEM à l'aide d'un *DataGuide* [GW97], un objet OEM représentant la structure d'une

1. Un objet OEM a un identificateur et une valeur atomique ou complexe. Les valeurs atomiques peuvent être entiers, réels, chaînes de caractères, etc. Une valeur complexe est une collection de zéro ou plusieurs sous-objets OEM. Chacun des sous-objets est lié à l'objet père au moyen d'un lien nommé.

base OEM. Comme langage d'interrogation, nous adoptons Lorel [AQM+97], le langage associé au modèle OEM [2]. Nous utilisons LOREL pour l'interrogation du schéma global et pour la définition des correspondances entre ce schéma et les schémas des sources. Nous détaillons ensuite les activités de la phase d'intégration.

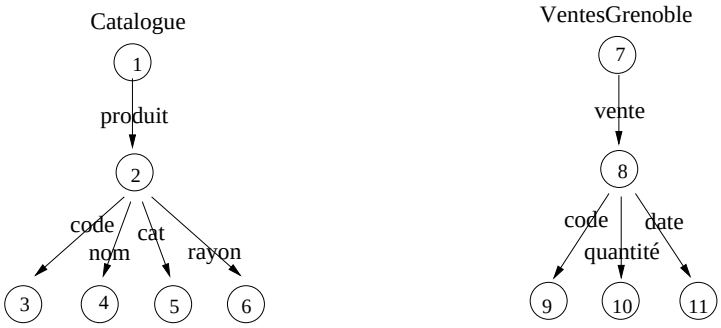


FIG. 4.8 – *Schéma global*

Création du schéma global. Le schéma global est créé comme un ensemble de schémas OEM. Le schéma global doit être complet, dans le sens qu'il doit contenir toutes les informations nécessaires à la construction de l'entrepôt. La création de ce schéma peut être faite de manière *ad hoc* ou bien en utilisant des techniques plus formelles comme celles de l'intégration de vues [SP94]. La figure 4.8 montre un schéma global décrivant les informations nécessaires pour la construction de l'entrepôt de la chaîne de magasins. Il est composé d'un catalogue de produits et d'un registre des ventes effectuées dans le magasin à Grenoble. Chacun des produits du catalogue a un code, un nom, une catégorie et un rayon. Chacune des ventes du registre a un code de produit, une date et la quantité de produits vendus à la date spécifiée.

Spécification des schémas exportés. Une fois créé le schéma global, il faut sélectionner les sources qui peuvent contribuer à répondre aux besoins d'information. Les schémas des sources sélectionnées doivent être ensuite traduits dans le modèle OEM. Supposons par exemple que les données nécessaires à la construction de notre entrepôt se trouvent dans deux sources : une source à objets, `scatalogue` et une source relationnelle `sgrenoble`. La figure 4.9 montre les schémas OEM de ces deux sources. La

2. Lorel prend ses racines en OQL, le langage d'interrogation des bases de données à objets, et fournit des fonctionnalités spécifiques à l'interrogation de données semi-structurées

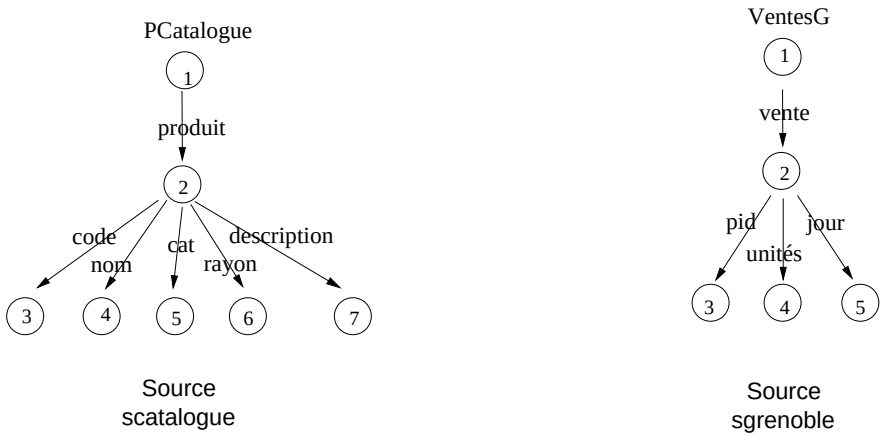


FIG. 4.9 – *Schémas OEM des sources*

source `scatalogue` fournit un catalogue de produits qui stocke, pour chaque produit, son code, son nom, sa catégorie, son rayon et une description. La source `sgrenoble` décrit les ventes journalières réalisées par le magasin localisé à Grenoble.

Spécification des correspondances. Pour associer le schéma global aux schémas exportés de sources nous adoptons l'approche *Global as View* [CHS+95, GMPQ+97], c'est à dire que le schéma global est défini comme une vue sur les schémas exportés des sources. Chacun des schémas OEM du schéma global a une requête LOREL associée permettant de calculer ses données. Cette requête est définie en termes des schémas OEM des sources. La figure 4.10 montre les requêtes associées au schéma global de notre exemple. La requête associée au schéma OEM du catalogue calcule les produits à partir des données de la source `scatalogue`. La requête associée au schéma OEM du registre de ventes récupère les ventes de la source `sgrenoble`. Il faut remarquer que les requêtes présentées ici sont très simples. Dans le cas général elles pourraient être plus complexes, nécessitant d'accéder à plusieurs sources.

L'intégration de données se fait entre l'intégrateur et les adaptateurs (cf. figure 4.3). Un adaptateur permet d'avoir une vue logique semi-structurée d'une source de données structurées (relationnelles, par exemple). Il fournit un schéma semi-structuré de la source. Il accepte des requêtes exprimées en termes de ce schéma et les traduit en requêtes exprimées dans le langage d'interrogation de la source. L'adaptateur envoie ces requêtes à la source pour leur exécution et récupère les résultats. Il traduit ensuite ces résultats en données semi-structurées. L'intégrateur est responsable de l'évaluation

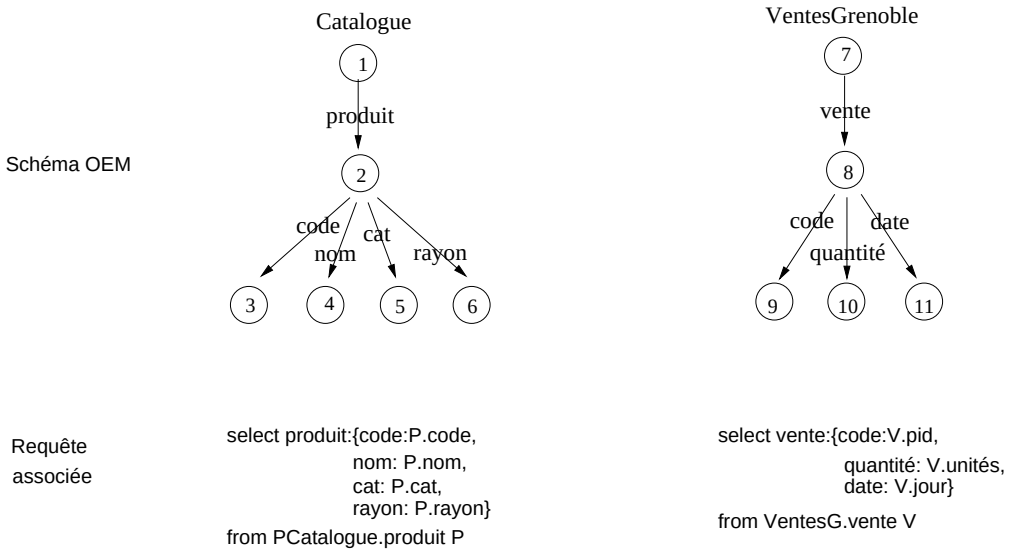


FIG. 4.10 – *Le schéma global et les requêtes associées*

des requêtes posées sur le schéma global. Pour cela, il réécrit la requête en un ensemble de requêtes posées sur les sources, envoie ces requêtes aux adaptateurs concernées, récupère les résultats partiels des adaptateurs et fusionne les résultats partiels pour construire le résultat de la requête initiale.

Structuration des données intégrées

Le phase de structuration a comme objectif la construction des dimensions et des cubes dont les schémas ont été préalablement définis par l'administrateur. La construction d'une dimension consiste à (i) calculer les membres de ses niveaux et (ii) former avec ces membres une hiérarchie d'agrégation conforme au schéma de la dimension. La construction d'un cube consiste en calculer un ensemble de cellules conformes au schéma du cube.

Un ensemble de correspondances permet d'associer les schémas de dimension et les schémas de cube au schéma global afin de récupérer les données pertinentes à leur construction. Une correspondance prend la forme d'une requête exprimée en termes du schéma global. Nous utilisons également LOREL pour exprimer ces requêtes. Considérons d'abord les correspondances entre un schéma de dimension et un SG. Dans ce cas, chaque niveau, chaque relation d'ordre entre deux niveaux et chaque propriété est associé au SG par une correspondance spécifique.

Correspondance entre un niveau et un SG. Une correspondance entre un niveau l d'une dimension d et un SG prend la forme d'une requête q exprimée en termes du SG. La requête q doit être spécifiée de telle façon que le résultat de son évaluation soit l'ensemble de membres du niveau l. Par exemple, la correspondance entre le niveau *code* de la dimension *Produit* et le schéma global montré par la figure 4.8 est la requête `'select P.code from Catalogue.produit P'`. Cette requête récupère les codes de tous les produits du catalogue.

Correspondance entre une propriéte et un SG. Une correspondance entre une propriété p (associée au niveau l d'une dimension d) et un SG est une requête q sur le SG. La requête q doit être spécifiée de telle façon que le résultat de son évaluation soit un ensemble de paires (u, v) où u est un membre du niveau l et v est la valeur de la propriété p pour le membre u. Par exemple, la correspondance entre la propriété *nom* associée au niveau *code* de la dimension *Produit* et le schéma global de la figure 4.8 est `'select P.code, P.nom from Catalogue.produit P'`. Cette requête récupère les paires formées par le code et le nom de chacun des produits du catalogue.

Correspondance entre une relation d'ordre et un SG. Une correspondance entre une relation d'ordre entre deux niveaux l_1 et l_2 d'une dimension d (tels que l_2 est père de l_1 et $l_2 \neq \top$) et un SG est une requête q sur le SG. La requête q doit être spécifiée de manière à ce que le résultat de son évaluation soit un ensemble de paires (e_1, e_2) où e_1 est un membre du niveau l_1 et e_2 est un membre du niveau l_2. Par exemple, la correspondance entre la relation d'ordre entre les niveaux *code* et *catégorie* de la dimension *Produit* et le schéma global de la figure 4.8 est la requête `'select P.code, P.cat from Catalogue.produit P'`. Cette requête récupère les paires formées par le code et la catégorie de chacun des produits du catalogue.

Notons que la correspondance qui associe une relation d'ordre entre un niveau l et le niveau $\top$ de la dimension au SG n'est pas définie. Ceci s'explique parce que tous les membres de l sont associées au membre unique t du niveau $\top$ et donc il n'est pas nécessaire d'accéder aux sources.

Considérons maintenant les correspondances entre un schéma de cube et un schéma global. Rappelons qu'un schéma de cube est un n-uplet $(c, \{l_1, \ldots, l_s\}, \{m_1, \ldots, m_t\})$, où c est le nom du cube, $\forall_{i=1,\ldots,s} l_i$ est un axe du cube et $\forall_{j=1,\ldots,s} m_j$ est une mesure. Chacune de ces mesures est associée au schéma global par une correspondance spéci-

fique. Une correspondance entre une mesure m d'un cube c et un SG est une requête q exprimée en termes du SG. La requête q doit être spécifiée de façon à que le résultat de son évaluation soit un ensemble de n-uplets de la forme $(e_1, \ldots, e_s, v)$ où $\forall_{i=1,\ldots,s} e_i$ est un membre de l'axe l_i et v est la valeur que la mesure m prend pour la combinaison de membres $e_1, \ldots, e_s$. Par exemple, la correspondance associée à la mesure *quantité* du cube *Ventes* est la requête

```
select V.code, V.date, V.quantite
from VentesGrenoble.vente V
```

Cette requête récupère un ensemble de n-uplets de la forme (code, jour, quantité) représentant la quantité vendue d'un produit (identifié par son code) un jour déterminé.

Une fois que nous avons défini les correspondances associant le schéma multidimensionnel au schéma global, nous pouvons décrire les processus de construction des dimensions et des cubes.

Considérons d'abord la construction d'une dimension. Pour construire une dimension, il faut d'abord calculer les membres du niveau base l du schéma de la dimension. Ceci est fait en exécutant la requête associée à ce niveau. Puis, pour chacun des membres m du niveau l, il faut calculer le membre m' du niveau immédiatement supérieur l' auquel m est associé. Pour cela, il est nécessaire d'évaluer la requête associée à la relation d'ordre entre les niveaux l et l'. Ensuite, il faut calculer, pour chaque membre m' du niveau l', le membre m'' du niveau immédiatement supérieur l'' auquel m' est associé. Il faut évaluer pour cela la requête associée à la relation d'ordre entre le niveaux l' et l''. Ce processus continue jusqu'à atteindre le niveau sommet du schéma de la dimension. Signalons qu'un identificateur unique (de type entier) est associé à tout membre de la dimension.

Prenons comme exemple la construction de la dimension *Produit*. Le processus de construction commence par le calcul des membres du niveau *code*. La requête associée à ce niveau est alors exécutée. L'ensemble de membres suivant est un extrait du résultat de l'exécution de cette requête :

{'SE001', 'SE012', 'SE019', 'LB095','LB018','OB293',...}.

L'étape suivante est le calcul, pour chacun des codes, de la catégorie correspondante. La requête associée à la relation d'ordre entre les niveaux *code* et *catégorie*

est alors exécutée. L'ensemble de paires (code, catégorie) suivant est une partie du résultat de l'exécution de cette requête :

```
{('SE001', 'Système d'exploitation'), ('OB293', 'Ordinateur Bureau'),...}.
```

L'étape suivante est le calcul, pour chaque catégorie, du rayon correspondant. Pour cela, il faut exécuter la requête associée à la relation d'ordre entre les niveaux *catégorie* et *rayon*. Le résultat est un ensemble de paires (catégorie, rayon). Le processus de construction se termine une fois que les membres du niveau *rayon* sont associés au membre `t` du niveau sommet $\top$ de la dimension.

Étudions à présent le processus de construction d'un cube. L'objectif de ce processus est la création d'un ensemble de cellules conformes au schéma du cube. Rappelons qu'un schéma de cube est un n-uplet de la forme $(c, \{l_1, \ldots, l_s\}, \{m_1, \ldots, m_t\})$, où c est le nom du cube, $\forall_{i=1,\ldots,s} l_i$ est un axe du cube et $\forall_{j=1,\ldots,s} m_j$ est une mesure. Pour construire un cube conforme à un schéma de cette forme, il faut (i) évaluer les requêtes associées aux mesures du cube et (ii) fusionner les résultats partiels. Nous avons précédemment vu que le résultat de l'exécution d'une requête q_j associée à une mesure m_j est un ensemble de n-uplets de la forme $(e_1, \ldots, e_s, v_j)$ où $\forall_{i=1,\ldots,s} e_i$ est un membre de l'axe l_i et v_j est la valeur que la mesure m_j prend pour la combinaison de membres $e_1, \ldots, e_n$. L'objectif de l'étape de fusion des résultats partiels est de créer enfin les cellules du cube de la manière suivante :

$$\{(e_1, \ldots, e_s, v_1, \ldots, v_t) \mid (e_1, \ldots, e_s, v_1) \in r(q_1) \wedge \cdots \wedge (e_1, \ldots, e_s, v_t) \in r(q_t)\},$$

où $r(q)$ représente l'ensemble de n-uplets résultat de l'exécution de la requête q.

Considérons à titre d'illustration la construction du cube *Ventes*. Rappelons que le schéma de ce cube est composé des axes *code* et *jour* et de la mesure *quantité*. Pour construire ce cube, il faut exécuter la requête associée à la mesure *quantité*. Le résultat de l'exécution de cette requête est un ensemble de n-uplets de la forme (code, jour, quantité) :

```
{('SE001','01/01/02',3), ('SE012','01/01/02',5),...}.
```

Dans ce cas, il n'est pas nécessaire d'effectuer l'étape de fusion des résultats partiels, parce que le cube *Ventes* n'a qu'une seule mesure (*quantité*).

Stockage des données structurées

Une fois construits, les dimensions et les cubes sont prêts pour être stockés dans l'entrepôt. Pour pouvoir les stocker, il faut les traduire en données représentées dans le modèle d'implantation de l'entrepôt. Par exemple, si le modèle d'implantation est le modèle relationnel, il faut traduire les dimensions et les cubes en relations.

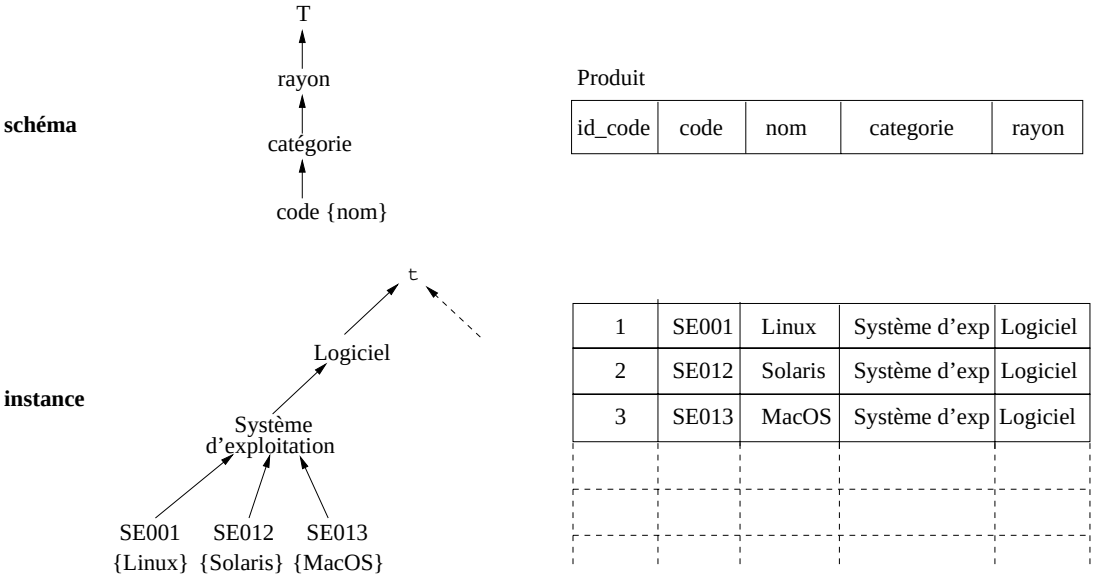


FIG. 4.11 – *Stockage relationnel de la dimension Produit*

Considérons le cas particulier d'un entrepôt relationnel dont le schéma est en étoile. Les dimensions et les cubes sont alors stockées sous la forme de relations. Étudions d'abord le stockage relationnel d'une dimension. Pour chaque membre m du niveau base l de la dimension d, il existe un n-uplet t dans la relation R_d associée à d. Chaque n-uplet t est dérivé de la manière suivante :

- La valeur de l'attribut id_l, clé de la relation, est l'identificateur du membre m.
- La valeur de l'attribut l est le membre m
- La valeur d'un attribut l' correspondant à un niveau ($l' \neq l$ et $l' \neq \top$) est le membre m' de l' qui est atteint à partir de m directement ou par transitivité.
- La valeur d'un attribut p correspondant à une propriété du niveau l est la valeur de la propriété p du membre m.

Considérons à titre d'illustration le stockage relationnel de la dimension *Produit* (cf. figure 4.11). Chacun des membres du niveau *code* de cette dimension est à l'origine d'un n-uplet de la relation `Produit`. Par exemple, le membre `SE001` est à l'origine du n-uplet `(id_code:1, code:'SE001', nom:'Linux', categorie:'Système d'exploitation', rayon:'Logiciel')`.

Étudions maintenant le stockage relationnel d'un cube. Dans ce cas, pour chacune des cellules du cube, il existe un n-uplet t dans la relation de faits R_f associée au cube. Á partir d'une cellule de la forme $(e_1, \ldots, e_s, v_1, \ldots, v_t)$, où $\forall_{i=1,\ldots,s} e_i$ est un membre de l'axe l_i et $\forall_{j=1,\ldots,t} v_i$ est la valeur de la mesure m_j, on dérive un n-uplet t de la manière suivante :

- La valeur d'un attribut id_l_i est l'identificateur du membre e_i.
- La valeur d'un attribut m_j correspondant à une mesure est la valeur v_j.

Prenons comme exemple le stockage relationnel du cube *Ventes*. Pour chacune de ses cellules, il existe un n-uplet dans la relation de faits `Ventes`. Par exemple, à partir de la cellule `('SE001', '01/01/02', 3)`, on dérive le n-uplet `(id_code: 1, id_jour: 31, quantite: 3)`, où 1 est l'identificateur de `SE001`, 31 est l'identificateur de la date '01/01/02' et 3 est la valeur de la mesure *quantité*. Le reste des n-uplets de la relation `Ventes` est dérivé de manière analogue.

4.4 Évolution de l'entrepôt

Un gestionnaire d'entrepôt autorise l'administrateur à faire évoluer le schéma d'implantation de l'entrepôt. L'administrateur exprime ses besoins en matière d'évolution à l'aide du langage MDL. Les évolutions demandées sont alors exprimées au niveau du schéma multidimensionnel et c'est le gestionnaire qui propage les changements au schéma d'implantation de l'entrepôt et adapte les données associées. Faire évoluer un entrepôt est un processus qui consiste en trois phases : modification du schéma multidimensionnel, évolution du schéma d'implantation et adaptation des instances. Nous détaillons chacune de ces phases dans la suite.

4.4.1 Modification du schéma multidimensionnel

Nous avons identifié un ensemble de modifications du schéma multidimensionnel (cf. chapitre 3). Nous avons organisé ces modifications sous la forme d'une taxonomie

qui les classe dans deux groupes : celui concernant l'évolution de l'ensemble de schémas de dimension et celui concernant l'évolution de l'ensemble de schémas de cube (cf. tableau 4.1). Nous détaillons dans la suite ces deux groupes.

Ensemble	Type d'évolution	Opération d'évolution
Dimensions	création/suppression	Créer une dimension Supprimer une dimension
	modification	Changer le nom d'une dimension Changer le nom d'un niveau Ajouter un niveau Supprimer un niveau Ajouter une propriété à un niveau Supprimer une propriété d'un niveau
Cubes	création/suppression	Créer un cube Supprimer un cube
	modification	Changer le nom d'un cube Changer le nom d'une mesure Ajouter un axe Supprimer un axe Ajouter une mesure Supprimer une mesure

TAB. 4.1 – *Taxonomie d'opérations d'évolution*

L'évolution de l'ensemble de schémas de dimension comprend la création, la suppression et la modification de schémas de dimension. La création de nouveaux schémas de dimension est nécessaire pour autoriser les utilisateurs à incorporer dans l'entrepôt des informations pour effectuer des nouvelles analyses. Par exemple, pour l'entrepôt de la chaîne de magasins, on peut créer le schéma de la dimension *Magasin* : "*ville* $\rightarrow$ *région* $\rightarrow$ *pays* ". Nous avons déjà abordé ces aspects précédemment (cf. section 4.3.1).

La suppression d'un schéma de dimension est une opération simple qui vise à l'éliminer de l'ensemble de schémas de dimension. Cette opération est utile pour éliminer les schémas de dimension que deviennent obsolètes avec le temps. Remarquons qu'un schéma de dimension ne peut pas être supprimé s'il existe un cube qui utilise un de ses niveaux comme un axe. Par exemple, le schéma de la dimension *Produit* ne peut pas être supprimé parce que son niveau *code* est un axe du cube *Ventes*. Si ce n'était pas le cas, il pourrait être supprimé.

Les paragraphes suivants détaillent les opérations de modification de schémas de dimension. Pour chacune d'entre elles, nous définissons de manière informelle la

sémantique associée et donnons un exemple. Remarquons que nous avons défini la sémantique de ces opérations de manière à assurer la cohérence du schéma multidimensionnel lors des évolutions (cf. chapitre 3 pour plus de détails).

Changer le nom d'une dimension. Il s'agit de remplacer l'ancien nom n d'une dimension par un nouveau nom n'. Ce nom ne doit pas être déjà utilisé dans le schéma multidimensionnel, afin d'assurer l'unicité des noms (invariant *noms uniques*). Cette opération autorise à l'utilisateur à corriger des erreurs (par exemple, un nom mal écrit) introduites au moment de la création du schéma de dimension ou bien de donner un nom plus approprié. Par exemple, on peut remplacer le nom de la dimension *Produit* par *Product*.

Changer le nom d'un niveau. Cette opération remplace l'ancien nom n d'un niveau avec un nouveau nom n' non utilisé dans le schéma multidimensionnel (invariant *noms uniques*). On peut par exemple remplacer le nom du niveau *code* de la dimension *Produit* par *codeProduit*. Cette opération peut être utilisée dans les mêmes circonstances (correction d'erreurs, changement par un nom plus approprié) que l'opération de changement du nom d'une dimension.

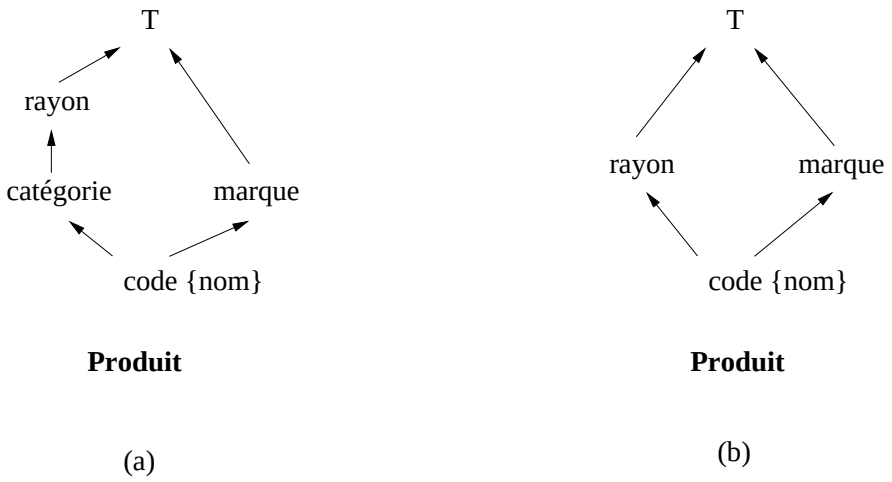


FIG. 4.12 – *Schéma de la dimension Produit après (a) ajout et (b) suppression d'un niveau*

Ajouter un niveau à une dimension. Il s'agit d'ajouter un nouveau niveau l' comme père d'un niveau l d'une dimension. Le niveau $\top$ devient le père de l' pour

assurer que le schéma de dimension soit toujours valide après cette modification. Par exemple, si l'on veut maintenant analyser les ventes par marque de produit, il faut insérer ce niveau dans le schéma de la dimension *Produit*. La partie gauche de la figure 4.12 montre le schéma de dimension résultant.

Supprimer un niveau d'une dimension. Il s'agit de supprimer le niveau l ainsi que les propriétés associées à ce niveau. Chacun des niveaux père de l devient un père des enfants de l. Ceci vise à préserver l'invariant *dimension valide*. Considérons par exemple la suppression du niveau *categorie* du schéma de la dimension *Produit*. Dans ce cas, le niveau *rayon* devient le père du niveau *code*. La partie droite de la figure 4.12 montre le schéma de dimension résultant.

Ajouter une propriété à un niveau. Cette opération associe une nouvelle propriété p au niveau l d'un schéma de dimension. On peut, par exemple, ajouter la propriété *description* au niveau *code* dans le schéma de la dimension *Produit*. Afin de préserver l'unicité des noms, le nom p ne doit pas exister dans le schéma multidimensionnel avant l'évolution.

Supprimer une propriété d'un niveau. Il s'agit de supprimer une propriété p associée au niveau l d'un schéma de dimension. Par exemple, on peut éliminer la propriété *description* associée au niveau *code* dans le schéma de la dimension *Produit* si elle n'est plus nécessaire lors des analyses.

Considérons maintenant l'évolution de l'ensemble de schémas de cube. Dans ce cas, il est possible de créer des nouveaux schémas, de supprimer les schémas obsolètes et de modifier les schémas existants. Nous avons déjà abordé les aspects concernant leur création dans la section 4.4.1. La suppression d'un schéma de cube vise, comme dans le cas des schémas de dimension, à éliminer le schéma correspondant du schéma multidimensionnel.

Dans les paragraphes suivants nous précisons les opérations de modification de schémas de cube. Pour chacune d'entre elles, nous expliquons de manière informelle la sémantique associée et donnons un exemple d'utilisation. Soulignons que nous avons défini aussi la sémantique de ces opérations de façon à préserver le cohérence du schéma multidimensionnel lors des évolutions.

Changer le nom d'un cube. Il s'agit de remplacer l'ancien nom n du cube par un nouveau nom n'. Le nouveau nom du cube ne doit pas exister dans le schéma multidimensionnel afin d'assurer l'unicité des noms. Cette opération est utile dans les situations déjà citées pour les autres opérations de changement de nom. Il est possible par exemple de remplacer le nom du cube *Ventes* par le nom *Cumul_Ventes*.

Changer le nom d'une mesure. Le nom n d'une mesure est remplacé par un nouveau nom n'. Les préconditions requises et les situations d'utilisation sont similaires aux autres opérations de changement de nom. On peut par exemple changer le nom de la mesure *quantité* par *unités_vendues*.

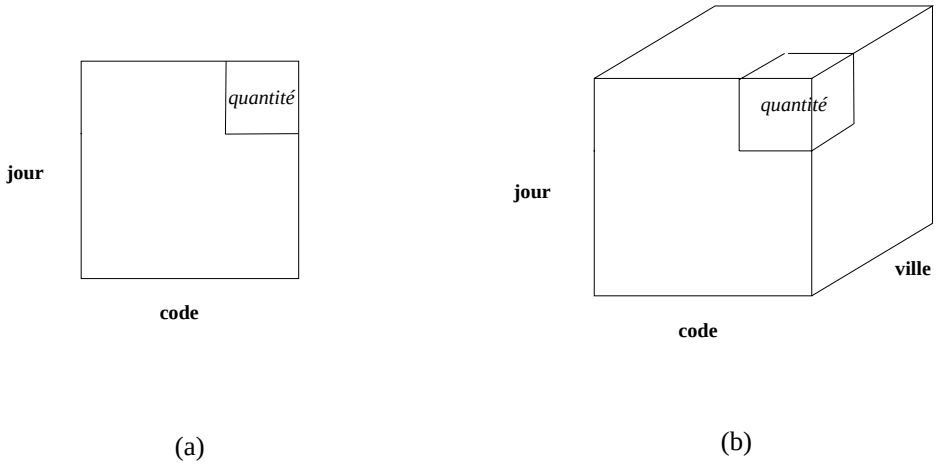


FIG. 4.13 – *Le schéma du cube Ventes (a) avant et (b) après l'ajout de l'axe ville*

Ajouter un axe à un cube. Cette opération ajoute un axe a au schéma de cube c_s. Pour préserver la validité de c_s après l'évolution, l'axe a doit être un niveau de base d'une dimension d. Par exemple, si l'on veut analyser les ventes par ville, il faut alors introduire le niveau *ville* comme un axe dans le schéma du cube *Ventes*. La figure 4.13 montre le schéma de ce cube avant et après l'évolution.

Supprimer un axe d'un cube. Il s'agit d'éliminer l'axe a du schéma de cube c_s. Par exemple, on peut supprimer l'axe *ville* du schéma du cube *Ventes*. Un axe peut être supprimé seulement si le nombre d'axes du cube est supérieur ou égal à deux. Ceci assure que le schéma c_s est toujours valide après la modification.

Ajouter une mesure à un cube. Cette opération ajoute une nouvelle mesure m au schéma de cube c_s. Par exemple, on peut ajouter la mesure *prix* au schéma du cube *Ventes*. Le nom m doit être distinct de tous les autres noms dans le schéma multidimensionnel.

Supprimer une mesure d'un cube. Il s'agit d'éliminer la mesure m du schéma de cube c_s. Par exemple, on peut supprimer la mesure *prix* du schéma du cube *Ventes*. Cette opération peut être utilisée pour supprimer les mesures obsolètes ou bien celles qui peuvent être dérivées à partir d'autres mesures[3]. Afin de préserver la validité de c_s, une mesure peut être supprimée seulement si le nombre de mesures dans c_s est supérieur ou égal à deux.

Les opérations d'évolution exposées ci-dessus sont exécutées par le coordinateur. L'exécution de ces opérations vise à modifier le schéma multidimensionnel stocké dans le référentiel. Pour cette raison, le coordinateur vérifie d'abord que les préconditions nécessaires à l'exécution d'une opération quelconque sont satisfaites. Dans le cas contraire, l'opération est rejetée.

4.4.2 Évolution du schéma d'implantation

Suite à une modification du schéma multidimensionnel, l'adaptateur d'entrepôt fait évoluer le schéma d'implantation de l'entrepôt pour le rendre conforme au schéma multidimensionnel modifié. Ceci dépende du type de schéma d'implantation (étoile, flocon, etc.) et des fonctionnalités offertes par le SGBD sous-jacent. Afin d'illustrer ces aspects, supposons que nous avons un entrepôt dont le modèle d'implantation est le modèle relationnel et le schéma est un schéma en étoile. Considérons aussi qu'il sera modifié (c'est à dire physiquement changé) suite à une évolution du schéma multidimensionnel.

Examinons d'abord les modifications que le schéma relationnel de notre entrepôt doit subir lorsque l'ensemble de schémas de dimension évolue. La création d'un schéma de dimension entraîne la création d'un schéma de relation. Par exemple, la création du schéma de la dimension *Magasin* entraîne la création du schéma de rela-

3. Il faut pas oublier le fait qu'il existe une opération pour calculer les valeurs d'une mesure. Cette opération peut être très coûteuse en termes d'accès aux sources. Il semble alors plus raisonnable de dériver une mesure à partir d'autres mesures disponibles que de la calculer complètement.

tion $Magasin[id\_ville, ville, region, pays]$. Nous avons précédemment abordé ce sujet (cf. section 4.3.2). La suppression d'un schéma de dimension se traduit simplement par l'élimination du schéma (et de l'instance) de la relation associée. Par exemple, si l'on supprime le schéma de la dimension *Produit*, la relation Produit est éliminée.

La modification d'un schéma de dimension d_s entraîne la modification du schéma de la relation R_d associée. Nous détaillons dans la suite, pour chacune des possibles modifications de d_s, les changements au schéma $S_d = R_d[id\_l_1, l_1, \ldots, l_i, p_1, \ldots, p_j]$ de R_d.

Changer le nom d'une dimension. Soit n' le nom qui remplace l'ancien nom n dans le schéma de dimension d_s. Cette opération se traduit par le changement du nom de la relation R_d. Son nouveau schéma est $S'_d = n'[id\_l_1, l_1, \ldots, l_i, p_1, \ldots, p_j]$. Par exemple, si l'on remplace le nom de la dimension *Produit* par *Product*, le schéma de relation $Produit[id\_code, code, categorie, rayon, nom]$ devient le schéma de relation $Product[id\_code, code, categorie, rayon, nom]$.

Changer le nom d'un niveau. Soit l_i le niveau à renommer et l' le nouveau nom. Cette opération se traduit par le changement du nom de l'attribut l_i par l'. Le nouveau schéma de la relation R_d est $S'_d = R_d[id\_l_1, l_1, \ldots, l', p_1, \ldots, p_j]$. Par exemple, si l'on remplace le nom du niveau *code* par *codeProduit*, le schéma de relation $Produit[id\_code, code, categorie, rayon, nom]$ devient le schéma de relation $Produit[id\_code, codeProduit, categorie, rayon, nom]$.

Ajouter un niveau à une dimension. Soit l le niveau à ajouter dans le schéma de dimension d_s. Cette opération se traduit par l'ajout de l'attribut l à S_d. Le nouveau schéma S'_d est $R_d[id\_l_1, l_1, \ldots, l_i, l, p_1, \ldots, p_j]$. Considérons l'ajout du niveau *marque* dans le schéma de la dimension *Produit*. Dans ce cas, il faut modifier le schéma de relation $Produit[id\_code, code, categorie, rayon, nom]$ en ajoutant l'attribut *marque*. Le schéma de relation résultant est $Produit[id\_code, code, categorie, rayon, marque, nom]$.

Supprimer un niveau d'une dimension. Soit l_i le niveau à supprimer dans le schéma de dimension d_s. Cette opération se traduit par la suppression de l'attribut l_i de S_d. Le nouveau schéma est $S'_d = R_d[id\_l_1, l_1, \ldots, l_{i-1}, p_1, \ldots, p_j]$. Si l'on supprime par exemple le niveau *catégorie* du schéma de la dimension *Produit*, le schéma de

relation $Produit[id\_code, code, categorie, rayon, marque, nom]$ est modifié, donnant comme résultat $Produit[id\_code, code, rayon, marque, nom]$.

Ajouter une propriété à un niveau. Soit p la propriété à ajouter dans le schéma de dimension d_s. Cette opération se traduit par l'ajout de l'attribut p à S_d. Le nouveau schéma est $S'_d = R_d[id\_l_1, l_1, \ldots, l_i, p_1, \ldots, p_j, p]$. Considérons l'ajout de la propriété *description* au niveau *code* de la dimension *Produit*. Dans ce cas, il faut ajouter l'attribut *description* dans le schéma de relation $Produit[id\_code, code, rayon, marque, nom]$. Le schéma résultat est $Produit[id\_code, code, rayon, marque, nom, description]$.

Supprimer une propriété d'un niveau. Soit p_j la propriété à supprimer dans le schéma de dimension d_s. Cette opération se traduit par la suppression de l'attribut p_j de S_d. Le schéma résultant est $S'_d = R_d[id\_l_1, l_1, \ldots, l_i, p_1, \ldots, p_{j-1}]$. Par exemple, si l'on supprime la propriété *description* associée au niveau *code* de la dimension *Produit*, le schéma de relation $Produit[id\_code, code, rayon, marque, nom, description]$ devient $Produit[id\_code, code, rayon, marque, nom]$.

Examinons maintenant les modifications que le schéma relationnel de notre entrepôt doit subir lorsque l'ensemble de schémas de cube évolue. La création d'un nouveau schéma de cube provoque la création d'un schéma de relation de faits. Dans la section 4.4.2 nous avons introduit les règles pour créer le schéma d'une relation de ce type à partir d'un schéma de cube. La suppression d'un schéma de cube se traduit par l'élimination de la relation de faits associée. Si l'on élimine par exemple le cube *Ventes* alors la relation Ventes (schéma et instance) est supprimée.

La modification d'un schéma de cube c_s conduit à la modification du schéma de la relation de faits R_f associée. Nous détaillons dans la suite, pour chacune des modifications possibles de c_s, les changements qu'il faut réaliser au schéma $S_f = R_f[id\_l_1, \ldots, id\_l_s, m_1, \ldots, m_t]$ de R_f.

Changer le nom d'un cube. Soit n' le nom qui remplace l'ancien nom n dans le schéma de cube c_s. Cette opération se traduit par le changement du nom de la relation R_f. Le nouveau schéma est $S'_f = n'[id\_l_1, \ldots, id\_l_s, m_1, \ldots, m_t]$. Considérons par exemple le changement du nom du cube *Ventes* par *Cumul_ Ventes*. Le schéma de relation $Ventes[id\_code, id\_jour, quantite]$ devient $Cumul\_Ventes[id\_code, id\_jour, quantite]$.

Changer le nom d'une mesure. Soit m'_t le nom qui remplace l'ancien nom m_t dans le schéma du cube c_s. Cette opération se traduit par le changement du nom de l'attribut m_t par m'_t. Le nouveau schéma S'_f est $R'_f[id\_l_1, \ldots, id\_l_s, m_1, \ldots, m'_t]$. Par exemple, si l'on change le nom de la mesure *quantité* du cube *Ventes* par *unités_vendues*, le schéma de relation $Ventes[id\_code, id\_jour, quantite]$ devient le schéma de relation $Ventes[id\_code, id\_jour, unites\_vendues]$.

Ajouter un axe à un cube. Soit l l'axe à ajouter au schéma de cube c_s. Cette opération se traduit par l'ajout d'un attribut id_l au schéma S_f. Le nouveau schéma S'_f est $R_f[id\_l_1, \ldots, id\_l_s, id\_l, m_1, \ldots, m_t]$. Considérons par exemple l'ajout du niveau *ville* de la dimension *Magasin* comme axe du cube *Ventes*. Dans ce cas, l'attribut id_ville est ajouté au schéma de relation $Ventes[id\_code, id\_jour, quantite]$. Le schéma de relation résultant est $Ventes[id\_code, id\_jour, id\_ville, quantite]$.

Supprimer un axe d'un cube. Soit l_s l'axe à supprimer du schéma de cube c_s. Cette opération se traduit par la suppression de l'attribut id_l_s de S_f. Le nouveau schéma S'_f est $R_f[id\_l_1, \ldots, id\_l_{s-1}, m_1, \ldots, m_t]$. Si l'on élimine l'axe *ville* du schéma du cube *Ventes*, le schéma de relation $Ventes[id\_code, id\_jour, id\_ville, quantite]$ devient $Ventes[id\_code, id\_jour, quantite]$.

Ajouter une mesure à un cube. Soit m la mesure à ajouter au schéma de cube c_s. Cette opération se traduit par l'ajout de l'attribut m à S_f. Le nouveau schéma S'_f est $R_f[id\_l_1, \ldots, id\_l_s, m_1, \ldots, m_t, m]$. Si l'on ajoute par exemple la mesure *prix* au schéma du cube *Ventes*, le schéma de relation $Ventes[id\_code, id\_jour, quantite]$ devient $Ventes[id\_code, id\_jour, quantite, prix]$.

Supprimer une mesure d'un cube. Soit m_t la mesure à supprimer du schéma de cube c_s. Cette opération se traduit par la suppression de l'attribut m_t de S_f. Le nouveau schéma S'_f est $R_f[id\_l_1, \ldots, id\_l_s, m_1, \ldots, m_{t-1}]$. Considerons à titre d'illustration la suppression de la mesure *prix* du cube *Ventes*. Dans ce cas, le schéma de relation $Ventes[id\_code, id\_jour, quantite, prix]$ devient $Ventes[id\_code, id\_jour, quantite]$.

Notons que la mise en oeuvre des modifications du schéma présentées ci-dessus dépend des fonctionnalités offertes par le SGBD sous-jacent. La plupart des systèmes relationnels à l'heure actuelle offrent des fonctionnalités primitives pour faire évoluer

le schéma des relations, plus précisément pour ajouter des attributs aux relations (commande `alter table`). L'ajout de niveaux et de propriétés peut alors se réaliser d'une manière simple. En revanche, seulement certains systèmes relationnels offrent la possibilité de supprimer directement des attributs. La modification du schéma est alors traduite par la création d'un nouveau schéma, dérivé à partir de l'ancien, qui prend seulement en compte les attributs d'intérêt.

4.4.3 Adaptation des données de l'entrepôt

Suite à une évolution du schéma, les données stockées dans l'entrepôt doivent être mises en conformité avec le schéma modifié. L'adaptateur d'entrepôt implante un ensemble de règles pour réaliser les adaptations nécessaires.

Supposons que l'entrepôt est relationnel et le schéma est un schéma en étoile. Considérons les règles suivantes pour dériver une nouvelle relation R' à partir de l'ancienne relation R. Nous utilisons les opérations de l'algèbre relationnelle pour exprimer la dérivation de R'. Nous utilisons l'opération de projection étendue pour représenter les agrégats[4] et l'opération d'extension pour introduire de nouveaux attributs dans les n-uplets d'une relation[5].

Considérons d'abord l'adaptation des relations de dimension. Soit R_d une relation de dimension dont le schéma est $R_d[id\_l_1, l_1, \ldots, l_i, p_1, \ldots, p_j]$. Les opérations de changement du nom d'une dimension ou d'un niveau n'ont pas d'impact sur la relation R_d. Nous présentons dans la suite les autres cas.

Ajouter un niveau à une dimension. Soit l le niveau à ajouter dans le schéma de dimension d_s. La nouvelle relation R'_d est dérivée comme l'extension des n-uplets de R_d par l'attribut l ayant x comme valeur par défaut : $R'_d = \varepsilon_{l=x}(R_d)$. Considérons

4. L'opération de projection pour la fonction d'agrégat f sur la relation R est définie comme suit [AHV00]:

$$\pi_{j_1,\ldots,j_m,f_k(R)} = \{(a_{j_1}, \ldots, a_{j_m}, f_k(\sigma_{j_1=a_{j_1} \wedge \cdots \wedge j_m=a_{j_m}}(R)))|(a_1, \ldots, a_n) \in R\}$$

Remarquons que la fonction d'agrégat f_k est appliquée séparément à chaque groupe de n-uplets de R, correspondant aux diverses valeurs possibles des attributs $j_1, \ldots, j_m$

5. En général, l'expression EXTEND R ADD exp Z est une relation ayant un schéma égal à celui de R étendu avec un nouvel attribut Z et dont les n-uplets t sont tels que t est un n-uplet de R augmenté d'une valeur correspondant au nouvel attribut z calculable par l'évaluation de l'expression exp sur le n-uplet de R [Dat00]. Nous utilisons la notation $\epsilon_{z=exp}(R)$ comme écriture équivalente d'une expression de ce type.

par exemple l'ajout du niveau *marque* au schéma de la dimension *Produit*. Dans ce cas, la nouvelle relation $Produit'$ est définie par $\varepsilon_{marque='m'}(Produit)$.

Supprimer un niveau d'une dimension. Soit l_i le niveau à supprimer du schéma de dimension d_s. La nouvelle relation R'_d est dérivée comme étant la projection de R_d sur l'ensemble d'attributs $A\backslash\{l_i\}$, où A est l'ensemble d'attributs dans le schéma de R_d : $R'_d = \pi_{id\_l_1,l_1,\ldots,l_{i-1},p_1,\ldots,p_j}(R_d)$. Par exemple, si l'on supprime le niveau *catégorie* de la dimension *Produit*, alors la nouvelle instance de la relation $Produit'$ est définie par $\pi_{id\_code,code,rayon,marque,nom}(Produit)$.

Ajouter une propriété à un niveau. Soit p la propriété à ajouter dans le schéma de dimension d_s. La nouvelle relation R'_d est dérivée comme l'extension des n-uplets de R_d par l'attribut p ayant x comme valeur par défaut : $R'_d = \varepsilon_{p=x}(R_d)$. Considérons par exemple l'ajout de la propriété *description* au niveau *code* dans le schéma de la dimension *Produit*. Ici, la nouvelle relation $Produit'$ est dérivée comme $\varepsilon_{description='desc'}(Produit)$, où la valeur par défaut de la propriété *description* pour les n-uplets de la relation est 'desc'.

Supprimer une propriété d'un niveau. Soit p_j la propriété à supprimer du schéma de dimension d_s. La nouvelle relation R'_d est dérivée comme la projection de R_d sur l'ensemble d'attributs $A\backslash\{p_j\}$, où A est l'ensemble d'attributs dans le schéma de R_d : $R'_d = \pi_{id\_l_1,l_1,\ldots,l_i,p_1,\ldots,p_{j-1}}(R_d)$. Par exemple, si l'on supprime la propriété *description* de la dimension *Produit*, alors la nouvelle instance de la relation $Produit'$ est dérivée par $\pi_{id\_code,code,rayon,marque,nom}(Produit)$.

Étudions maintenant l'adaptation des relations de faits. Soit R_f une relation de faits dont le schéma est $R_f[id\_l_1,\ldots,id\_l_s,m_1,\ldots,m_t]$. Les opération de changement du nom d'un cube ou d'une mesure n'ont pas d'impact sur R_f. Nous présentons dans la suite les autres cas.

Ajouter un axe à un cube. Soit a l'axe à àjouter dans le schéma de cube c_s. La nouvelle relation R'_f est dérivée comme l'extension des n-uplets de R_f par l'attribut id_a ayant x comme valeur par défaut : $R'_f = \varepsilon_{id\_a=x}(R_d)$. Considerons par exemple l'ajout de l'axe *ville* au cube *Ventes*. Dans ce cas, la nouvelle relation $Ventes'$ est définie par $\varepsilon_{id\_ville=1}(Ventes)$.

Supprimer un axe d'un cube. Soit l_s l'axe à supprimer du schéma de cube c_s. La nouvelle relation R'_f est dérivée comme la projection étendue de la relation R_f sur l'ensemble d'attributs $A \backslash \{id\_l_s\}$, où A est l'ensemble d'attributs dans le schéma de R_f : $\pi_{id\_l_1,...,id\_l_{s-1},f_1(m_1),...,f_t(m_t)}(R_f)$. Par exemple, si l'on supprime l'axe *ville* du cube *Ventes*, la nouvelle relation $Ventes'$ est définie par $\pi_{id\_code,id\_jour,sum(quantite)}(Ventes)$.

Ajouter une mesure à un cube. Soit m la mesure à ajouter dans le schéma de cube c_s. La nouvelle relation R'_f est dérivée comme l'extension des n-uplets de R_f par l'attribut m ayant x comme valeur par défaut : $R'_f = \varepsilon_{m=x}(R_f)$. Par exemple, si l'on ajoute la mesure *prix* au cube *Ventes* alors la nouvelle relation $Ventes'$ est définie par $\varepsilon_{prix=0.0}(Ventes)$.

Supprimer une mesure d'un cube. Soit m_t la mesure à supprimer du schéma du cube c_s. La nouvelle instance R'_f est dérivée comme la projection de la relation R_f sur l'ensemble d'attributs $A \backslash \{m_t\}$, où A est l'ensemble d'attributs dans le schéma de R_f : $R'_f = \pi_{id\_l_1,...,id\_l_s,m_1,...,m_{t-1}}(R_f)$. Par exemple, si l'on supprime la mesure *prix* du cube *Ventes*, alors la nouvelle relation $Ventes'$ est définie par $\pi_{id\_code,id\_jour,quantite}(Ventes)$.

Plusieurs facteurs influencent la façon d'adapter les données suite à une évolution du schéma. Le volume de données est un facteur important à considérer. Lorsque le volume est faible, on peut recalculer les données. Ceci est le cas des dimensions. Lorsque le volume de données est important, comme dans le cas des cubes, il devient irréaliste de tout recalculer. Dans ce cas, il est plus raisonnable de réutiliser les données déjà stockées dans l'entrepôt pour en dériver de nouvelles.

Les fonctionnalités offertes par le SGBD sous-jacent influencent également la façon d'adapter les données. Par exemple, si l'on utilise un système relationnel, il est possible de recalculer ou de dériver les relations. Si l'on utilise un système multidimensionnel, la seule manière possible d'adapter les données est de les recalculer. Ceci est dû à ce que les fonctionnalités offertes par ces systèmes en ce qui concerne la manipulation des données sont à l'heure actuelle très limitées.

4.5 Conclusion

Nous avons présenté dans ce chapitre notre approche pour la gestion de l'évolution des entrepôts de données. Nous avons introduit des systèmes (que nous appelons

gestionnaires d'entrepôt) autorisant l'évolution du schéma des entrepôts de données. Un gestionnaire d'entrepôt implante le modèle multidimensionnel de données présenté dans le chapitre 3, dont les concepts (*dimension*,*cube*, *niveau*, *mesure*) ont une sémantique riche. L'administrateur de l'entrepôt décrit donc un schéma multidimensionnel et c'est le gestionnaire qui le traduit dans un schéma décrit en termes du modèle d'implantation de l'entrepôt. L'administrateur fait évoluer le schéma multidimensionnel et le gestionnaire traduit alors les évolutions au schéma d'implantation.

Nous avons également vu que la manière de faire évoluer le schéma de l'entrepôt et adapter les données associées dépend de plusieurs facteurs. Parmi eux, on peut citer le modèle d'implantation adopté, le type de schéma choisi (étoile, flocon) et les fonctionnalités offertes par le SGBD sous-jacent. Cette constatation renforce notre motivation pour fournir une infrastructure autorisant la création d'un gestionnaire mieux adapté à une situation concrète.

Une avantage de notre approche est que les gestionnaires permettent de réaliser l'indépendance entre schéma multidimensionnel et schéma d'implantation. Ces deux schémas décrivent les mêmes données mais à des niveaux différents. Il s'agit donc que l'administrateur puisse définir un schéma multidimensionnel et ensuite qu'il puisse choisir le schéma d'implantation le plus adéquat en tenant compte des critères de performances et de la flexibilité d'accès demandés par les applications d'analyse à utiliser.

Un inconvénient de notre approche est qu'un gestionnaire doit vérifier que les correspondances entre le schéma multidimensionnel et le schéma d'implantation sont toujours valides. Dans le cas contraire, le fonctionnement correct du gestionnaire n'est plus assuré. Il est donc important de considérer des mécanismes de reprise après d'éventuelles pannes du matériel informatique où le gestionnaire est exécuté.

Chapitre 5

Implantation d'un gestionnaire d'entrepôt

Ce chapitre présente la mise en œuvre d'un gestionnaire d'entrepôt. L'objectif était de valider notre proposition de manière expérimentale. Nous nous sommes intéressés à l'implantation d'un gestionnaire pour un entrepôt relationnel dont le schéma est en étoile[1]. Le gestionnaire d'entrepôt relationnel (GER) implante notre modèle multidimensionnel de données et fournit des outils pour créer et faire évoluer le schéma de l'entrepôt. Il fournit également des outils pour construire l'entrepôt à partir de données stockées dans plusieurs sources.

Nous avons privilégié l'utilisation de technologies indépendantes de toute plate-forme matérielle pour réaliser notre prototype. Ainsi nous avons utilisé le langage à objets Java pour programmer un ensemble de fonctionnalités de base. Nous avons aussi utilisé le langage de description de données XML, et les technologies associées, pour représenter et manipuler des données au sein du gestionnaire. Le langage XML permet de définir l'information sous un format indépendant de la plate-forme qui produit cette information ou qui l'utilise. Java permet de traiter l'information représentée en XML de manière indépendante de la plate-forme qui exécute l'application.

Ce chapitre est organisé de la manière suivante. La section 5.1 introduit l'infrastructure technique utilisée pour l'implantation du GER. La section 5.2 présente l'architecture du gestionnaire et ses principales fonctions. Les sections 5.3 à 5.8 détaillent les composants implantés. La section 5.9 présente les expérimentations que

1. Une version préliminaire du prototype que nous décrivons dans ce chapitre est présenté dans [Lon01]

nous avons menées afin de tester le GER. La section 5.10 conclut ce chapitre en discutant les résultats et les perspectives envisageables.

5.1 Infrastructure technique

Nous détaillons ici l'infrastructure technique choisie pour réaliser l'implantation de notre prototype. Nous présentons en particulier le langage XML, et les technologies associées à ce langage, que nous avons utilisées pour l'implantation de notre prototype : le langage d'interrogation XML-QL et l'interface de programmation d'applications (API) *Document Object Model* (DOM). Nous invitons le lecteur à consulter des ouvrages tels que [Fla02] pour une discussion approfondie du langage à objets Java.

5.1.1 Le langage XML

XML (*eXtensible Markup Language*) [XML98] est un langage de description de documents structurés qui est devenu un standard pour la représentation de données. Il se trouve à l'heure actuelle à la base de nombreuses applications sur l'Internet et le WWW. XML permet de représenter des données grâce à un système de balises défini par l'utilisateur. Grâce aux balises proprement définies, les données peuvent être traitées par leurs sémantiques. Un document XML est composé des éléments suivants :

- Un élément racine. Il constitue le point d'entrée dans la hiérarchie d'informations du document.
- Un ensemble d'éléments. Les éléments XML sont délimités par des balises et constituent les unités d'informations de la hiérarchie du document. Un élément peut être défini par une balise de début et de fin, s'il contient d'autres éléments, ou par des nœuds texte. Il peut aussi être vide et comporter une seule balise.
- Des attributs associés aux éléments. Les attributs d'un élément sont précisés à l'intérieur d'une balise de début ou d'élément vide.
- Des nœuds texte. Un nœud texte (Text et CDATA) est une chaîne de caractères (excluant `<`, `>` et `&`).

```
<!ELEMENT geog (pays*)>
<!ELEMENT pays (nomp, region*)>
<!ELEMENT region (nomr, ville*)>
<!ELEMENT nomp (#PCDATA)>
<!ELEMENT nomr (#PCDATA)>
<!ELEMENT ville (#PCDATA)>
```

```
<geog>
  <pays>
     <nomp> France </>
     <region>
        <nomr> Rhone-Alpes </>
        <ville> Grenoble </>
        <ville> Lyon </>
        ...
     </>
     ...
  </>
</>
```

FIG. 5.1 – *Représentation XML d'un catalogue de pays*

La structure arborescente du document XML (intitulé des balises, imbrications des balises, caractère obligatoire ou facultatif des balises et de leur ordre de succession) peut être déclarée formellement. Cette déclaration s'appelle une Définition de Type de Document (DTD). Une DTD est un ensemble de définitions des éléments et des attributs du document XML, en précisant les règles de construction de ce document. Elle s'effectue selon un formalisme particulier défini lui-aussi dans la spécification XML. En XML cette déclaration est facultative, ce qui donne une grande souplesse aux développeurs. On n'écrira donc une DTD que lorsqu'il y aura vraiment intérêt à le faire, par exemple pour contraindre le contenu du document XML. Lorsqu'un document XML possède une DTD associée et la respecte, on dit qu'il est valide. Lorsqu'il respecte seulement les règles de la grammaire XML (balises fermées, correctement imbriquées) on dit qu'il est bien formé.

Supposons que nous voulons représenter des informations géographiques. Nous voulons en particulier représenter les pays, leur régions et leur villes. La partie gauche de la figure 5.1 montre une DTD conçu à cet effet. Ici, l'élément `geog` est un ensemble d'éléments `pays`. Chacun d'entre eux comprend le nom (balise `nomp`) et un ensemble d'éléments `region`. Chacun de ces éléments a un nom (balise `nomr`) et un ensemble d'éléments `ville`. La partie droite de la figure 5.1 montre un extrait d'une instance possible de cette DTD. Ici, on peut voir que le pays `France` a une région nommée `Rhone-Alpes` et que dans cette région se trouvent, entre autres, les villes `Grenoble` et `Lyon`.

Nous avons opté pour l'utilisation de XML pour représenter des données au sein de notre prototype pour la raison suivante. Comme il a été dit précédemment, la construction des entrepôts de données nécessite l'accès intégré à des données sto-

ckées dans des sources autonomes et hétérogènes, avec des modèles de données et des mécanismes d'accès différents. XML facilite la fusion de données hétérogènes en proposant une représentation flexible des données non structurées, structurées et semi-structurées. Il faut remarquer cependant que le langage XML ne résout pas par lui-même les conflits structurels et sémantiques des sources [BB01]. Les conflits structurels sont le résultat de la flexibilité de représentation : il est possible d'avoir plusieurs représentations (utilisant différents systèmes de balises) d'une même donnée. Les conflits sémantiques existent parce que, même si les données sont représentées de manière uniforme en XML, il est possible qu'une même donnée soit interprétée de plusieurs manières. Ceci est un des problèmes abordés dans le cadre du projet XYLEME [CR01].

5.1.2 Le langage d'interrogation XML-QL

L'interrogation de données XML est reconnue aujourd'hui comme un véritable besoin. Il n'existe pas cependant à l'heure actuelle un langage d'interrogation standard. Plusieurs langages proposés, tels que XQL [RLS98] et XQuery [DeR98], ont été conçus suivant une approche recherche d'information et permettent d'interroger de données XML stockées dans une seule source. D'autres propositions, comme LOREL [Wid98] et XML-QL [FDL+99], ont été conçus suivant une approche bases de données et permettent d'interroger des données stockées dans une ou plusieurs sources.

XML-QL est un langage déclaratif conçu pour interroger une ou plusieurs sources de données XML. Ce langage est fondé sur les "patrons XML", c'est à dire des documents XML où les éléments sont remplacées par des variables ou des prédicats. Les patrons XML sont utilisés pour exprimer les conditions de sélection des données XML sources et pour définir la structure des données XML résultantes.

Une requête XML-QL est constituée d'une construction `where-construct` similaire à la construction `select-from-where` de SQL. La partie `where` permet de sélectionner les données d'une ou plusieurs sources et la partie `construct` correspond à la définition de la structure des données XML résultantes. Considérons par exemple la requête montrée dans la figure 5.2. Dans ce cas, il s'agit de trouver, pour chacune des villes stockées dans la source `spjaunes`, la région où elles se localisent. Cette information se trouve dans la source `sgeog`.

Nous avons choisi XML-QL pour interroger les sources de données XML pour deux raisons. La première raison est que ce langage autorise l'expression de jointures,

```
where
  <pjaunes>
    <magasin>
      <ville> $v </>
    </>
  </>IN spjaunes,
  <geog>
    <pays>
      <region>
        <nomr> $r </>
        <ville> $v </>
      </>
    </>
  </> IN sgeog
construct
  <vr>
    <ville> $v </>
    <region> $r </>
  </>
```

FIG. 5.2 – *Exemple de requête XML-QL*

ce qui est important pour récupérer des données à partir de plusieurs sources. La deuxième raison est d'ordre pragmatique : l'interpréteur est disponible au public[2]. Il est écrit en Java, ce qui nous a permis de l'intégrer aisément dans notre prototype.

Nous avons défini et utilisé un sous-ensemble de XML-QL pour faciliter la programmation de certains composants de notre prototype. Le sous-ensemble choisi prend en compte les caractéristiques principales de XML-QL : patrons XML et prédicats, entre autres. Il ne prend pas en compte les fonctions, les requêtes imbriquées, les fonctions de Skolem et les expressions de chemins régulières. Ainsi, ce qui apporte la plus grande flexibilité de XML-QL a été omis pour des raisons de rapidité d'implémentation. La grammaire du sous-ensemble de XML-QL que nous avons utilisée se trouve en annexe (cf. Annexe B).

5.1.3 L'interface de programmation d'applications DOM

Les interfaces de programmation d'applications (API) pour XML permettent notamment l'analyse et la manipulation de documents XML. Il existe à l'heure actuelle deux interfaces standardisées par le consortium W3C : SAX et DOM. L'interface de programmation SAX (*Simple API for XML*) analyse un document XML selon un approche événementielle. SAX présente le document comme un flot d'événements correspondant à l'ouverture d'une balise d'un élément, sa fermeture ou l'analyse d'un

2. Disponible à l'adresse `http://www.research.att.com/sw/tools/xmlql`

nœud texte. L'application utilisant SAX doit implémenter les fonctions de traitement de ces événements.

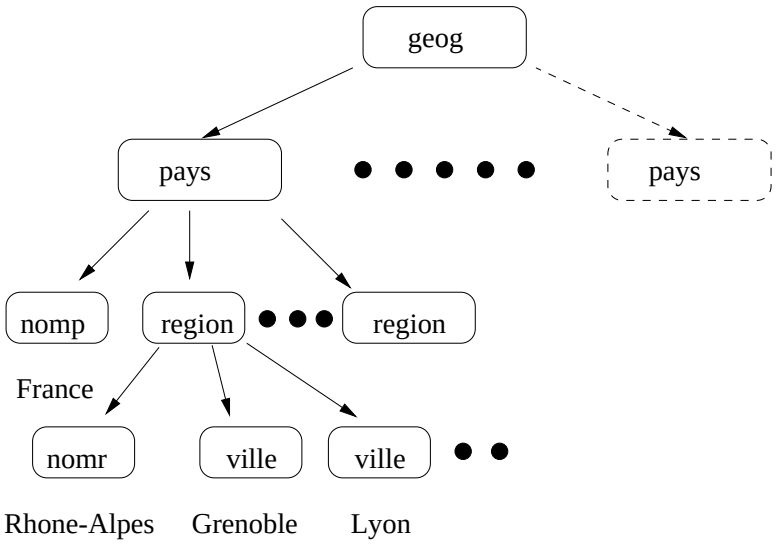


FIG. 5.3 – *Exemple d'arbre DOM*

L'interface *Document Object Model* (DOM) [DOM98] suit une approche hiérarchique. Elle construit une représentation arborescente du document en mémoire pour permettre à une application de la manipuler. Le document est alors montré dans son ensemble. Les opérations de manipulation fournies par DOM comprennent le parcours du document, l'ajout ou la suppression d'éléments ou d'attributs, le réordonnancement des éléments et leur renommage. La figure 5.3 montre la représentation arborescente du document montré par la figure 5.1. La racine de l'arbre correspond à l'élément racine `geog`. Chacun des nœuds du deuxième niveau correspond à un élément `pays`, ses éléments se trouvant comme nœuds du troisième niveau.

Nous avons opté pour l'utilisation de l'interface DOM pour la manipulation de documents XML. Lorsque les données XML doivent être manipulées dans leur ensemble et restructurées, cette interface apporte une souplesse importante. L'inconvénient est qu'il faut payer une empreinte mémoire conséquente. L'interface SAX est plus appropriée pour des applications qui ont seulement besoin d'opérations de recherche. Nous avons utilisé en particulier l'implantation de DOM proposée par IBM, nommée XML4J[3], pour la programmation de notre prototype.

3. Disponible à l'adresse `http://www.alphaworks.ibm.com/tech/xml4j`

5.2 Architecture du gestionnaire d'entrepôt implanté

La figure 5.4 montre l'architecture du gestionnaire d'entrepôt implanté. Le référentiel stocke les informations utilisées par le gestionnaire. L'administrateur interagit avec le gestionnaire d'entrepôt par l'intermédiaire d'un outil d'administration. Cet outil accepte des expressions du langage MDL (cf. chapitre 3) pour la création et la modification du schéma multidimensionnel de l'entrepôt. Notre gestionnaire permet la création de schémas de dimension où la relation entre les niveaux est linéaire (c'est à dire, les hiérarchies alternatives ne sont pas supportées) et de schémas de cube avec une seule mesure. Les informations collectées par l'outil d'administration sont transmises au coordinateur, qui coordonne les activités au sein du gestionnaire.

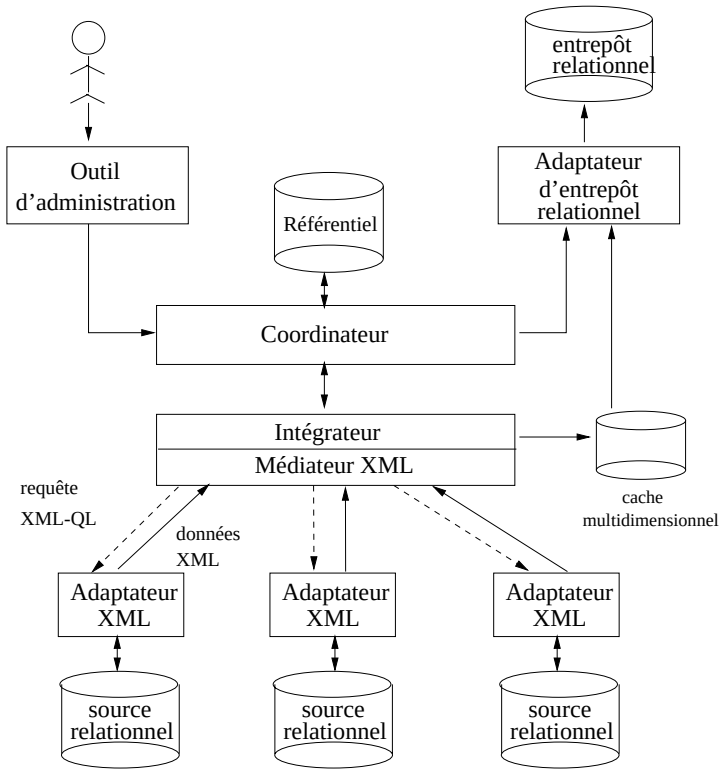


FIG. 5.4 – *Architecture du gestionnaire d'entrepôt implanté*

La fonction de l'intégrateur est de calculer les données à stocker dans l'entrepôt. Pour cela, il récupère auprès des sources les données nécessaires à la construction de l'entrepôt et structure celles-ci sous la forme de dimensions ou de cubes. Les données pertinentes à la construction sont extraites des sources grâce à un sous-système de médiation qui permet d'interroger les sources individuelles comme s'il s'agissait d'une

seule source globale virtuelle. Un ensemble de correspondances associe le schéma multidimensionnel de l'entrepôt au schéma de cette source globale (nommé schéma XML global) afin de récupérer les données nécessaires à la construction. Les correspondances sont définies sous la forme de requêtes XML-QL exprimées en termes du schéma XML global. L'intégrateur stocke les dimensions et les cubes qu'il construit dans un cache multidimensionnel, une collection de fichiers temporaires. L'outil d'administration accepte aussi des expressions WCL (*Warehouse Construction Language*), un langage simple qui autorise l'administrateur à définir les correspondances et à signaler le début de la construction.

Le sous-système de médiation est composé par un médiateur et un ensemble d'adaptateurs XML. Chacun des adaptateurs est associé à une source particulière. Un adaptateur XML permet d'avoir une vue logique en XML de la source relationnelle à laquelle il est associé. Il fournit un schéma XML (sous forme de DTD) de la source concernée. Il accepte des requêtes XML-QL exprimées en termes du schéma XML fourni. Il traduit les requêtes XML-QL en requêtes SQL qui sont envoyées à la source relationnelle pour être exécutées. Il récupère ensuite les résultats relationnels et les transforme en données XML.

Le médiateur fournit le schéma XML global. Ce schéma est construit comme étant l'union des schémas XML des sources [4]. Le médiateur accepte des requêtes XML-QL exprimées en termes du schéma XML global et les évalue. Il décompose une requête globale en plusieurs sous-requêtes, une requête par source concernée, et envoie ces sous-requêtes aux adaptateurs XML. Il récupère ensuite les résultats XML partiels en provenance des adaptateurs et les fusionne pour calculer le résultat global.

L'adaptateur d'entrepôt relationnel est le composant qui interagit directement avec l'entrepôt. Il interagit avec le SGBD sous-jacent afin de créer et faire évoluer le schéma relationnel de l'entrepôt. Il interagit également avec le SGBD afin de stocker dans l'entrepôt les données calculées par l'intégrateur qui se trouvent dans le cache multidimensionnel.

4. Notons que cette solution est très limitée. Dans le cas général, le schéma XML global est construit comme un ensemble de vues définies en termes des schémas XML des sources. Ces vues sont exprimées en utilisant un langage de définition de vues déclaratif basé sur un langage de requêtes de type XML-QL. Nous n'avons pas abordé ces aspects dans ce travail, mais il est clair qu'il s'agit d'une extension importante de notre prototype.

5.3 L'adaptateur XML pour sources relationnelles

Dans cette section, nous détaillons l'implantation d'un adaptateur XML qui permet de voir la source relationnelle associée comme étant une source de données XML. Supposons par exemple que nous avons une relation `pjaunes` qui stocke des informations (adresse, numéro de téléphone et ville de localisation) sur un ensemble de magasins. La partie gauche de la figure 5.5 montre le schéma de cette relation. En utilisant l'adaptateur XML, elle peut être vue comme un document XML conforme à la DTD montrée par la partie droite de la figure 5.5. Nous présentons dans la suite l'architecture interne de l'adaptateur et ensuite nous détaillons chacun de ces composants.

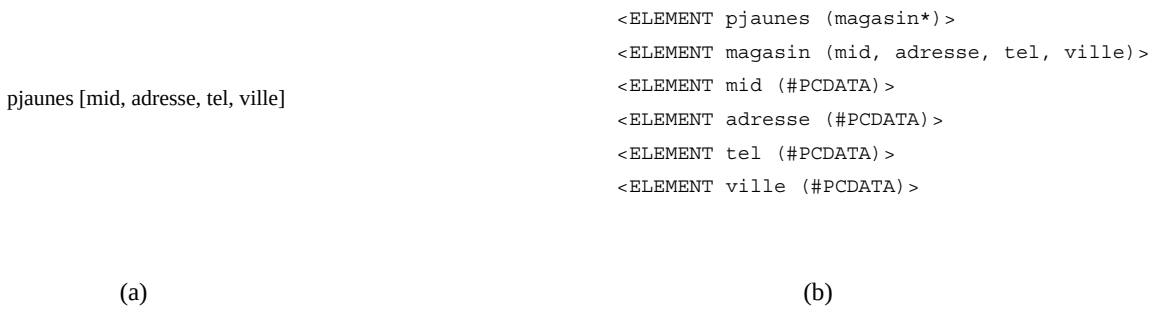


FIG. 5.5 – *(a) Schéma de la relation* `pjaunes`*. (b) Schéma XML exporté*

5.3.1 Architecture interne de l'adaptateur XML implanté

La figure 5.6 montre l'architecture interne de l'adaptateur implanté. L'adaptateur est composé d'un traducteur de requêtes qui traduit les requêtes XML-QL dans SQL. Une requête SQL est ensuite envoyée au composant d'accès à la source, qui l'adresse à la source et récupère les résultats correspondants. Enfin, ces résultats sont traduits par le traducteur de données dans une représentation XML. Dans la suite, nous présentons chacun de ces composants.

5.3.2 Le traducteur de requêtes

La fonction du traducteur de requêtes est de traduire une requête q_x exprimée en XML-QL en une requête équivalente q_s exprimée en SQL. La requête XML-QL en entrée est d'abord analysée pour vérifier si elle est une expression valide du langage.

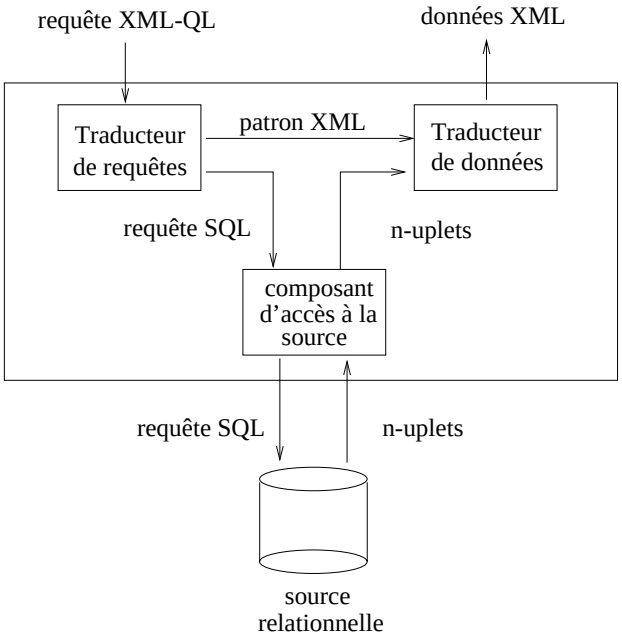


FIG. 5.6 – *Architecture de l'adaptateur XML*

Si c'est le cas, le traducteur effectue ensuite la traduction en SQL. Les expressions non valides sont rejetées.

La vérification de la validité de la requête q_x est réalisée par un parseur XML-QL. Il assure les vérifications lexicographique et syntaxique de la requête. Le résultat de la phase de vérification est une représentation de q_x sous la forme d'un arbre en mémoire. Cette représentation est ensuite exploitée lors de la phase de traduction.

Les règles de traduction d'une requête XML-QL q_x en une requête SQL q_s sont les suivantes :

- Le nom R de l'élément racine de la partie `where` de la requête q_x correspond au nom d'une relation et est donc placé dans la partie `from` de la requête SQL.

- Une variable associée à un attribut d'un élément ou à un élément correspond à une projection sur un attribut de R correspondant au nom de l'élément. Les éléments intermédiaires sont ignorés. Si des éléments appartiennent à deux relations différentes, ils doivent correspondre alors à deux conditions différentes dans la partie `where` de la requête XML-QL.

- Les constantes, les prédicats et les équijointures apparaissant en q_x forment la partie `where` de la requête SQL q_s.

$q_x =$

```
where
   <pjaunes>
       <magasin>
          <ville> $v </>
       </>
    </> in spjaunes
construct
    <ville> $v </>
```

$q_s =$

```
select rel1.ville
from pjaunes rel1
```

FIG. 5.7 – *Exemple de requête SQL générée*

Supposons que la requête XML-QL q_x en entrée à l'adaptateur est celle montrée dans la partie gauche de la figure 5.7. Elle récupère les villes de localisation des magasins. Dans ce cas, le nom de l'élément racine de la partie `where` de q_x est `pjaunes`, correspondant au nom de la relation à interroger. La variable `$v` est associée à l'élément de nom `ville`, correspondant alors à l'attribut sur lequel on fait la projection de la relation `pjaunes`. La partie droite de la figure 5.7 montre la requête SQL q_s résultat de la traduction.

5.3.3 Le composant d'accès à la source

Le composant d'accès à la source (CAS) a deux fonctions principales. La première est l'acheminement de la requête SQL générée par le traducteur de requêtes vers la source relationnelle pour son exécution. La deuxième fonction est la récupération du résultat de l'exécution de la requête.

L'interaction entre ce composant et la source relationnelle est implantée suivant un modèle client/serveur. Le CAS est en effet un client JDBC[5] de la source. Le CAS crée une connexion JDBC à la source pour chaque demande d'exécution d'une requête et la ferme à la fin de cette exécution. Une fois que la connexion a été établie, le CAS envoie la requête SQL à la source pour son exécution et reste en attente des résultats. Il récupère le résultat de l'exécution de la requête, un ensemble de n-uplets, une fois que celui-ci est disponible. Ce résultat est ensuite mise à la disposition du traducteur de données.

5. L'interface JDBC (*Java DataBase Connectivity*) est un ensemble de classes Java permettant de développer des composants et/ou des applications capables de se connecter à des serveurs de bases de données relationnels.

5.3.4 Le traducteur de données

La fonction du traducteur de données est de représenter les données relationnelles récupérées par le composant d'accès à la source sous un format XML. Ce format est déterminé par le patron XML dans la partie `construct` de la requête XML-QL donnée en entrée à l'adaptateur. Le traducteur implante le processus de traduction suivant :

1. Il récupère le patron XML à partir de la représentation interne de la requête.
2. Pour chaque n-uplet, il met en correspondance ses attributs et les éléments XML présents dans le patron, afin d'attribuer les valeurs correctes aux variables.
3. Il ajoute enfin une balise de début (`<Result>`) et une balise de fin (`</Result>`) pour assurer que le résultat XML soit un document bien formé.

Il est à noter que le traducteur de données renvoie le résultat XML sous forme d'une chaîne de caractères. Ainsi, le résultat peut être placé dans un fichier ou exploité directement.

5.4 Médiateur XML

La fonction du médiateur XML est de fournir une vision unifiée des différentes sources de données. Il fournit un schéma XML global. Le médiateur accepte des requêtes XML-QL exprimées en termes de ce schéma et les évalue. Supposons par exemple que les données nécessaires à la construction de l'entrepôt de la chaîne de magasins se trouvent dans cinq sources. La source `scatalogue` fournit des informations sur les produits : code, nom, catégorie, rayon. La source `spjaunes` fournit des informations sur chacun des magasins, telles que son numéro d'identification, son adresse et son numéro de téléphone. La source `sgeog` fournit des données géographiques. Finalement, les sources `sgrenoble`, `sparis` et `slondres` décrivent les ventes journalières réalisés par les magasins situés à Grenoble, Paris et Londres, respectivement. Dans cet exemple, le médiateur fournit le schéma XML global montré dans la figure 5.8. Il est possible de poser des requêtes sur ce schéma pour connaître par exemple les régions où les magasins de la chaîne sont localisés.

Nous présentons dans la suite l'architecture interne du médiateur XML que nous avons implanté. Nous détaillons ensuite chacun de ses composants.

```
<ELEMENT catalogue (produit*)>
<ELEMENT produit (pid, nom, cat, rayon)>
<ELEMENT pid (#PCDATA)>
<ELEMENT nom (#PCDATA)>
<ELEMENT cat (#PCDATA)>
<ELEMENT rayon (#PCDATA)>
```

catalogue@scatalogue

```
<ELEMENT pjaunes (magasin*)>
<ELEMENT magasin (mid, adresse, tel, ville)>
<ELEMENT mid (#PCDATA)>
<ELEMENT adresse (#PCDATA)>
<ELEMENT tel (#PCDATA)>
<ELEMENT ville (#PCDATA)>
```

pjaunes@spjaunes

```
<ELEMENT geog (pays*)>
<ELEMENT pays (nomp, region*)>
<ELEMENT region (nomr, ville*)>
<ELEMENT nomp (#PCDATA)>
<ELEMENT nomr (#PCDATA)>
<ELEMENT ville (#PCDATA)>
```

geog@sgeog

```
<ELEMENT ventes (vente*)>
<ELEMENT vente (pid, q, date)>
<ELEMENT pid (#PCDATA)>
<ELEMENT unites (#PCDATA)>
<ELEMENT date (#PCDATA)>
```

ventes@sgrenoble

FIG. 5.8 – *Schéma XML global*

5.4.1 Architecture interne du médiateur XML implanté

L'architecture interne du médiateur que nous avons implanté est montrée dans la figure 5.9. La requête XML-QL globale en entrée est en premier lieu analysée par le *parseur de requêtes*, afin de vérifier si elle est une expression valide du langage. Il construit ensuite une représentation interne pour faciliter sa manipulation. Le *décomposeur de requêtes* analyse cette représentation afin de générer un ensemble de sous-requêtes, une par source de l'entrepôt concernée par la requête XML-QL globale. Chacune de ces sous-requêtes est alors envoyée par le *distributeur de requêtes* à l'adaptateur associé à la source sur laquelle porte son calcul. Le distributeur récupère les résultats XML produits par les différents adaptateurs. Ces résultats temporaires sont alors fusionnés par le constructeur de résultats dans un unique résultat XML global. Nous détaillons dans la suite chacun de ces composants.

5.4.2 Le parseur XML-QL

La fonction principale du parseur XML-QL est la vérification de la validité de la requête globale XML-QL en entrée du médiateur. Cette analyse vise à vérifier la lexicographie et la syntaxe de la requête. Si aucune erreur n'est détectée lors de cette analyse, le résultat du parseur est une représentation interne de la requête pour

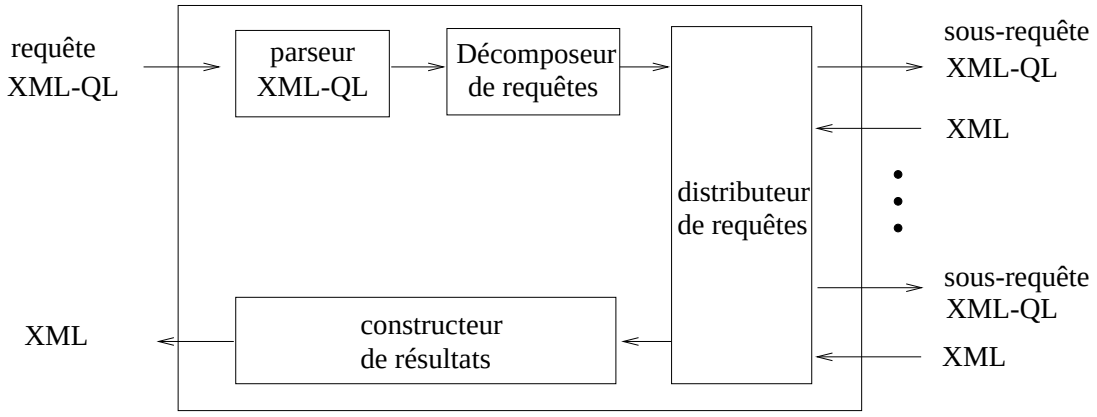


FIG. 5.9 – *Architecture du médiateur de données XML*

faciliter sa manipulation. La représentation est en effet un arbre en mémoire. Pour la construction du parseur XML-QL, nous nous sommes appuyés sur l'outil de génération de parseurs JavaCC.

5.4.3 Le décomposeur de requêtes

Le décomposeur de requêtes décompose une requête globale q_g en un ensemble Q de sous-requêtes. Rappelons qu'une requête globale porte en général sur plusieurs sources. Il existe une sous-requête q dans Q pour chacune des sources référencées dans la requête globale q_g. Une sous-requête q sur une source s est générée de la manière suivante :

- La partie `where` de q est formé par (i) la condition de sélection sur s qui apparaît dans la requête globale q_g et (ii) les prédicats ne contenant que des variables déclarées dans la condition de sélection du point (i)[6].

- La partie `construct` de q est une copie de la condition de sélection de la partie `where` de q. Nous avons établi cette convention pour faciliter la fusion des résultats partiels.

Supposons que la requête en entrée au médiateur est celle de la figure 5.2. Rappelons que cette requête récupère, pour chacune des villes stockées dans la source

6. Les prédicats comparant deux variables présentes dans des sources différentes ne sont pris en compte que lors de la fusion des résultats.

$q_1 =$

```
where
  <pjaunes>
    <magasin>
      <ville> $v </>
    </>
  </>IN spjaunes,
construct
  <pjaunes>
    <magasin>
      <ville> $v </>
    </>
  </>
```

$q_2 =$

```
where
  <geog>
    <pays>
      <region>
        <nomr> $r </>
        <ville> $v </>
      </>
    </>
  </>IN sgeog
construct
  <geog>
    <pays>
      <region>
        <nomr> $r </>
        <ville> $v </>
      </>
    </>
  </>
```

FIG. 5.10 – *Sous-requêtes générées à partir de la requête de la figure 5.2*

`spjaunes`, sa région auprès de la source `sgeog`. La décomposition de cette requête donne comme résultat les deux sous-requêtes q_1 et q_2 montrées dans la figure 5.10. La requête q_1 récupère les villes de la source `spjaunes` et la requête q_2 récupère toutes les paires (ville,région) de la source `sgeog`.

5.4.4 Le distributeur de requêtes

Le distributeur de requêtes a deux fonctions principales. La première consiste à envoyer pour exécution chacune des sous-requêtes générées par le décomposeur à l'adaptateur correspondant. La deuxième fonction du distributeur est de récupérer les résultats partiels (sous la forme de fichiers de données XML) produits par les différents adaptateurs.

Considérons par exemple la figure 5.11. Soient q_1 et q_2 les sous-requêtes montrées par la figure 5.10. Le distributeur de requêtes envoie la requête q_1 à l'adaptateur A_1 associé à la source `spjaunes`. Il envoie aussi la requête q_2 à l'adaptateur A_2 de la source `sgeog`. Les adaptateurs évaluent leur requêtes respectives et envoient les résultats XML partiels r_1 et r_2. Enfin, le distributeur récupère ces résultats et les stocke

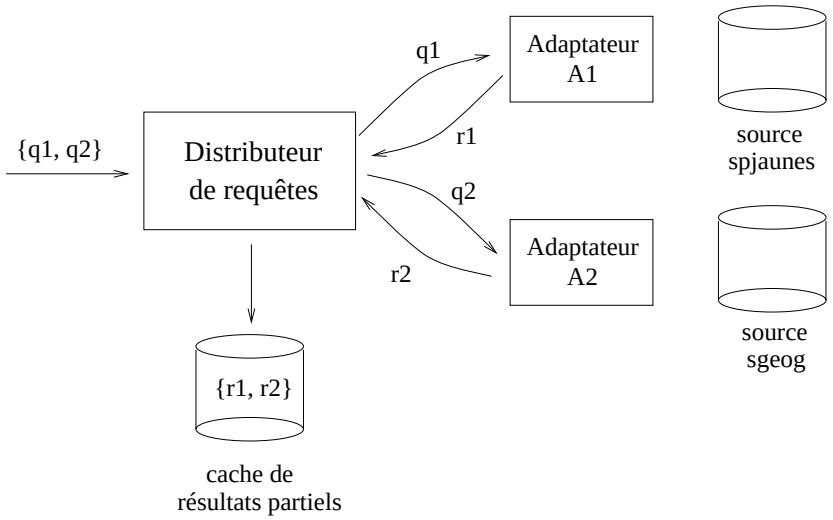


FIG. 5.11 – *Le distributeur de requêtes*

temporairement dans un cache sous la forme de fichiers. Ce cache est à la disposition du constructeur de résultats lors du calcul du résultat global. Nous détaillons dans la suite les activités que le constructeur de résultats réalise.

5.4.5 Le constructeur de résultats

Le constructeur de résultats fusionne les résultats partiels obtenus par le distributeur de requêtes en un résultat global. Pour mettre en œuvre cette fusion, nous avons opté pour la génération et l'exécution des requêtes que nous appelons *de fusion*. Une requête de fusion q_f est en essence une copie de la requête globale q_g donné en entrée au médiateur, restreinte pour ne prendre en compte que les aspects globaux de q_g. La figure 5.12 montre le processus de fusion.

La première phase du processus de fusion est la génération de la requête de fusion à partir de la requête globale. Le constructeur de résultats utilise les règles suivantes pour générer la requête de fusion q_f à partir de la requête globale q_g :

- La partie `where` de q_f est formée par (i) les conditions de sélection qui apparaissent dans la requête globale q_g (le nom de la source est remplacé par le nom du fichier temporaire stockant les résultats partiels en provenance d'un adaptateur) et (i) les prédicats globaux, c'est-à-dire comparant des variables associées à plusieurs sources.

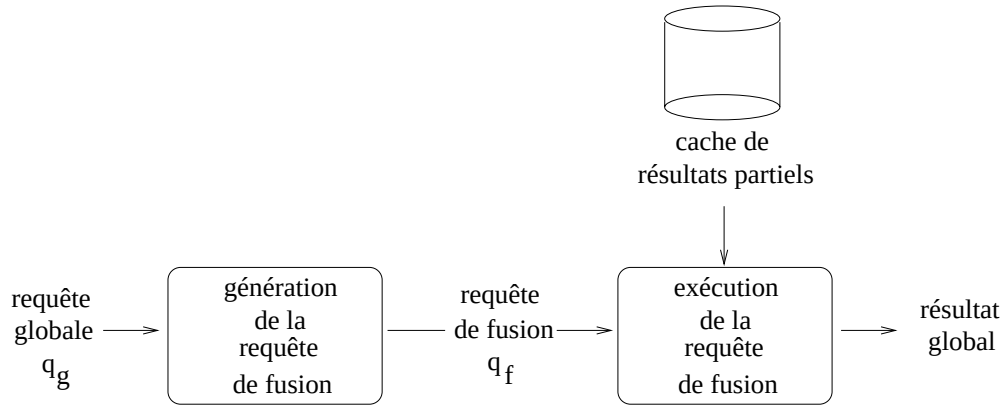


FIG. 5.12 – *Processus de fusion réalisé par le constructeur de résultats*

– La partie `construct` de q_f est une copie de celle de la requête globale q_g.

Supposons que nous voulons générer une requête de fusion q_f à partir de la requête globale g_g montrée dans la partie gauche de la figure 5.13. Dans ce cas, la condition de sélection `<pjaunes>..</> in spjaunes` de q_g est transformée en la condition `<Result><pjaunes>..</></> in 'temp1.xml'`, où `temp1.xml` est le nom du fichier stockant les résultats partiels en provenance de l'adaptateur associé à la source `spjaunes`. Une transformation similaire est subie par la condition `<geog>..</> in sgeog`. La partie droite de la figure 5.13 présente la requête de fusion q_f générée.

La phase finale du processus de fusion de données est l'exécution de la requête q_f générée. Pour cela, le constructeur de résultats fait appel à l'interpréteur XML-QL en lui passant comme paramètre q_f. Lors de l'exécution, l'interpréteur XML-QL récupère du cache les résultats partiels à fusionner. Le résultat de l'exécution de la requête est un fichier XML stockant les données fusionnées.

5.5 L'outil d'administration

L'outil d'administration est composé de deux sous-composants : un interpréteur du langage MDL et un interpréteur du langage WCL. Le premier sous-composant traite les expressions MDL de création et de modification du schéma multidimensionnel. Le deuxième sous-composant traite les expressions WCL pour définir les correspondances et pour signaler le début de la construction de l'entrepôt. Nous nous sommes

q_g =

```
where
  <pjaunes>
    <magasin>
      <ville> $v </>
    </>
  </>IN spjaunes,
  <geog>
    <pays>
      <region>
        <nomr> $r </>
        <ville> $v </>
      </>
    </>
  </> IN sgeog
construct
  <vr>
    <ville> $v </>
    <region> $r </>
  </>
```

q_f =

```
where
  <Result>
    <pjaunes>
      <magasin>
        <ville> $v </>
      </>
    </>
  </>IN 'temp1.xml',
  <Result>
    <geog>
      <pays>
        <region>
          <nomr> $r </>
          <ville> $v </>
        </>
      </>
    </>
  </>IN 'temp2.xml',
construct
  <vr>
    <ville> $v </>
    <region> $r </>
  </>
```

FIG. 5.13 – *Requête globale et requête de fusion*

appuyés sur JavaCC[7], un outil de génération de parseurs, pour la construction de nos interpréteurs. Nous détaillons ces composants dans la suite.

5.5.1 L'interpréteur MDL

L'interpréteur MDL traduit une expression MDL en une représentation interne au gestionnaire. Pour cela, il analyse d'abord l'expression MDL donnée en entrée afin de vérifier sa validité. Une expression MDL est valide si elle est conforme aux règles d'écriture du langage. Par exemple, l'expression MDL suivante décrivant le schéma de la dimension *Magasin* est valide :

```
create dimension Magasin
  level ville : char(40)
  level region: char(40)
  level pays : char(40)
  rollup ville, region
  rollup region, pays
```

7. Disponible à l'adresse `http://www.metamata.com/javacc/index.html`

Le résultat de la phase d'analyse d'une expression MDL valide est un arbre en mémoire. L'interpréteur parcourt ensuite cet arbre afin de récupérer les informations correspondant à l'opération d'évolution demandée. Elles comprennent le nom de l'opération ainsi que les paramètres nécessaires à son exécution correcte. Ces informations sont communiquées ensuite au coordinateur.

5.5.2 L'interpréteur WCL

Ce composant traite les expressions WCL pour définir les correspondances et l'expression `build warehouse` pour signaler le début de la construction de l'entrepôt. Rappelons que dans notre prototype, les correspondances entre le schéma multidimensionnel et le schéma XML global sont exprimées sous la forme de requêtes XML-QL. Dans le cas d'un schéma de dimension, on associe une requête globale (i) au niveau base et (ii) à chacune des relations entre les niveaux du schéma :

- Une expression WCL pour définir la correspondance qui associe le niveau base d'un schéma de dimension au schéma XML global est de la forme `maplevel(D,l,q)`, où D est le nom de la dimension, l est le niveau et q est une requête XML-QL.

- Une expression WCL pour définir la correspondance qui associe la relation d'ordre entre deux niveaux au schéma XML global a la forme `maprup(D,(l,l'),q)`, où D est le nom de la dimension, l et l' sont deux niveaux et q est une requête XML-QL.

La figure 5.14 montre les correspondances associant le schéma de la dimension *Magasin* au schéma XML global de la figure5.8. La requête associée au niveau *ville* récupère les villes de la source `spjaunes`. La requête associée à la relation d'ordre entre les niveaux *ville* et *région* récupère de la source `sgeog` la région d'appartenance d'une ville. La requête associée à la relation d'ordre entre les niveaux *région* et *pays* récupère aussi de la source `sgeog` le pays d'appartenance d'une région.

Dans le cas d'un schéma de cube, une correspondance associe la mesure du cube au schéma XML global. Une expression WCL pour définir une telle correspondance est de la forme `mapmeasure(C,m,q)`, où C est le nom du cube, m est la mesure et q est une requête XML-QL. La figure 5.15 montre la correspondance entre la mesure *quantité* du cube *Ventes* et le schéma XML global de la figure 5.8. La requête XML-QL définie récupère les ventes effectuées par les magasins.

```
maprup ('Magasin', ('ville','region'),
        'where
           <pjaunes>
              <magasin>
                 <ville> $v </>
              </>
           </> in spjaunes
           <geog>
              <pays>
                 <region>
                     <nomr> $r </>
                     <ville> $v </>
                 </>
              </>
           </> in sgeog
        construct
           <rupvr>
              <ville> $v </>
              <region> $r </>
           </>' )
```

```
maprup ('Magasin',('region','pays'),
        'where
           <geog>
              <pays>
                 <nomp> $p </>
                 <region>
                    <nomr> $r </>
                 </>
              </>
           </> in sgeog
        construct
           <ruprp>
              <region> $r </>
              <pays> $p </>
           </>' )
```

```
maplevel ('Magasin', 'ville',
          'where
             <pjaunes>
                <magasin>
                   <ville> $v </>
                </>
             </> in spjaunes
          construct
             <ville> $v </>')
```

FIG. 5.14 – *Les correspondances entre le schéma de la dimension Magasin et le schéma XML global de la figure 5.8.*

5.6 L'intégrateur

L'intégrateur implante le processus d'intégration de données décrit dans le paragraphe 4.3.3. Le but de ce processus est la création d'une instance du schéma multidimensionnel de l'entrepôt. L'intégrateur construit d'abord les instances des schémas de dimension et ensuite les instances des schémas de cube. Les données nécessaires à la création de ces instances sont extraites des sources grâce aux correspondances qui associent les schémas de dimension et les schémas de cube au schéma XML global. Une fois construits, les dimensions et les cubes sont stockés dans un cache. Ce cache prend la forme d'une collection de fichiers XML temporaires que l'intégrateur supprime une fois que les dimensions et les cubes sont stockés de manière définitive dans l'entrepôt.

Pour construire une dimension, l'intégrateur calcule les membres de tous les niveaux du schéma de la dimension et forme avec eux la hiérarchie d'agrégation. L'intégrateur récupère d'abord les membres du niveau base du schéma de la dimension. Pour cela, l'intégrateur demande au médiateur XML d'évaluer la requête XML-QL

```
mapmeasure ('Ventes','quantité'),
'where                                  union   where                                  union   where
  <ventes>                                        <ventes>                                        <ventes>
    <vente>                                         <vente>                                         <vente>
      <pid> $c1 </>                                   <pid> $c2 </>                                   <pid> $c3 </>
      <unites> $q1 </>                                <unites> $q2 </>                                <unites> $q3 </>
      <date> $j1 </>                                  <date> $j2 </>                                  <date> $j3 </>
    </>                                             </>                                             </>
  </> in sgrenoble                                </> in sparis                                   </> in slondres
construct                                       construct                                       construct
  <vente>                                         <vente>                                         <vente>
    <ville> 'Grenoble' </>                          <ville> 'Paris' </>                             <ville> 'Londres' </>
    <code> $c1 </>                                  <code> $c2 </>                                  <code> $c3 </>
    <jour> $j1 </>                                  <jour> $j2 </>                                  <jour> $j3 </>
    <quantite> $q1 </>                              <quantite> $q2 </>                              <quantite> $q3 </>
  </>                                             </>                                             </>' )
```

FIG. 5.15 – *La correspondance entre le schéma du cube Ventes et le schéma XML global.*

associée à ce niveau. Pour chaque membre m récupéré, il cherche ensuite à déterminer le membre m' du niveau immédiatement supérieur l' auquel le membre m est associé. Pour cela, l'intégrateur demande au médiateur d'évaluer la requête associée à la relation d'ordre entre le niveau base et le niveau l'. L'intégrateur cherche ensuite à déterminer pour chaque membre m' du niveau l' le membre m'' du niveau l'' (supérieur à l') auquel m' est associé. Ce processus continue jusqu'à atteindre le niveau $\top$ du schéma de la dimension. L'intégrateur associe un identificateur unique à chacun des membres de la dimension.

```
<toplevel>
  <pays mid='7' value='France'>
     <region mid='4' value='Rhone-Alpes'>
        <ville mid='1' value='Grenoble'/>
     </>
     <region mid='5' value='Ile-de-France'>
        <ville mid='2' value='Paris'/>
     </>
  </>
  <pays mid='8' value='Angleterre'>
     <region mid='6' value='South East'>
       <ville mid='3' value='Londres'/>
     </>
  </>
</>
```

FIG. 5.16 – *Représentation XML de la dimension Magasin*

Considérons à titre d'illustration la construction de la dimension *Magasin*. Dans ce cas, l'intégrateur demande d'abord au médiateur d'évaluer la requête XML-QL

associée au niveau *ville* pour récupérer toutes les villes. Ensuite, pour chaque ville, il cherche la région à laquelle appartient la ville. Pour cela, il demande au médiateur d'évaluer la requête associée à la relation d'ordre entre les niveaux *ville* et *région*. L'intégrateur cherche alors à déterminer, pour chaque région, le pays auquel appartient la région. L'intégrateur demande au médiateur d'évaluer la requête associée à la relation d'ordre entre les niveaux *région* et *pays*. Finalement, les membres du niveau *pays* sont associés au membre unique `t` du niveau $\top$. La figure 5.16 montre la dimension *Magasin* dans le format XML sous lequel elle est stockée dans le cache.

Pour construire un cube, l'intégrateur calcule les cellules qui le forment. Pour cela, l'intégrateur demande au médiateur d'évaluer la requête associée à la mesure du cube. Dans le cas du cube *Ventes*, par exemple, la requête à évaluer est celle associée à la mesure *quantité* (cf. figure 5.15). L'intégrateur récupère ensuite le résultat de l'évaluation de la requête (un ensemble de cellules) et le stocke dans le cache.

5.7 Adaptateur d'entrepôt relationnel

L'adaptateur d'entrepôt relationnel est le composant qui interagit directement avec l'entrepôt. Il est composé de deux sous-composants : le traducteur de schémas multidimensionnels et le traducteur de données multidimensionnelles. Le premier sous-composant interagit avec le SGBD sous-jacent afin de créer et modifier le schéma relationnel en étoile de l'entrepôt. Le deuxième sous-composant interagit également avec le SGBD sous-jacent afin de stocker dans l'entrepôt les données calculées par l'intégrateur. Nous détaillons ces sous-composants dans la suite.

5.7.1 Le traducteur de schémas multidimensionnels

Le traducteur de schémas multidimensionnels (TSM) implante les règles pour traduire le schéma d'une dimension ou d'un cube en un schéma de relation (cf. section 4.3). Le résultat de la traduction est une expression SQL de la forme `create table r (`a_1 t_1`, ...,` a_n t_n`)`, où `r` est le nom d'une relation et les attributs a_1 à a_n sont des types t_1 à t_n, respectivement. Nous détaillons dans la suite les traductions des schémas de dimension et des schémas de cube.

Nous avons vu précédemment les règles de traduction d'un schéma de dimension en un schéma de relation. Ce règles indiquent que (i) le nom de la relation est le nom de la dimension, (ii) il existe un attribut l pour chacun des niveaux l de la dimension

et (iii) il existe un attribut de la forme id_l_1, où l_1 est le niveau base de la dimension. Le résultat de la traduction est alors une expression SQL de la forme

$$\texttt{create table r (}id\_l_1\ \texttt{int,}\ l_1\ t_1 \ldots,\ l_n\ t_n\texttt{)},$$

où `r` est le nom de la dimension et les attributs l_1 à l_n correspondent aux niveaux de la dimension ; les types de ces attributs sont t_1 à t_n, respectivement. Notons que l'attribut id_l_1 est de type entier. Par exemple, l'expression SQL générée à partir du schéma de la dimension *Magasin* est :

```
create table Magasin (
      id_ville int,
      ville char(40),
      region char(40),
      pays char(40)
).
```

Le TSM traduit également un schéma de cube en un schéma de relation. Dans ce cas, les règles de traduction indiquent que (i) le nom de la relation est le nom du cube, (ii) il existe un attribut de la forme id_a pour chaque axe a du schéma de cube et (iii) il existe un attribut m pour chacune des mesures m du cube. Le résultat de la traduction est une expression SQL de la forme

$$\texttt{create table r (}id\_a_1\ \texttt{int, ...,}\ id\_a_i\ \texttt{int,}\ m_1\ t_1\texttt{, ...,}\ m_j\ t_j\texttt{)},$$

où `r` est le nom du cube et les attributs id_a_1 à id_a_i sont de type entier et les attributs m_1 à m_j sont des types t_1 à t_j, respectivement. Par exemple, l'expression SQL générée à partir du schéma du cube *Ventes* est :

```
create table Ventes (
      id_code int,
      id_ville int,
      id_jour int,
      quantite int
).
```

Le TSM interagit également avec le SGBD afin de modifier le schéma en étoile de l'entrepôt. Ces modifications comprennent le changement de nom d'une relation,

la suppression d'une relation ou l'ajout d'attributs à une relation (cf. section 4.4). Par exemple, lorsque le nom d'une dimension ou d'un cube est modifié, ce composant génère une expression SQL de la forme `rename table r to r'`, où `r` est l'ancien nom et `r'` est le nouveau nom. Lorsque un schéma d'une dimension ou d'un cube est supprimé, ce composant génère une expression de la forme `drop table r`, où r est le nom de la dimension ou du cube supprimé.

Les expressions SQL générées sont envoyées au SGBD relationnel pour être exécutées. L'interaction entre le traducteur de schémas multidimensionnels et le SGBD relationnel est implantée suivant un modèle client/serveur. Le TSM est en effet un client JDBC du serveur relationnel de l'entrepôt. Le TSM crée une connexion JDBC au serveur pour chaque demande d'exécution d'une expression SQL générée et ferme la connexion à la fin de cette exécution.

5.7.2 Le traducteur de données multidimensionnelles

Le traducteur de données multidimensionnelles (TDM) interagit avec le SGBD afin de stocker dans l'entrepôt les dimensions et les cubes construits par l'intégrateur. Il implante pour cela les règles de traduction de données multidimensionnels en n-uplets présentées dans le paragraphe 4.3. Le résultat de la traduction est un ensemble d'expressions SQL de la forme `insert into r (<liste_attributs>) values <liste_valeurs>)`, où `r` est le nom d'une relation, `liste_attributs` est la liste d'attributs de la relation `r` et `liste_valeurs` sont les valeurs que prennent ces attributs.

Considérons d'abord la génération d'expressions SQL à partir d'une dimension. Nous avons précédemment vu qu'une dimension est stockée dans le cache multidimensionnel sous la forme d'un document XML. Le TDM génère les expressions SQL à partir de l'arbre DOM de ce document. Une expression SQL est générée pour chaque chemin allant du nœud racine de l'arbre jusqu'à un nœud feuille. Un chemin est une séquence de nœuds n_0, n_1, ... , n_k où n_0 est le nœud racine de l'arbre, n_k est un nœud feuille et $\forall_{i=0,\ldots,k-1}$ le nœud n_{i+1} est un fils du nœud n_i. Un chemin n_0, n_1, ... , n_k se trouve alors à l'origine d'une expression de la forme

`insert into r (`l_1`, ... , `l_k`, `id_l_k`) values (`v_1`, ... , `v_k`, `v`)`

où r est le nom de la dimension, $\forall_{i=1,\ldots,k}\ l_i$ est le nom d'élément que le nœud n_i contient, $\forall_{i=1,\ldots,k}\ v_i$ est la valeur de l'attribut `value` du nœud n_i et v est la valeur de l'attribut `mid` du nœud n_k.

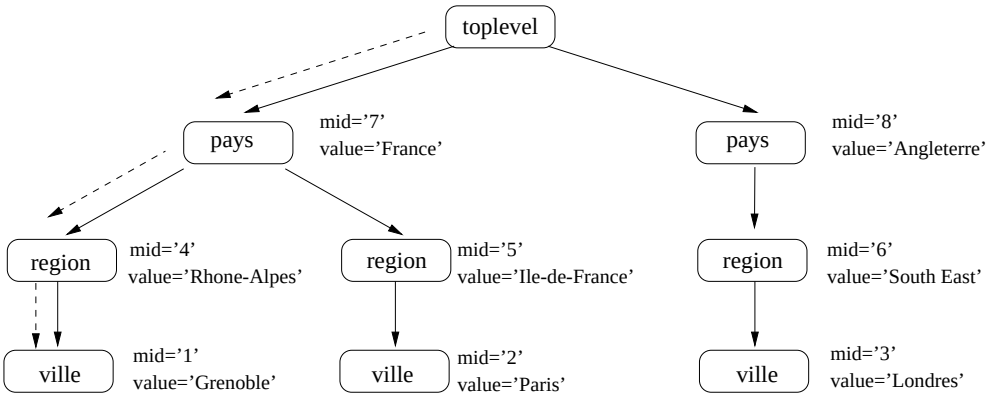


FIG. 5.17 – *L'arbre DOM du document XML stockant la dimension Magasin.*

La figure 5.17 présente l'arbre DOM du document XML qui stocke la dimension *Magasin* (cf. figure 5.16). Le TDM génère, à partir de cet arbre, des expressions SQL de la forme `insert into Magasin (pays, region, ville, id_ville) values (p,r,v,id)`, où `p`, `r` et `v` sont des chaînes de caractères représentant respectivement un nom de ville, de région et de pays et `id` est un entier représentant l'identificateur d'une ville. Par exemple, l'expression SQL générée à partir du chemin indiqué dans la figure 5.17 est `insert into Magasin (pays, region, ville, id_ville) values ('France', 'Rhone-Alpes', 'Grenoble', 1)`.

Les expressions SQL générées sont envoyées au SGBD relationnel pour être exécutées. L'interaction entre le traducteur de données multidimensionnelles et le SGBD relationnel est implantée suivant un modèle client/serveur, étant le TDM un client JDBC du serveur relationnel. Le TDM crée une connexion JDBC au serveur relationnel au début du processus de traduction des données et ferme la connexion à la fin.

5.8 Le référentiel

Le référentiel effectue le stockage et la récupération des informations utilisées par le gestionnaire d'entrepôt. Ces informations comprennent le schéma multidimensionnel de l'entrepôt et les correspondances entre ce schéma et le schéma XML global. Ces informations sont stockées sous la forme de fichiers XML. La génération dynamique et la manipulation de ces fichiers passe par l'utilisation de l'interface de programmation DOM (cf. paragraphe 5.1.3). Nous détaillons dans la suite la représentation XML

sous laquelle les schémas de dimension et les schémas de cube sont stockés. Nous présentons ensuite le format XML sous lequel les correspondances sont stockées.

```
<!ELEMENT dimension (name,level+,rollup+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT level (#PCDATA)>
<!ATTLIST level lid ID #REQUIRED
                type CDATA #REQUIRED>
<!ELEMENT rollup EMPTY>
<!ATTLIST rollup origin IDREF #REQUIRED
                 target IDREF #REQUIRED>
```

```
<dimension>
  <name> Magasin </>
  <level lid='1' type='char(40)'> ville </>
  <level lid='2' type='char(40)'> region </>
  <level lid='3' type='char(40)'> pays </>
  <level lid='4' type='none'> All </>
  <rollup origin='1' target='2' />
  <rollup origin='2' target='3' />
  <rollup origin='3' target='4' />
</>
```

FIG. 5.18 – *Représentation XML d'un schéma de dimension*

Un schéma de dimension est stocké sous la forme d'un document XML conforme à la DTD montrée dans la partie gauche de la figure 5.18. Cette DTD indique qu'un élément `dimension` est composé d'un élément `name`, le nom de la dimension, un ou plusieurs éléments `level`, les niveaux de la dimension, et un ou plusieurs éléments `rollup`, les relations d'ordre entre les niveaux. Un élément `level` a comme attributs un identificateur (`lid`) et un type (`type`). Un élément `rollup` a deux attributs : un attribut `origin` et un attribut `target`, représentant respectivement le niveau source et le niveau cible d'une relation d'ordre. La partie droite de la figure 5.18 montre le schéma de la dimension *Magasin* représenté sous format XML.

```
<!ELEMENT cube (name, axis+, measure+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT axis (#PCDATA)>
<!ELEMENT measure (#PCDATA)>
<!ATTLIST measure type CDATA #REQUIRED>
```

```
<cube>
  <name> ventes </>
  <axis> code </>
  <axis> ville </>
  <axis> jour </>
  <measure type='integer'> quantité </>
</cube>
```

FIG. 5.19 – *Représentation XML d'un schéma de cube*

Un schéma de cube est aussi stocké sous la forme d'un document XML. Dans ce cas, le document est conforme à la DTD montrée dans la partie gauche de la figure 5.19. Cette DTD spécifie qu'un élément `cube` est composé d'un élément `name`, le nom du cube, un ou plusieurs éléments `axis`, un pour chaque axe du cube, et un ou plusieurs éléments `measure`, un pour chacune des mesures. La partie droite de la figure 5.19 montre la représentation XML du schéma du cube *Ventes*.

Une correspondance entre un niveau et le schéma XML global est stockée sous la forme d'un document XML conforme à la DTD montrée par la partie gauche de

```
<!ELEMENT maplevel (dimension, level, query)>
<!ELEMENT dimension (#PCDATA)>
<!ELEMENT level (#PCDATA)>
<!ELEMENT query (#PCDATA)>
```

```
<maplevel>
   <dimension> Magasin </>
   <level> ville </>
   <query>
       <![ CDATA [ where
                      <pjaunes>
                         <magasin>
                            <ville> $v </>
                         </>
                      </> in spjaunes
                   construct
                      <ville> $v </>
                ]
       ]>
   </>
</>
```

FIG. 5.20 – *Représentation XML d'une correspondance*

la figure 5.20. Cette DTD indique que l'élément `maplevel` est composé d'un élément `dimension`, d'un élément `level` et d'un élément `query`, représentant respectivement le nom d'une dimension, un niveau et une requête XML-QL. La partie droite de la figure 5.20 montre le format XML de la correspondance entre le niveau *ville* de la dimension *Magasin* et le schéma XML global.

Les correspondances associant les relations d'ordre d'une dimension et les mesures d'un cube au schéma XML global sont également stockées sous un format XML. Dans les deux cas, les représentations XML choisies sont de légères variations de celle utilisée pour stocker une correspondance entre un niveau d'une dimension et le schéma XML global.

5.9 Expérimentation

L'objectif de nos expériences a été de montrer l'utilisation de notre prototype pour construire et faire évoluer un entrepôt de données relationnelles. Nous avons choisi une application typique : les ventes des produits dans une chaîne de magasins.

La figure 5.21 montre le schéma multidimensionnel de notre application. Il est formé par les schémas des dimensions *Produit*, *Magasin* et *Temps*, et par le schéma du cube *Ventes* :

– La schéma de la dimension *Produit* a trois niveaux : *code*, *catégorie* et *rayon*. Le niveau *code* représente le code des produits proposés par les magasins de la chaîne, le niveau *catégorie* représente les catégories dans lesquelles les produits sont classés et le niveau *rayon* regroupe les catégories par rayon.

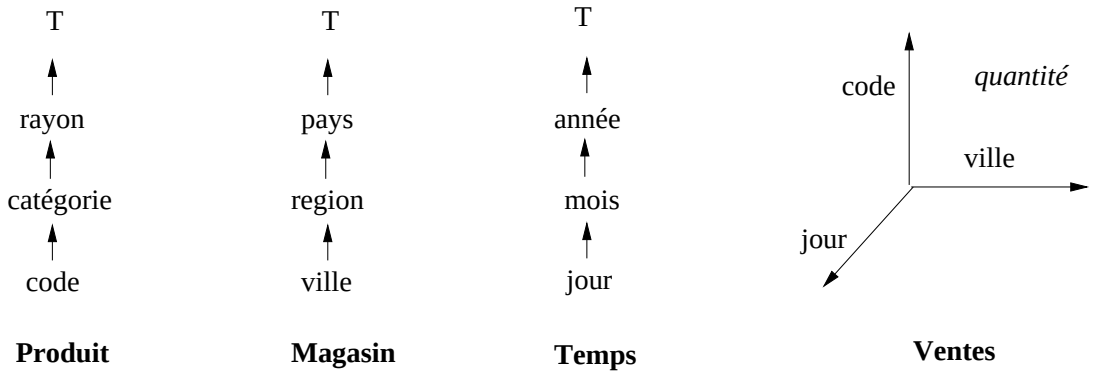


FIG. 5.21 – *Schéma multidimensionnel de l'application exemple*

- Le schéma de la dimension *Magasin* a trois niveaux : *ville*, *région* et *pays*. Le niveau *ville* représente les villes où les magasins sont localisés, le niveau *région* regroupe les villes et le niveau *pays* regroupe les régions.
- Le schéma de la dimension *Temps* a trois niveaux : *jour*, *mois* et *année*.
- Le schéma du cube *Ventes* a comme mesure *quantité* et il est déterminé par les axes *code* (niveau de la dimension *Produit*), *ville* (niveau de la dimension *Magasin*) et *jour* (niveau de la dimension *Temps*).

Nous décrivons dans la suite les tests que nous avons réalisés : création du schéma, construction et évolution du schéma.

5.9.1 Création du schéma de l'entrepôt

Nous avons utilisé le langage MDL pour décrire le schéma multidimensionnel de notre entrepôt. La figure 5.22 montre les expressions MDL utilisées. Chacune de ces expressions est traitée par l'interpréteur MDL afin de vérifier sa validité syntaxique. Une fois que cette vérification est effectuée, les informations sur l'opération à réaliser et les paramètres qu'elle requiert sont envoyés au coordinateur.

Le coordinateur vérifie la validité des schémas de dimension et du schéma de cube. Lorsque un schéma de dimension ou de cube est validé, le coordinateur demande au référentiel de stocker celui-ci pour une utilisation postérieure. Par exemple, la figure 5.23 montre la représentation XML sous laquelle le schéma de la dimension *Produit* est stocké.

```
create dimension produit
level code: char(10)
level categorie: char(40)
level rayon: char(40)
rollup code, categorie
rollup categorie, rayon
```

```
create dimension magasin
level ville: char(40)
level region: char(40)
level pays: char(40)
rollup ville, region
rollup region, pays
```

```
create dimension temps
level jour: date
level mois: char(10)
level année: char(4)
rollup jour, mois
rollup mois, année
```

```
create cube ventes
axis code
axis ville
axis jour
mesure quantité : integer
```

FIG. 5.22 – *Les expressions MDL utilisées*

```
<dimension>
  <name> Produit </>
  <level lid='1' type='char(10)'> code </>
  <level lid='2' type='char(40)'> categorie </>
  <level lid='3' type='char(40)'> rayon </>
  <level lid='4' type='none'> All </>
  <rollup origin='1' target='2' />
  <rollup origin='2' target='3' />
  <rollup origin='3' target='4' />
</>
```

FIG. 5.23 – *Représentation XML du schéma de la dimension Produit*

Une fois que le référentiel a stocké le schéma de dimension ou le schéma de cube, le coordinateur demande à l'adaptateur d'entrepôt de générer, à partir du schéma de la dimension ou du cube, l'expression SQL pour créer la relation correspondante dans l'entrepôt. Par exemple, l'expression suivante est dérivée à partir du schéma de la dimension *Produit* :

```
create table Produit (
      id_code int,
      code char(10),
      categorie char(40),
      rayon char(40)
).
```

Les expressions ainsi générées sont envoyées ensuite au SGBDR sous-jacent pour leur exécution. Dans notre exemple, le résultat est la création de la nouvelle relation `Produit` dans l'entrepôt.

5.9.2 Construction de l'entrepôt

Notre entrepôt est construit à partir du schéma XML global montré par la figure 5.8. Rappelons que cette vue intègre cinq sources. La source `scatalogue` fournit des informations sur les produits : code, nom, catégorie, rayon. La source `spjaunes` fournit des informations sur chacun des magasins, telles que son numéro d'identification, son adresse et son numéro de téléphone. La source `sgeog` fournit des données géographiques. Une source décrit les ventes journalières réalisés par chacun des magasins localisés à Grenoble, Paris et Londres.

```
maprup('Produit', ('code','categorie'),
        'where
           <catalogue>
              <produit>
                 <pid> $code </>
                 <cat> $cat </>
              </>
           </> in scatalogue
        construct
           <rupcc>
              <code> $code </>
              <categorie> $cat </>
           </>')

maprup('Produit', ('categorie','rayon'),
        'where
           <catalogue>
              <produit>
                 <cat> $cat </>
                 <rayon> $r </>
              </>
           </> in scatalogue
        construct
           <rupcr>
              <categorie> $cat </>
              <rayon> $r </>
           </>')

maplevel('Produit', 'code', 'where
                               <catalogue>
                                  <produit>
                                     <pid> $c </>
                                  </>
                               </> in scatalogue
                            construct
                               <code> $c </>')
```

FIG. 5.24 – *Les correspondances entre le schéma de la Produit et le schéma XML global*

La figure 5.24 montre les correspondances entre le schéma de la dimension *Produit* et le schéma XML global. Dans ce cas, toutes les données sont récupérées à partir de la source `scatalogue`. La requête associée au niveau *code* récupère les codes des produits. La requête associée à la relation d'ordre entre les niveaux *code* et *catégorie* récupère la catégorie à laquelle un produit (identifié par son code) appartient. La

requête associée à la relation d'ordre entre les niveaux *catégorie* et *rayon* récupère le rayon auquel une catégorie appartient.

```
<toplevel>
  <rayon id='30' value='Logiciel'>
     <categorie id='8' value='Système d'exploitation'>
        <code id='1' value='SE001'/>
        <code id='2' value='SE012'/>
        <code id='3' value='SE019'/>
     </>
     <categorie id='9' value='Bureautique'>
        <code id='4' value='LB095'/>
        <code id='5' value='LB018'/>
     </>
     .....
  </>
  <rayon id='31' value='Materiel'>
     <categorie id='10' value='Ordinateur Bureau'>
       <code id='6' value='OB293'/>
       <code id='7' value='OB122'/>
     </>
     ....
  </>
</>
```

FIG. 5.25 – *Représentation XML de l'instance de la dimension Produit*

Lorsque le début de la construction est signalé, l'intégrateur construit d'abord les dimensions et ensuite le cube. Considérons la construction de la dimension *Produit*. L'intégrateur récupère les correspondances entre cette dimension et le schéma XML global. L'intégrateur demande alors au médiateur d'évaluer les requêtes définies dans les correspondances afin de récupérer les données nécessaires à la construction. L'instance construite est ensuite stockée comme un fichier XML dans le cache (cf. figure 5.25).

L'intégrateur signale au coordinateur la fin du processus d'intégration de données. Le coordinateur indique à la fois à l'adaptateur d'entrepôt que des données intégrées sont prêtes pour être stockées dans l'entrepôt. L'adaptateur d'entrepôt récupère du cache les données intégrées et génère les expressions de SQL adéquates pour les insérer dans l'entrepôt. Le résultat est une relation stockant les données intégrées. Par exemple, l'instance de la dimension *Produit* de la figure 5.25 est stockée dans l'entrepôt sous la forme de la relation `Produit` de la figure 5.26.

Produit

id_code	*code*	*categorie*	*rayon*
1	SE001	Système d'exploitation	Logiciel
2	SE012	Système d'exploitation	Logiciel
3	SE013	Système d'exploitation	Logiciel
6	OB293	Ordinateur Bureau	Logiciel

FIG. 5.26 – *La relation* Produit *stockée dans l'entrepôt*

5.9.3 Évolution du schéma de l'entrepôt

Nous avons utilisé le langage MDL pour exprimer les besoins en matière d'évolution du schéma de l'entrepôt. L'interpréteur MDL traite alors les expressions fournies par l'administrateur afin de vérifier leur validité. Une fois qu'une expression MDL est reconnue en tant qu'expression valide, les informations sur l'opération d'évolution demandée et ses paramètres sont envoyées au coordinateur.

Le coordinateur demande au référentiel le schéma de la dimension ou du cube sur lequel l'opération d'évolution sera appliquée. Il effectue la modification demandée et si le résultat est un schéma valide, l'ancien schéma est remplacé par le schéma modifié. Le coordinateur demande ensuite à l'adaptateur d'entrepôt de réaliser les modifications dans l'entrepôt.

Nous avons testé les expressions MDL suivantes :

- `rename dimension Produit to Product.`
- `drop dimension Product.`
- `rename cube Ventes to CumulVentes.`
- `drop cube CumulVentes.`

5.10 Conclusion

Nous avons détaillé dans ce chapitre notre démarche de construction d'un gestionnaire d'entrepôt de données relationnel. Le gestionnaire est un logiciel qui fournit un

support pour faire évoluer le schéma en étoile d'un entrepôt relationnel. Le gestionnaire fournit également des outils pour construire l'entrepôt en intégrant de données en provenance des sources multiples.

Notre prototype a été développé en s'appuyant sur des technologies indépendantes de toute plate-forme matérielle. Le langage de programmation à objets Java a été employé pour programmer ses différents composants. Le langage XML a été utilisé pour représenter des données d'origines diverses. Une caractéristique importante de notre prototype est donc sa portabilité. Le langage XML assure la portabilité des données et Java assure la portabilité du code.

Chapitre 6

Conclusion

6.1 Bilan du travail effectué

Dans ce travail, nous nous sommes intéressés au problème de l'évolution du schéma des entrepôts de données. Nous avons réalisé une étude approfondie de l'état de l'art du domaine des entrepôts qui nous a permis d'identifier les besoins des utilisateurs en matière d'évolution. Notre recherche nous a menés aux résultats que nous détaillons dans la suite.

6.1.1 Modèle multidimensionnel de données

Nous avons proposé un modèle multidimensionnel de données. Une caractéristique importante de ce modèle est la séparation claire du schéma et de l'instance des dimensions et des cubes. Une autre caractéristique importante est la complexité des dimensions, qui peuvent avoir de multiples hiérarchies alternatives, et des cubes, qui sont déterminés par un ou plusieurs axes et peuvent contenir une ou plusieurs mesures.

Associé à ce modèle, nous proposons un ensemble de 16 opérateurs d'évolution de schémas multidimensionnels. Ces opérateurs comprennent la création de nouveaux schémas de dimension et schémas de cube, la suppression d'anciens schémas et la modification de schémas existants. Nous avons défini la sémantique de ces opérateurs de manière à assurer le passage des cubes et des dimensions d'un état cohérent à un autre.

6.1.2 Le langage MDL

Nous avons proposé MDL (*Multidimensional Data Definition Language*), un langage associé au modèle multidimensionnel de données que nous proposons. Nous avons conçu ce langage comme un moyen de communication entre un gestionnaire d'entrepôt et ses utilisateurs. En utilisant des expressions MDL, un utilisateur peut créer et faire évoluer le schéma multidimensionnel de l'entrepôt. Les principales caractéristiques de MDL sont les suivantes :

- MDL autorise la création de schémas multidimensionnels complexes. Il autorise en particulier la création de schémas de dimension avec plusieurs niveaux et de schémas de cube avec une ou plusieurs mesures.
- MDL autorise l'évolution de ces schémas. Il permet plus précisément l'ajout, la suppression et la modification de schémas de dimension et de schémas de cube.

6.1.3 Infrastructure de spécification des gestionnaires d'entrepôt

Nous avons proposé une infrastructure pour le développement de systèmes (que nous appelons *gestionnaires d'entrepôts*) comme un ensemble de composants intégrés dans le cadre d'une architecture générique. L'objectif majeur d'un gestionnaire est de fournir une vue multidimensionnelle abstraite d'un entrepôt qui autorise sa construction et son évolution de manière indépendante de leur stockage (relationnelle, à objets, etc.).

6.1.4 Validation experimentale

Nous avons construit un prototype de gestionnaire d'entrepôt relationnel afin de valider notre approche. Nous avons utilisé Java comme langage de programmation et XML (et ses technologies associées) pour représenter, manipuler et interroger les données. Notre prototype autorise la création et l'évolution du schéma de l'entrepôt relationnel, ainsi que sa construction à partir de sources relationnelles.

6.1.5 Principales contributions

Nous souhaitons souligner certains aspects abordés dans cette thèse qui nous paraissent être des contributions importantes :

- modèle multidimensionnel et opérateurs d'évolution de schémas multidimensionnels
- infrastructure pour la construction des gestionnaires d'entrepôt.
- validation expérimentale

Ces points constituent, nous semble-t-il, un travail original par rapport aux travaux existants concernant les entrepôts de données.

6.2 Perspectives

Nous avons identifié un certain nombre de voies dans lesquelles notre recherche peut être poursuivie. Nous discutons dans les paragraphes 6.2.1, 6.2.2 et 6.2.3 celles liées directement au travail présenté dans ce document. Nous présentons dans le paragraphe 6.2.4 un ensemble d'axes de recherche plus généraux.

6.2.1 Langage pour la création, l'évolution et la manipulation d'entrepôts

Une perspective de recherche qui nous semble intéressante est la proposition d'un langage permettant la création et la manipulation d'entrepôts de données. Dans ce contexte, notre langage MDL assure les aspects concernant la définition de données. Le travail à faire consiste alors en la proposition d'un langage d'interrogation déclaratif pour les entrepôts. Ce langage permettrait d'exprimer des opérations de manipulation de cubes et de dimensions. Dans [Vu00] nous présentons un premier pas dans cette direction.

6.2.2 Adaptation de l'entrepôt suite aux évolutions

Nous avons discuté le fait que faire évoluer le schéma d'un entrepôt peut provoquer des incompatibilités avec les données et les programmes qui dépendent de lui. Nous

avons étudié en particulier les effets provoqués par les évolutions du schéma sur les données déjà stockées dans l'entrepôt.

Il serait intéressant d'étendre notre étude pour prendre en compte les agrégats précalculés et les index stockés dans l'entrepôt. En effet, la définition des agrégats doit évoluer en fonction des modifications survenant au niveau du schéma afin de préserver la cohérence structurelle de l'entrepôt. En ce qui concerne les index, les évolutions du schéma peuvent guider la création de nouveaux index ou la suppression des index obsolètes.

6.2.3 Gestionnaires d'entrepôt et les services NODS

Notre travail fait partie du projet de recherche NODS (*Networked Open Database Services*) [Col00]. L'objectif général de ce projet est de spécifier et de développer une infrastructure logicielle et des services bases de données ouverts et répartis facilitant la construction des serveurs de données et d'applications, serveurs aujourd'hui au cœur des systèmes d'information du réseau. L'idée est de ne plus voir un SGBD comme un logiciel fermé dédié à la gestion de données mais plutôt comme un ensemble de services coopérants, chaque service pouvant être adapté au mieux aux besoins de l'application qui va l'utiliser. Le projet NODS propose alors de décomposer un SGBD en services de différentes granularités et de ce fait de décloisonner les technologies utilisées pour réaliser les fonctions des SGBD.

Nous voulons valider l'utilisation des services NODS pour construire, maintenir et interroger des entrepôts de données :

- Le service de persistance peut être utilisé pour construire des gestionnaires de stockage capables de gérer les gros volumes de données des entrepôts.
- Le service de réplication sera utilisé lors de la construction et/ou le rafraîchissement pour détecter la mise à jour d'une source et pour propager cette modification vers l'entrepôt.
- Le service d'événement peut être utilisé pour construire des gestionnaires d'événements capables de détecter et gérer les événements produits pendant le cycle de vie d'un entrepôt (espace disque insuffisante, source indisponible, etc.).
- Le service d'interrogation est indispensable lors de la médiation de données de multiples sources hétérogènes, ainsi que lors de l'interrogation de gros volumes de données.

6.2.4 D'autres perspectives de recherche

Les entrepôts de données constituent un terrain fertile pour effectuer de nouvelles recherches. Nous avons identifié un ensemble de perspectives de recherche, comme l'adaptation des techniques d'indexation des bases de données spatiales, la proposition de langages de requêtes pour faciliter l'expression de requêtes sur les entrepôts, la génération automatique à partir de spécifications déclaratives des adaptateurs/moniteurs et des intégrateurs, et des nouvelles techniques d'analyse.

Nous avons également identifié d'autres axes de recherche. Il nous semble important d'étudier les liens avec d'autres domaines, comme les bases de données géographiques et temporelles, pour mieux représenter l'espace et le temps et enrichir ainsi le schéma des entrepôts. Il existe plusieurs points communs entre les entrepôts et les bases de données statistiques, comme la modélisation de dimensions et l'interrogation des données multidimensionnelles [Sho97]. Cependant, les résultats obtenus dans les bases de données statistiques commencent à peine à être réutilisés dans le contexte des entrepôts. Ce même phénomène se produit entre les entrepôts et les bases de données temporelles [Soo91, Sno99]. Il est évident que les travaux réalisés dans ce domaine sur la modélisation de la dimension temporelle peuvent être réutilisés dans les entrepôts.

Il serait aussi intéressant d'adapter les techniques de gestion de cache et de préchargement pour le traitement de requêtes sur les entrepôts [DRBG00]. En effet, l'ensemble d'agrégats précalculés peut être considéré comme un cache dont le contenu peut être adapté selon les besoins d'interrogation [KR99]. Il est également possible d'anticiper, dans certains cas, les besoins des utilisateurs et précharger dans le cache les données qui sont supposées être utiles pour répondre à une requête future [Sap99]. Les techniques de préchargement complètent la stratégie de cache utilisée.

Notre conclusion générale est que le domaine des entrepôts de données est jeune et qu'il n'est pas encore stable. Enfin, il est clair que les données générées aujourd'hui sont de plus en plus nombreuses, volumineuses, hétérogènes et réparties. Les intégrer dans des entrepôts, les organiser, les gérer, les rafraîchir et les analyser efficacement constituent les principaux défis pour construire les entrepôts de données du futur.

Bibliographie

[AAD+96] Sameet Agarwal, Rakesh Agrawal, Prasad Deshpande, Ashish Gupta, Jeffre Naughton, Raghu Ramakrishnan et Sunita Sarawagi. On the Computation of Multidimensional Aggregates. Dans *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB '96)*, pages 506–521, Bombay, India, September 1996.

[Adi81] Michel Adiba. Derived Relations: A Unified Mechanism for Views, Snapshots, and Distributed Data. Dans *Proceedings of the 7th International Conference on Very Large Data Bases (VLDB '81)*, pages 293–305, Cannes, France, September 1981.

[AGS95] Rakesh Agrawal, Ashish Gupta et Sunita Sarawagi. Modeling Multidimensional Databases. Rapport de recherche, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, USA, September 1995.

[AHV00] Serge Abiteboul, Rick Hull et Victor Vianu. *Fondements des bases de données*. Vuibert Informatique, 2000. Traduction de Patrick Cegielski.

[App98] Applix Inc. *TM1 Technology*, 1998. White Paper.

[AQM+97] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom et Janet Wiener. The Lorel Query Language for Semistructured Data. *Journal of Digital Libraries*, 1(1):68–88, 1997.

[Arb98] Arbor Software. *Essbase*, 1998. White paper.

[BB01] Zohra Bellahsène et Xavier Baril. XML et les systèmes d'intégration de données. *Ingénierie des Systèmes d'Information*, 6(3):11–32, 2001.

[BEH+94] John Boyle, Stephen G. Eick, Matthias Hemmje, Daniel A. Keim, J. P. Lee et Eric E. Sumner. Database Issues for Data Visualization: In-

teraction, User Interfaces, and Presentation. Dans *Proceedings of the 1994 IEEE Workshop on Database Issues for Data Visualization*, pages 25–34, October 1994.

[Bel95] Tarik Beldjilali. Managing Schema Evolution in an Object-Oriented Database Already Populated by Instances. Dans *Workshop Proceedings - 6th International Conference and Workshop on Database and Expert Systems Applications (DEXA'95)*, pages 214–222, London, United Kingdom, September 1995.

[Bel96] Zohra Bellahsène. View Mechanism for Schema Evolution. Dans *Proceedings of the 14th British National Conferenc on Databases (BNCOD 14)*, pages 18–35, Edinburgh, UK, July 1996.

[Bel97] Zohra Bellahsène. Extending a View Mechanism to Support Schema Evolution in Federated Database Systems. Dans *Proceedings of the 8th International Conference Database and Expert Systems Applications (DEXA '97)*, pages 573–582, Toulouse, France, September 1997.

[Bel98] Zohra Bellahsène. Structural View Maintenance in Data Warehousing Systems. Dans *Mémoires des Journées Bases de Données Avancées (BDA '98)*, Tunis, October 1998.

[Ber92] Elisa Bertino. A View Mechanism for Object-Oriented Databases. Dans *Proceedings of the 3rd International Conference on Extending Database Technology (EDBT'92)*, pages 136–151, Vienna, Austria, March 1992.

[BFK95] Philippe Brèche, Fabrizio Ferrandina et Martin Kuklok. Simulation of Schema Change using Views. Dans *Proceedings of the 6th International Conference on Database and Expert Systems Applications (DEXA'95)*, pages 247–258, September 1995.

[BG98] Edgard Benítez-Guerrero. Vers un modèle à objets pour les entrepôts de données. Mémoire, DEA Informatique : Systèmes et Communications-Université Joseph Fourier, Grenoble, France, Juin 1998. Responsables : Christine Collet et Michel Adiba.

[BGCA99] Edgard Benítez-Guerrero, Christine Collet et Michel Adiba. Entrepôts de données : synthèse et analyse. Rapport de recherche RR 1017-I-LSR-8, Laboratoire LSR-IMAG, Mai 1999.

[BGCA01] Edgard Benitez-Guerrero, Christine Collet et Michel Adiba. Entrepôts de données : caractéristiques et problématique. *Technique et science informatiques*, 20(2):145 à 178, Ed. Hermès Science Publications, 2001.

[BH92] M. Bright et A. Hursen. A Taxonomy and Current Issues in Multidatabase Systems. *IEEE Computer*, 25(3):213–224, March 1992.

[BKKK87] Jay Banerjee, Won Kim, Hyoung Joo Kim et Henry F. Korth. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. Dans *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data (SIGMOD '87)*, pages 311–322, San Francisco, U.S.A., May 1987.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider et Bernhard Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. Dans *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data (SIGMOD '90)*, pages 322–331, Atlantic City, U.S.A., May 23-25 1990.

[BL96] Zohra Bellahsène et Gilles Lucato. Schema Evolution by Means of a View. *French Journal Information Systems Engineering (ISI)*, 4(5):661–688, December 1996.

[Bla99] Markus Blaschka. FIESTA: A Framework for Schema Evolution in Multidimensional Information Systems. Dans *Proceedings of the 6th CAiSE Doctoral Consortium*, Heidelberg, Germany, June 1999.

[BM93] Kwang-June Byeon et Dennis McLeod. Towards the Unification of Views and Versions for Object Databases. Dans *Proceedings of the First International Symposium on Object Technologies for Advanced Software (ISOTAS'93)*, pages 220–236, Kanazawa, Japan, November 1993.

[BPT97] Elena Baralis, Stefano Paraboschi et Ernest Teniente. Materialized View Selection in a Multidimensional Database. Dans *Proceedings of*

the 23rd International Conference on Very Large Data Bases (VLDB '97), pages 156–165, Athenes, Greece, 1997.

[Bra92] Svein-Erik Bratsberg. Unified Class Evolution by Object-Oriented Views. Dans *Proceedings of the 11th International Conference on the Entity-Relationship Approach (ER'92)*, pages 423–439, Karlsruhe, Germany, October 1992.

[BSH99] Markus Blaschka, Carsten Sapia et Gabriele Höfling. On Schema Evolution in Multidimensional Databases. Dans *Proceedings of First International Conference on Data Warehousing and Knowledge Discovery (DaWak'99)*, pages 153–164, Florence, Italy, August 1999.

[BSHD98] Markus Blaschka, Carsten Sapia, Gabriele Höfling et Barbara Dinter. Finding your way through multidimensional Data models. Dans *Proceedings of International Workshop on Data Warehouse Design and OLAP Technology (DWDOT)*, pages 198–203, Vienna, Autrich, August 1998.

[CD97] Surajit Chaudhuri et Umeshwar Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26(1):65–74, March 1997.

[CGL+98] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi et R. Rosati. Source Integration in Data Warehousing. Dans *Proceedings of the 9th International Workshop on Database and Expert Systems Application (DEXA Workshop '98)*, pages 192–197, Vienna, Austria, August 1998.

[CHS+95] Michael J. Carey, Laura M. Haas, Peter M. Schwarz, Manish Arya, William F. Cody, Ronald Fagin, Myron Flickner, Allen Luniewski, Wayne Niblack, Dragutin Petkovic, Joachim Thomas II, John H. Williams et Edward L. Wimmers. Towards Heterogeneous Multimedia Information Systems: The Garlic Approach. Dans *Proceedings of the Fifth International Workshop on Research Issues in Data Engineering - Distributed Object Management (RIDE-DOM '95)*, pages 124–131, Taipei, Taiwan, March 1995.

[CHS01] Rada Chirkova, Alon Y. Halevy et Dan Suciu. A Formal Perspective on the View Selection Problem. Dans *Proceedings of 27th International*

Conference on Very Large Data Bases (VLDB'01), pages 59–68, Roma, Italy, September 2001.

[CHY96] Ming-Syan Chen, Jiawei Han et Philip-S. Yu. Data Mining: An Overview from Database Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):866–883, 1996.

[CJ98] Volker Coors et Volker Jung. Using VRML as an Interface to the 3D Data Warehouse. Dans *Proceedings of the Third Symposium on the Virtual Reality Modeling Language (VRML '98)*, page 121, Monterey, CA, USA, February 1998.

[Cla94] Stewart M. Clamen. Schema Evolution and Integration. *Distributed and Parallel Databases*, 2(1):101–126, 1994.

[CNS99] Sara Cohen, Werner Nutt et Alexander Serebrenik. Algorithms for Rewriting Aggregate Queries Using Views. Dans *Design and Management of Data Warehouses*, page 9, 1999.

[Cod93] E. Codd. Providing OLAP (On-Line Analytical Processing) to Users-Analysts: An IT Mandate. Rapport de recherche, E.F. Codd and Associates, 1993.

[Col96] George Colliat. OLAP, Relational, and Multidimensional Database Systems. *SIGMOD Record*, 25(3):64–69, 1996.

[Col00] Christine Collet. The NODS project : Networked Open Database Services. Dans *Proceedings of ECOOP Symposium on Objects and Databases*, June 2000.

[CR01] Dan Vodislav Chantal Reynaud, Jean-Pierre Sirot. Semantic Integration of XML Heterogeneous Data Sources. Dans *Proceedings of the 2001 International Database Engineering and Applications Symposium (IDEAS'01)*, pages 199–208, Grenoble, France, July 2001.

[CT97] Luca Cabibbo et Riccardo Torlone. Querying Multidimensional Databases. Dans *Proceedings of 6th Internation Workshop on Database Programming Languages (DBPL-6)*, pages 319–335, Estes Park, Colorado, U.S.A., August 1997.

[CT98] Luca Cabibbo et Riccardo Torlone. A Logical Approach to Multidimensional Databases. Dans *Proceedings of 6th International Workshop on Extending Database Technology (EDBT '98)*, pages 183–197, Valencia, Spain, March 1998.

[CVSGR00] Christine Collet, Genoveva Vargas-Solar et Helena Grazziotin-Ribeiro. Open Active Services for Data-Intensive Distributed Applications. Dans *16èmes journées Bases de Données Avancées (BDA'2000)*, pages 43–60, Blois, Octobre 2000.

[Dat00] Chris Date. *An Introduction to Database Systems.* Addison-Wesley Pub. Co., 2000.

[DeR98] Steven J. DeRose. XQuery: A Unified Syntax for Linking and Querying general XML. Dans *Proceedings of The Query Languages Workshop (QL'98)*, Boston, USA, December 1998.

[DG92] David J. DeWitt et Jim Gray. Parallel Database Systems: The Future of High Performance Database Systems. *Communications of the ACM*, 35(6):85–98, June 1992.

[DMT98] Anindy Datta, Bongki Moon et Helen Thomas. A Case for Parallelism in Data Warehousing and OLAP. Dans *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications (DEXA Workshop '98)*, pages 226–231, Vienna, Austria, August 1998.

[DOM98] Document Object Model (DOM) Level 1. http://www.w3.org/TR/xpath, October 1998.

[DRBG00] Stephane Drapeau, Claudia L. Roncancio et Edgard Benítez-Guerrero. Generating Association Rules for Prefetching. Dans *Proceedings of the ICDCS International Workshop of Knowledge Discovery and Data Mining in the World-Wide Web (KDMW'00)*, Taipei, Taiwan, Republic of China, April 2000.

[DSHB98] Barbara Dinter, Carsten Sapia, Gabriele Höfling et Markus Blaschka. The OLAP Market: State of the Art and Research Issues. Dans *Proceedings of the CIKM First International Workshop on Data Warehousing and OLAP (DOLAP '98)*, Washington, D.C., U.S.A., November 1998.

[DWI98] Data Warehouse Institute, 1998. http://www.dw-institute.com.

[ETI98] Evolutionary Technologies International, 1998. http://www.evtech.com.

[FB74] Raphael A. Finkel et Jon Louis Bentley. Quad Trees: A Data Structure for Retrieval on Composite keys. *Acta Informatica*, 4:1–9, 1974.

[FDL+99] D. Florescu, A. Deutsch, A. Levy, M. Fernandez et D. Suciu. A Query Language for XML. Dans *Proceedings of Eighth International World Wide Web Conference(WWW '99)*, 1999.

[Fla02] David Flanagan. *Java In A Nutshell*. O'Reilly and Associates, 2002.

[FMZ94] Fabrizio Ferrandina, Thorsten Meyer et Roberto Zicari. Implementing Lazy Database Updates for an Object Database System. Dans *Proceedings of 20th International Conference on Very Large Data Bases (VLDB'94)*, pages 261–272, Santiago de Chile, Chile, September 1994.

[FMZ+95] Fabrizio Ferrandina, Thorsten Meyer, Roberto Zicari, Guy Ferran et Joëlle Madec. Schema and Database Evolution in the O2 Object Database System. Dans *Proceedings of 21th International Conference on Very Large Data Bases (VLDB'95)*, pages 170–181, Zurich, Switzerland, September 1995.

[FPSSU96] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth et Ramasamy Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, U.S.A., 1996.

[Fra97] Jean-Michel Franco. *Le Data Warehouse*. Eyrolles, France, 1997.

[GC97] Sanjay Goil et Alok Choudhary. Sparse Data Storage of Multi-Dimensional Data for OLAP and Data Mining. Rapport de recherche CPDC-TR-9801-005, Center for Parallel and Distributed Computing, Northwestern University, November 1997.

[GCB+97] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow et Hamid Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By,

Cross-Tab, and Sub-Totals. *Data Mining and Knowledge Discovery: An International Journal*, 1(1):29–53, March 1997.

[GFSS00] Helena Galhardas, Daniela Florescu, Dennis Shasha et Eric Simon. Ajax: An Extensible Data Cleaning Tool. Dans *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, page 590, Dallas, USA, May 2000.

[GHQ95] Ashish Gupta, Venky Harinarayan et Dallan Quass. Aggregate-Query Processing in Data Warehousing Environments. Dans *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB '95)*, pages 358–369, Zurich, Switzerland, September 1995.

[GL97] Marc Gyssens et Laks Lakshmanan. A Foundation for Multi-Dimensional Databases. Dans *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB '97)*, pages 106–115, Athenes, Greece, August 1997.

[GLT97] Timothy Griffin, Leonid Libkin et Howard Trickey. An Improved Algorithm for the Incremental Recomputation of Active Relational Expressions. *Knowledge and Data Engineering*, 9(3):508–511, 1997.

[GM95] Ashish Gupta et Inderpal Singh Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin*, 18(2):3–18, June 1995.

[GMLWZ98] Hector Garcia-Molina, Wilburt Labio, Janet L. Wiener et Yue Zhuge. Distributed and Parallel Computing Issues in Data Warehousing (abstract). Dans *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing (PODC '98)*, page 7, Puerto Vallarta, Mexico, June 28 - July 2 1998.

[GMPQ+97] Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey D. Ullman, Vasilis Vassalos et Jennifer Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems (JIIS)*, 8(2):117–132, March/April 1997.

[GMV96] Isabelle Guyon, Nada Matic et Vladimir Vapnik. *Advances in Knowledge Discovery and Data Mining*, chapitre Discovering Informative Patterns and Data Cleaning., pages 181–203. AAAI/MIT Press, 1996.

[Gup97] Himanshu Gupta. Selection of Views to Materialize in a Data Warehouse. Dans *Proceedings of the 6th International Conference on Database Theory (ICDT '97)*, pages 98–112, Delphi, Greece, January 1997.

[Gut84] Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. Dans *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data (SIGMOD '84)*, pages 47–57, Boston, Massachusetts, June 1984.

[GW97] Roy Goldman et Jennifer Widom. DataGuides: Enabling Query Formulation and Optimization in semistructured Databases. Dans *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 436–445, Athens, Greece, August 1997.

[HFW+96] Jiawei Han, Yongjian Fu, Wei Wang, Krzysztof Koperski et Osmar Zaiane. DMQL: A Data Mining Query Language for Relational Databases. Dans *Proceedings of the ACM SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'96)*, Montreal, Canada, June 1996.

[HGMW+95] Joachim Hammer, Hector Garcia-Molina, Jennifer Widom, Wilburt Labio et Yue Zhuge. The Stanford Data Warehousing Project. *IEEE Data Engineering Bulletin*, 18(2):41–48, 1995.

[HK00] Jiawei Han et Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2000.

[HMV99a] Carlos A. Hurtado, Alberto O. Mendelzon et Alejandro A. Vaisman. Maintaining Data Cubes under Dimension Updates. Dans *Proceedings of the 15th International Conference on Data Engineering (ICDE 1999)*, pages 346–355, Sydney, Austrialia, march 1999.

[HMV99b] Carlos A. Hurtado, Alberto O. Mendelzon et Alejandro A. Vaisman. Updating OLAP Dimensions. Dans *Proceedings of the Second Interna-*

tional Workshop on Data Warehousing and OLAP (DOLAP'99), pages 60–66, Kansas City, USA, november 1999.

[Hon95] E. Hong. A Schema Evolution Mechanism. Dans *Workshop Proceedings of the 6th International Conference and Workshop on Database and Expert Systems Applications (DEXA'95)*, pages 97–104, September 1995.

[HRU96] V. Harinarayan, A. Rajaraman et J. Ullman. Implementing Data Cubes Efficiently. Dans *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD '96)*, pages 205–227, Montreal, Canada, 1996.

[HS98] Mauricio A. Hernández et Salvatore J. Stolfo. Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery: An International Journal*, 2(1):9–37, 1998.

[HSB00] Karl Hahn, Carsten Sapia et Markus Blaschka. Automatically Generating OLAP Schemata from Conceptual Graphical Models. Dans *Proceedings of 3rd International Workshop on Data Warehousing and OLAP (DOLAP'00)*, Washington D.C., USA, November 2000.

[Hyp99] Hyperion Solutions. *Essbase Partitioning Option*, 1999. White paper.

[IBM02] IBM Corporation. *DB2 Universal Server (v7.1) - SQL Reference*, 2002. http://www-3.ibm.com/software/data/db2/library/.

[IM96] Tomasz Imielinski et Heikki Mannila. A Database Perspective on Knowledge Discovery. *Communications of the ACM*, 39(11):58–64, November 1996.

[Inf98] Informix. *Data Warehouse Administrator's Guide*, 1998. White paper.

[Inm92] William Inmon. *Building the Data Warehouse*. QED Technical Publishing Group, Wellesley, Massachusetts, U.S.A., 1992.

[Inm95] William Inmon. *What is a Data Warehouse?*, 1995. White paper.

[Jac00] Reed Jacobson. *SQL Server Analysis Services: Step by Step*. Microsoft Press, 2000.

[JLS99] H V. Jagadish, Laks V. S. Lakshmanan et Divesh Srivastava. What can Hierarchies do for Data Warehousing? Dans *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*, Edinburg, Scotland, September 1999.

[JLVV00] Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou et Panos Vassiliadis. *Fundamentals of Data Warehouses*. Springer Verlag, 2000.

[JV97] M. Jarke et Y. Vassiliou. Data Warehouse Quality Design: A Review of the DWQ Project. Dans *Proceedings of the 2nd Conference on Information Quality*, Cambridge, USA, 1997.

[KC88] Won Kim et Hong-Tai Chou. Versions of Schema for Object-Oriented Databases. Dans *Proceedings of the 14th International Conference on Very Large Data Bases*, pages 148–159, Los Angeles, USA, August 29 - September 1, 1988.

[Ken95] Kenan Systems Corporation. *An Introduction to Multidimensional Database Technology*, 1995. White paper.

[Ken97] Kenan Systems Corporation. *Multidimensional Database Technology and Data Warehousing*, 1997. White paper.

[Kim96] Ralph Kimball. *The Data Warehouse Toolkit*. John Wiley, U.S.A., 1996.

[KLTW95] Daniel A. Keim, John Peter Lee, Bhavani M. Thuraisingham et Craig M. Wittenbrink. Database Issues for Data Visualization: Supporting Interactive Database Exploration. Dans *Proceedings of the 1995 IEEE Workshop on Database Issues for Data Visualization*, pages 12–25, October 1995.

[KMP02] Panos Kalnis, Nikos Mamoulis et Dimitris Papadias. View Selection using Randomized Search. *Data Knowledge Engineering*, 42(1):89–111, 2002.

[KR99] Yannis Kotidis et Nick Roussopoulos. DynaMat: A Dynamic View Management System for Data Warehouses. Dans *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD'99)*, Philadephia, USA, June 1999.

[KS95a] Ralph Kimball et Kevin Strehlo. SQL is Our Language, Fix it Now. *SIGMOD Record*, 24(3), 1995.

[KS95b] Ralph Kimball et Kevin Strehlo. Why Decision Support Fails and How to Fix it. *SIGMOD Record*, 24(3):92–97, 1995.

[Lau96] Sven-Eric Lautemann. An Introduction to Schema Versioning in OODBMS. Dans *Proceedings of the Seventh International Workshop on Database and Expert Systems Applications (DEXA Workshop 1996)*, pages 132–139, Zurich,Switzerland, September 1996.

[LDB97] Zoé Lacroix, Claude Delobel et Philippe Brèche. Object Views and Database Restructuring. Dans *Proceedings of the 6th International Workshop on Database Programming Languages (DBPL-6)*, pages 180–201, Estes Park, USA, August 1997.

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv et Divesh Srivastava. Answering Queries using Views. Dans *Proceedings of the 14th ACM Symposium on Principles of Database Systems (PODS '95)*, pages 95–104, San Jose, CA, U.S.A., March 1995.

[Lon01] Stéphanie Lonchampt. Vers une infrastructure pour la construction et l'évolution d'entrepôts de données. Mémoire, Diplôme d'Ingénieur en Informatique - ENSIMAG, INPG, Grenoble, France, Septembre 2001. Responsables : Christine Collet et Edgard Benitez-Guerrero.

[LW96] Chang Li et X.Sean Wang. A Data Model for Supporting On-line Analytical Processing. Dans *Proceedings of the Fifth International Conference on Information and Knowledge Management (CIKM '96)*, pages 81–88, Rockville, U.S.A., November 1996.

[LZW+97] Wilburt Labio, Yue Zhuge, JanetL. Wiener, Himanshu Gupta, Hector Garcia-Molina et Jennifer Widom. The WHIPS Prototype for Data Warehouse Creation and Maintenance. Dans *Proceedings ACM SIGMOD International Conference on Management of Data (SIGMOD 1997)*, pages 557–559, Tucson, USA, May 1997.

[Man97] Heikki Mannila. Inductive Databases and Condensed Representations for Data Mining. Dans *Proceedngs of the International Symposium on Logic Programming (ILPS '97)*, pages 21–30, Long Island, U.S.A., 1997.

[Mic98] MicroStrategy. *The Case for Relational OLAP*, 1998. White paper.

[Mon93] Simon R. Monk. *A Model for Schema Evolution in Object-Oriented Database Systems*. PhD thesis, Lancaster University, 1993.

[MPC96] Rosa Meo, Giuseppe Psaila et Stefano Ceri. A New SQL-like Operator for Mining Association Rules. Dans *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB '96)*, pages 122–133, Bombay, India, September 1996.

[MQM97] Inderpal S. Mumick, Dallan Quass et Barinderpal S. Mumick. Maintenance of Data Cubes and Summary Tables in a Warehouse. Dans *Proceedings of the ACM SIGMOD Conference on Management of Data (SIGMOD'97)*, pages 100–111, Tuscon, Arizona, May 1997.

[MS90] L. Edwin McKenzie et Richard T. Snodgrass. Schema Evolution and the Relational Algebra. *Information Systems Journal*, 15(2):207–232, 1990.

[MS93] Simon R. Monk et Ian Sommerville. Schema Evolution in OODBs Using Class Versioning. *SIGMOD Record*, 22(3):16–22, 1993.

[NLR98] Anisoara Nica, Amy J. Lee et Elke A. Rundensteiner. The CVS Algorithm for View Synchronization in Evolvable Large-Scale Information Systems. Dans *Proceedings of the 6th International Conference on Extending Database Technology (EDBT'98)*, pages 359–373, Valencia, Spain, March 1998.

[Odb92] Erik Odberg. A Framework for Managing Schema Versioning in Object-Oriented Databases. Dans *Proceedings of the 3rd International Conference on Database and Expert Systems Applications (DEXA'92)*, pages 115–120, Valencia, Spain, 1992.

[OG95] Patrick O'Neil et Goetz Graefe. Multi-Table Joins Through Bitmapped Join Indices. *SIGMOD Record*, 24(3):8–11, 1995.

[O'N87] Patrick O'Neil. Model 204 - Architecture and Performance. Dans *Proceedings of the Second International Workshop on High Performance Transaction Systems*, pages 40–59, Pacific Grove, USA, September 1987.

[OQ97] Patrick O'Neil et Dallan Quass. Improved Query Performance with Variant Indexes. Dans *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data (SIGMOD '97)*, pages 38–49, Tucson, Arizona, U.S.A., May 1997.

[Ora98a] Oracle Corporation. *Oracle Warehouse: Unleash the Power of Information*, 1998. White paper.

[Ora98b] Oracle Corporation. *Oracle8i for Data Warehousing : Fast and Simple for More Data and More Users*, 1998. White paper.

[Ora01] Oracle Corporation. *Oracle9i Data Warehousing Guide*, 2001. White paper.

[Ora02] Oracle Corporation. *Oracle9i (v9.2) - Online Documentation*, 2002. http://otn.oracle.com/docs/products/oracle9i/.

[PC98] Nigel Pendse et Richard Creeth. *The OLAP Report*. Business Intelligence Inc., U.S.A., 1998. http://www.olapreport.com.

[PGMW95] Yannis Papakonstantinou, Hector Garcia-Molina et Jennifer Widom. Object Exchange Across Heterogeneous Information Sources. Dans *Proceedings of the 11th International Conference on Data Engineering (ICDE'95)*, pages 251–260, Taipei, Taiwan, March 6 - 10 1995.

[Pil98] Pilot Software. *An Introduction to OLAP: Multidimensional Terminology and Technology*, 1998. White paper.

[PS87] D. Jason Penney et Jacob Stein. Class Modification in the GemStone Object-Oriented DBMS. Dans *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'87)*, pages 111–117, October 1987.

[Qui99] Christoph Quix. Repository Support for Data Warehouse Evolution. Dans *Proceedings of the Intl. Workshop on Design and Management of Data Warehouses (DMDW'99)*, Heidelberg, Germany, June 14-15 1999.

[Red96] Red Brick Systems. *Decision-Makers, Business Data and RISQL*, 1996. White paper.

[RKZ+99] Elke A. Rundensteiner, Andreas Koeller, Xin Zhang, Amber van Wyk, Yong Li, Amy J. Lee et Anisoara Nica. Evolvable View Environment (EVE): Non-Equivalent View Maintenance under Schema Changes. Dans *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 1999)*, pages 553–555, Philadelphia, USA, 1999.

[RKZ00] Elke A. Rundensteiner, Andreas Koeller et Xin Zhang. Maintaining Data Warehouses over Changing Information Sources. *Communications of the ACM*, 43(6):57–62, june 2000.

[RLS98] Jonathan Robie, Joe Lapp et David Schach. Xml Query Language (XQL). Dans *Proceedings of The Query Languages Workshop (QL'98)*, Boston, USA, December 1998.

[Rod92] John F. Roddick. Schema Evolution in Database Systems - An Annotated Bibliography. *SIGMOD Record*, 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 et Elke A. Rundensteiner. A Transparent Schema-Evolution System Based on Object-Oriented View Technology. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 9(4):600–624, July/August 1997.

[SA99] Arnaud Sahuguet et Fabien Azavant. Building Light-Weight Wrappers for Legacy Web Data-Sources Using W4F. Dans *(VLDB '99)*, pages 738–741, 1999.

[Sap99] C. Sapia. On Modeling and Predicting Query Behavior in OLAP Systems. Dans *Proceedings of the Intl. Workshop on Design and Mana-*

gement of Data Warehouses (DMDW'99), Heidelberg, Germany, June 14-15 1999.

[Sar97] Sunita Sarawagi. Indexing OLAP Data. *IEEE Data Engineering Bulletin*, 20(1):36–43, March 1997.

[SCR+99] Shashi Shekhar, Sanjay Chawla, Siva Ravada, Andrew Fetterer, Xuan Liu et Chang tien Lu. Spatial Databases - Accomplishments and Research Needs. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):45–55, January/February 1999.

[SDJL96] Divesh Srivastava, Shaul Dar, H. V. Jagadish et Alon Y. Levy. Answering Queries with Aggregation Using Views. Dans *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB '96)*, pages 318–329, Bombay, India., September 1996.

[Sea98] Seagate. *Seagate Holos : In Depth Analysis*, 1998. White paper.

[Sho97] Arie Shoshani. OLAP and Statistical Databases: Similarities and Differences. Dans *Proceedings of the 16th ACM Symposium on Principles of Database Systems (PODS '97)*, pages 185–196, Tucson, Arizona, U.S.A., May 1997.

[SL90] Amit P. Sheth et James A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[SMKK98] Sunil Samtani, Mukesh K. Mohania, Vijay Kumar et Yahiko Kambayashi. Recent Advances and Research Problems in Data Warehousing. Dans *Proceedings of ER '98 Workshops on Data Warehousing and Data Mining, Mobile Data Access, and Collaborative Work Support and Spatio-Temporal Data Management*, pages 81–92, Singapore, September 1998.

[Sno99] Richard T. Snodgrass. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann, 1999.

[Soo91] Michael D. Soo. Bibliography on Temporal Databases. *SIGMOD Record*, 20(1):14–23, 1991.

[SP94] Stefano Spaccapietra et Christine Parent. View Integration: A Step Forward in Solving Structural Conflicts. *Knowledge and Data Engineering*, 6(2):258–274, 1994.

[Spe97] Speedware Corporation Inc. *Media/MR: The Hybrid OLAP Technology for the Enterprise*, 1997. White paper.

[SRF87] Timos K. Sellis, Nick Roussopoulos et Christos Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. Dans *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB '87)*, pages 507–518, September 1-4 1987.

[SZ90] Andrea H. Skarra et Stanley B. Zdonik. Type Evolution in an Object-Oriented Database. Dans Alfonso F. Cardenas et Dennis McLeod, éditeurs, *Research Foundations in Object-Oriented and Semantic Database Systems*, pages 137–155. Prentice-Hall, 1990.

[TDV97] Helen Thomas, Anindya Datta et Igor Viguier. A Conceptual Model and Algebra for On-Line Analytical Processing in Decision Support Databases. Rapport de recherche, University of Arizona, 1997.

[TS93] Markus Tresch et Marc H. Scholl. Schema Transformation without Database Reorganization. *SIGMOD Record*, 22(1):21–27, 1993.

[VAL98] Vality technology incorporated, 1998. http://www.vality.com.

[Vas98] Panos Vassiliadis. Modeling Multidimensional Databases, Cubes and Cube Operations. Dans *Proceedings of the 10th International Conference on Scientific and Statistical Database Management (SSDBM '98)*, pages 53–62, 1998.

[VDR97] I. Viguier, Anindya Datta et K. Ramamritham. Have Your Data and Index it, Too. Rapport de recherche, University of Arizona, 1997.

[VGD99] Athanasios Vavouras, Stella Gatziu et Klaus R. Dittrich. The SIRIUS Approach for Refreshing Data Warehouses Incrementally. Dans *GI-Fachtagung Datenbanken in Büro, Technik und Wissenschaft (BTW'99)*, pages 80–96, March 1999.

[VRM97] The VRML Consortium Incorporated. *The Virtual Reality Modeling Language*, 1997. International Standard ISO/IEC 14772-1.

[Vu00] Tuyet-Trinh Vu. Application des techniques OLAP à un entrepôt de données XML. Mémoire, Diplôme d'Etudes Professionnelles Approfondies - Institut de la Francophonie pour l'Informatique, Vietnam, Septembre 2000. Responsables : Christine Collet et Edgard Benitez-Guerrero.

[WB97] Ming-Chuan Wu et Alejandro Buchmann. Research Issues in Data Warehousing. Dans *GI-Fachtagung Datenbanken in Büro, Technik und Wissenschaft (BTW'97)*, pages 61–82, Ulm, Germany, March 1997.

[Wid95] Jennifer Widom. Research Problems in Data Warehousing. Dans *Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM '95)*, pages 25–30, Baltimore, U.S.A., Novembre 1995.

[Wid98] Jennifer Widom. Querying XML with Lore. Dans *Proceedings of The Query Languages Workshop (QL'98)*, Boston, USA, December 1998.

[XML98] Extensible Markup Language (XML) 1.0 Specification. http://www.w3c.org/TR/REC-xml, February 1998.

[YM98] Clement T. Yu et Weiyi Meng. *Principles of Database Query Processing*. Morgan Kaufmann Publishers, Inc., San Francisco, USA, 1998.

[ZDN97] Yihong Zhao, Prasad Deshpande et Jeffrey F. Naughton. An Array-Based Algorithm for Simultaneous Multidimensional Aggregates. Dans *Proceedings of the 1997 ACM SIGMOD Conference on Management of Data (SIGMOD '97)*, pages 159–170, May 1997.

[ZHKF95] Gang Zhou, Richard Hull, Roger King et Jean-Claude Franchitti. Data Integration and Warehousing Using H2O. *IEEE Data Engineering Bulletin*, 18(2):29–40, June 1995.

[ZHL+98] Osmar Zaiane, Jiawei Han, Ze-Nian Li, Sonny H. Chee et Jenny Chiang. Multimedia-Miner: A System Prototype for Multimedia Data Mining.

Dans *Proceedings of the 1998 ACM SIGMOD Conference on Management of Data (SIGMOD '98)*, pages 581–583, Seattle, Washington, U.S.A., June 1998.

[ZS99] Thomas Zurek et Markus Sinnwell. Data Warehousing Has More Colours Than Just Black and White. Dans *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*, pages 726–728, Edinburgh, Scotland, UK, September 1999.

Annexe A

Quelques formalismes de cube proposés

Le modèle multidimensionnel de données a attiré l'attention de la communauté industrielle et maintenant de nombreux produits basés sur ce modèle se trouvent sur le marché ([Ken97, App98, Arb98], par exemple). Les services proposés par ces produits visent à couvrir les besoins des utilisateurs et chaque produit fournit sa propre vision du modèle et des opérations associées. Donc il n'existe pas (i) un formalisme indépendant de la mise en œuvre qui autorise la définition de dimensions structurées avec hiérarchies multiples et mesures complexes, et (ii) un langage d'interrogation sous-jacent qui facilite l'écriture de requêtes complexes dont les agrégations *ad hoc* soient autorisées [AGS95, BSHD98]. Pour pallier cette situation, la communauté de recherche a proposé plusieurs formalismes (cf. table A.1).

Hypercube

Agrawal et al. [AGS95] proposent un modèle basé sur la notion d'hypercube. Un hypercube est déterminé par k dimensions et ses éléments (cellules) sont définis par une fonction qui associe à une combinaison de valeurs des dimensions (i) la valeur 0, si la combinaison n'existe pas dans la base de données, (ii) la valeur 1, si la combinaison existe, ou (iii) un n-uplet, s'il existe de l'information supplémentaire associée aux éléments; la description des composants d'un n-uplet est stockée comme une méta donnée. Ce modèle utilise des fonctions pour la définition de hiérarchies multiples et d'agrégations ad-hoc.

Formalisme	Introduction des hiérarchies	Opérateurs
Hypercube (Agrawal et. al., 1995)	Fonctions	Pull, Push, Destroy Dimension, Restriction, Merge, Join
Cube Multidimensionnel (Li et Wang, 1996)	Relations	Add Dimension, Transfer, RC-Join Construct, Cube Aggregation, Union
Table N-Dimensionnelle (Gyssens et Lakshmanan, 1997)	Fonctions	Unfold, Fold, Selection Projection, Renaming, Union Intersection, Difference, Sumarization, Classification
Cube (Thomas et. al., 1997)	Ordre partiel entre attributs	Pull, Push, Partition Aggregation, Cartesian Product, Join, Union, Difference
F-table (Cabibbo et Torlone,1998)	Ordre partiel entre attributs	Roll-Up

TAB. A.1 – *Formalismes de cube proposés*

Les opérateurs (cf. table A.1) ont une sémantique bien définie et forment un ensemble minimal (un opérateur ne peut pas être éliminé sans perte des fonctionnalités) bien que leur définition soit complexe à cause de l'introduction des fonctions pour représenter les hiérarchies. Les opérateurs proposés peuvent être composés pour construire d'autres opérateurs de type relationnel tels que la Projection, l'Union, l'Intersection et la Différence, ainsi que l'agrégation comme Roll-Up et Drill-Down. Cette approche vise une mise en œuvre sur un SGBD relationnel et semble plutôt pragmatique.

La figure A.1(a) présente l'hypercube VENTES dont les dimensions sont PRODUIT, MAGASIN et TEMPS. Les éléments de cet hypercube contiennent des n-uplets (par exemple, l'élément qui correspond à $PRODUIT = P2$, $MAGASIN = Lyon$ et $TEMPS = 1994$ contient le n-uplet $< 1000, \ldots >$) ou la valeur 0 (pour les éléments non représentés dans la figure). La métadonnée de description des éléments est une annotation de l'hypercube (dans ce cas le n-uplet $< Quantite, \ldots >$). La figure A.1(b) présente un autre hypercube avec les dimensions PRODUIT, MAGASIN et QUANTITE et ses éléments prennent la valeur 1 (par exemple, l'élément qui correspond à $PRODUIT = P2$, $MAGASIN = Lyon$ et $QUANTITE = 1000$) ou bien la valeur 0.

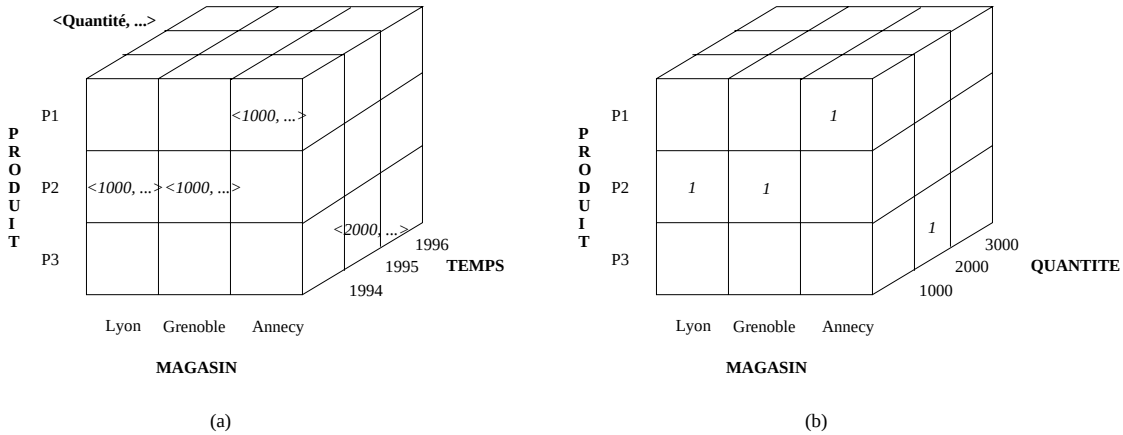


FIG. A.1 – *L'HyperCube de Agrawal et al.*

Cube Multidimensionnel

Pour Li et Wang [LW96] une base de données multidimensionnelle est un ensemble fini de Cubes Multidimensionnels et un ensemble fini de relations. Un Cube Multidimensionnel consiste en un certain nombre de relations (les dimensions) et pour chaque combinaison de n-uplets, un n-uplet pour chaque dimension, il existe une valeur associée (une mesure). Dans ce modèle, les mesures ne peuvent être que des valeurs numériques. Les auteurs introduisent la notion de relation de regroupement comme moyen pour représenter des hiérarchies. Les opérateurs (cf. table A.1) prennent des cubes et relations en entrée et produisent des cubes en sortie. Ils peuvent être composés pour exprimer des requêtes complexes.

La figure A.2(a) montre le cube multidimensionnel VENTES formé par les relations PRODUIT, MAGASIN et TEMPS. Dans cette figure on peut voir que la valeur associée aux n-uplets $MAGASIN =< Grenoble >$, $PRODUIT =< P1, 10, Noir >$ et $TEMPS = < 15, Mai, 1996 >$ est 100. La partie (b) de la figure A.2 montre la relation qui regroupe les villes de Grenoble, Lyon et Annecy dans la région Rhône-Alpes et les villes de Créteil et Paris dans la région Ile-de-France.

Tables N-dimensionnelles

Gyssens et Lakshmanan proposent dans [GL97] un formalisme autour de la notion de *table n-dimensionnelle*. Une instance de ce type de table est un ensemble de

MAGASIN	PRODUIT			TEMPS			
Ville	Nom	Poids	Couleur	Jour	Mois	Année	Quantité
Grenoble	P1	10	Noir	15	Mai	1996	100
Grenoble	P2	10	Noir	15	Mai	1996	80
Annecy	P1	20	Noir	15	Mai	1996	80
Lyon	P2	20	Blanc	15	Mai	1996	60
Creteil	P1	10	Blanc	15	Mai	1996	110
Paris	P2	10	Blanc	15	Mai	1996	150

(a)

Region	MAGASIN.Ville
Rhône-Alpes	Grenoble
Rhône-Alpes	Annecy
Rhône-Alpes	Lyon
Ile-de-France	Creteil
Ile-de-France	Paris

(b)

FIG. A.2 – *Le Cube Multidimensionnel de Li et Wang. (a) Cube Multidimensionnel. (b) Relation de regroupement*

relations, une pour chaque dimension et une autre pour les *clés de dimension* et les mesures respectives. Les hiérarchies ne sont pas explicitement mentionnées et elles sont incorporées en définissant des fonctions. Les opérateurs proposés (cf. table A.1) prennent en entrée une table n-dimensionnelle et produisent en sortie une autre table n-dimensionnelle. Les auteurs montrent qu'une table n-dimensionnelle peut être représentée par une relation classique et vice-versa, et c'est à partir de ce résultat qu'ils développent la sémantique des opérateurs. En fait, chaque opérateur convertit une table n-dimensionnelle en une relation classique, effectue les opérations nécessaires sur cette relation et finalement transforme la relation en une table n-dimensionnelle. Les opérateurs proposés peuvent être composés pour en exprimer d'autres, comme Roll-Up.

La figure A.3(a) présente la table bi-dimensionnelle VENTES avec les dimensions PRODUIT et TEMPS, et la mesure VENTES. Les attributs Nom et Poids sont associés à PRODUIT, alors que les attributs Mois et Année sont associés à TEMPS. On peut voir dans cette figure que 100 unités du produit P1 ont été vendus au mois de Janvier 1996. La partie (b) de la même figure présente une instance de VENTES, qui est formée par les relations rPRODUIT, rTEMPS et rm. Chaque n-uplet des relations rPRODUIT et rTEMPS est vue comme une "coordonnée" dans les dimensions PRODUIT et TEMPS respectivement, alors qu'un n-uplet de rm représente la valeur associée à une combinaison de coordonnées, une coordonnée pour chaque dimension.

VENTES			TEMPS						
			Année	1996			1997		
			Mois	Jan	Fev	....	Jan	Fev	....
PRODUIT	Nom	Poids	Vente						
	P1	10		100	80		80	100	
		20		80	100		120	160	
		....		....	....		....	....	
	P2	10		50	60		50	60	
		20		60	80		70	75	
		....		....	....		....	....	

(a)

VENTES

rPRODUIT

Tid	Nom	Poids
t1	P1	10
t2	P1	20
t3	P2	10
t4	P2	20
....	....	....

rTEMPS

Tid	Année	Mois
t1	1996	Jan
t2	1996	Fev
t3	1997	Jan
t4	1997	Fev
....	....	....

rm

P.Tid	T.Tid	Quantité
t1	t1	100
t2	t1	80
....	....	....
t1	t2	80
t2	t2	100

(b)

FIG. A.3 – *Table N-Dimensionnelle de Gyssens et Lakshmanan. (a) Table bi-dimensionnelle (b) Instance de la table bi-dimensionnelle*

Cube

Pour Thomas et. al. [TDV97] une base de données multidimensionnelle est un ensemble de cubes, définis en termes de dimensions, mesures, attributs et fonctions qui associent les attributs aux dimensions. Il est possible de définir des schémas de cubes, qui spécifient de manière abstraite la structure d'un cube. À partir d'un schéma, des instances peuvent être créées. Les hiérarchies ne sont pas explicitement représentées et elles doivent être définies *a priori* comme un ordre partiel entre les attributs d'une dimension. La figure A.4 montre le cube (instance de cube) VENTES défini par les dimensions $D1$ (PRODUIT), $D2$ (MAGASIN) et $D3$ (TEMPS). L'attribut a_1 de la dimension PRODUIT est *Nom*, l'attribut a_1 de la dimension MAGASIN est *Ville* et l'attribut a_3 de la dimension TEMPS est *Année*.

Les opérateurs de l'algèbre proposée (cf. table A.1) prennent en entrée un cube et produisent en sortie un autre cube. Le modèle fournit des définitions simples des opérateurs algébriques, où chaque opérateur exécute une fonction unique. Les opérateurs peuvent être composés pour définir d'autres opérateurs comme Roll-Up et Drill-Down.

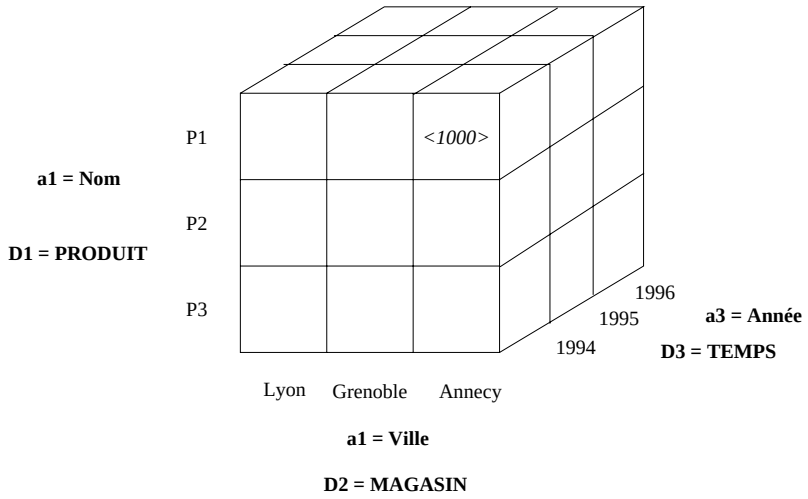


FIG. A.4 – *Le Cube de Thomas et al.*

F-tables

Cabibbo et Torlone [CT98] proposent une formalisme autour de la notion de *f-table*, qui est une fonction des coordonnées des dimensions aux mesures. Il est possible de spécifier le schéma d'une f-table et de définir, à partir de ce schéma, un ensemble d'instances. Ce formalisme autorise la structuration complexe des hiérarchies dans une dimension, en définissant des ordres partiels entre ses attributs. Le langage d'interrogation [CT97] est basé sur la logique du premier ordre et autorise donc l'écriture de requêtes de façon déclarative. Ce langage permet l'incorporation des fonctions d'agrégation mieux adaptés aux besoins du domaine d'application.

VENTES [v : ville <MAGASIN>,
p : nom <PRODUIT>,
j : jour <TEMPS>] : quantité

(a)

v	p	j	VENTES
Grenoble	P1	15 Mai 1996	100
Grenoble	P2	15 Mai 1996	80
Annecy	P1	15 Mai 1996	80
Lyon	P2	15 Mai 1996	60
Creteil	P1	15 Mai 1996	110
Paris	P2	15 Mai 1996	150

(b)

FIG. A.5 – *Les f-tables de Cabbibo et Torlone. (a) Schéma (b) Instance*

On peut voir dans la partie (a) de la figure A.5 le schéma de la *f-table* qui représente les ventes de la chaîne de magasins, mesurées en termes de la quantité vendue de produits. Les attributs v, p et j de ce schéma représentent, respectivement, la dimension MAGASIN au niveau ville, la dimension PRODUIT au niveau nom et la dimension TEMPS au niveau jour. La partie (b) de la figure A.5 montre une instance de ce schéma.

Annexe B

Grammaires

Grammaire de MDL

```
expression  ::= (createdim | alterdim | dropdim |
                    | renamedim | renamelevel |
                    | createcube | altercube | dropcube |
                    | renamecube | renamemeasure )";"
createdim   ::= <CREATE><DIMENSION><ID>(level)+
                (property)* (rollup)* (describedby)*
dropdim     ::= <DROP><DIMENSION><ID>
renamedim   ::= <RENAME><DIMENSION><ID><TO><ID>
renamelevel ::= <RENAME><LEVEL><ID><TO><ID>
alterdim    ::= <ALTER><DIMENSION><ID>
                (addlevel | dellevel | addprop | delprop )
addlevel    ::= <ADD>level rollup
dellevel    ::= <DELETE><LEVEL><ID>
addprop     ::= <ADD>property describedby
delprop     ::= <DELETE><PROPERTY><ID>
level       ::= <LEVEL><ID><COLON>type
property    ::= <PROPERTY><ID><COLON>type
rollup      ::= <ROLLUP><ID><COMMA><ID>
describedby ::= <DESCRIBEDBY><ID><COMMA><ID>
createcube  ::= <CREATE><CUBE><ID>(axis)+ (measure)+
dropcube    ::= <DROP><CUBE><ID>
renamecube  ::= <RENAME><CUBE><ID><TO><ID>
renamemeasure:= <RENAME><MEASURE><ID><TO><ID>
```

```
altercube   ::= <ALTER><CUBE><ID>
                (addaxis | delaxis | addmeasure | delmeasure)
addaxis     ::= <ADD>axis
delaxis     ::= <DELETE>axis
addmeasure  ::= <ADD>measure
delmeasure  ::= <DELETE><MEASURE><ID>
axis        ::= <AXIS><ID>
measure     ::= <MEASURE><ID><COLON>type
type        ::= <CHAR><LPAREN><DECIMALINT><RPAREN>|
                  <VARCHAR><LPAREN><DECIMALINT><RPAREN>|
                  <DECIMAL>| <SMALLINT>| <INTEGER>| <FLOAT>| <DATE>
```

Grammaire du sous-ensemble XML-QL

```
queryblock  ::= where (orderby)? construct
where       ::= <WHERE>condition (<COMMA>condition)*
condition   ::= (element <IN>datasource | predicate)
element     ::= starttag(<STRING>| literal | <VAR>| (element)+) endtag
            (<ELEMENT_AS><VAR>| <CONTENT_AS><VAR>)*
starttag    ::= <LT>(<VAR>| <ID>) (attribute)* <GT>
endtag      ::= <LTSLASH>(<VAR>| <ID>)? <GT>
attribute   ::= <ID><EQ>( <STRING>| <VAR>)
datasource  ::= <VAR>| <STRING>
predicate   ::= expression oprel expression
expression  ::= <VAR>| <STRING>| literal
oprel       ::= <LT>| <GT>| <LE>| <GE>| <EQ>| <NE>| <LIKE>
orderby     ::= <ORDER_BY><VAR>(<COMMA><VAR>)*
construct   ::= <CONSTRUCT>( result | <VAR>)
result      ::= starttag (<STRING>| literal | (<VAR>| result)+ | aggpred) endtag
aggpred     ::= <COUNT><LPAREN><STAR><RPAREN>
            | <SUM><LPAREN><VAR><RPAREN>
            | <MAX><LPAREN><VAR><RPAREN>
            | <MIN><LPAREN><VAR><RPAREN>
            | <AVG><LPAREN><VAR><RPAREN>
literal     ::= <INTEGER>| <FLOATING_POINT>| <CHARACTER>| <BOOLEAN>|<NULL>
```