N d'ordre 98 ISAL 0093 Année 1998

THESE

présentée devant

L'INSTITUT NATIONAL DES SCIENCES APPLIQUEES DE LYON

pour obtenir

LE GRADE DE DOCTEUR

FORMATION DOCTORALE : DEA Informatique de Lyon

PAR

Patrick MARCEL

Manipulations de Données Multidimensionnelles et Langages de Règles

Soutenue le 16 décembre 1998 devant la Commission d'Examen

Jury MM. N. Bidoit
P. Bosc
M. Bouzeghoub (rapporteur)
M.-S. Hacid
J. Kouloumdjian
D. Laurent (rapporteur)

Cette thèse a été préparée au Laboratoire d'Ingénierie des Systèmes d'Information de l'INSA de Lyon

Remerciements

Je souhaite exprimer ma sincère gratitude envers Jacques Kouloumdjian, Professeur à l'INSA de Lyon et co-responsable de ma thèse, qui m'a accueilli dans son équipe et a mis à ma disposition les moyens nécessaires pour mener à bien ce travail.

Je suis très reconnaissant à Mokrane Bouzeghoub, Professeur à l'Université de Versailles, et à Dominique Laurent, Professeur à l'Université de Tours, d'avoir été rapporteur de mon mémoire. Je les remercie vivement pour le travail qu'ils ont consacré à son évaluation.

Je remercie également Nicole Bidoit, Professeur à l'Université de Bordeaux, et Patrick Bosc, Professeur à l'ENSSAT de Lannion, qui m'ont fait l'honneur d'accepter de participer à mon jury de thèse.

Je tiens à souligner le mérite de Mohand-Saïd Hacid, Maître de conférences à l'Université Claude Bernard de Lyon, co-responsable de ma thèse, dont les qualités scientifiques et humaines ont contribué à créer un cadre réellement motivant et enrichissant.

Je souhaite remercier très chaleureusement Christophe Rigotti, Maître de conférences à l'INSA de Lyon, dont les qualités pédagogiques et la rigueur scientifique, mises au service de ce travail, ont grandement influencé son déroulement.

Je remercie également Jean-François Boulicaut, Maître de conférences à l'INSA de Lyon, pour son enthousiasme dans mon travail et sa contribution éclairée.

Je termine par un salut amical à tous les membres de l'équipe ECRIN, et à toutes les personnes qui m'ont supporté sur toute la durée de ce travail.

Table des matières

Table des figures

Chapitre I

Introduction

1 Problématique

Les bases de données ont permis d'informatiser le stockage et la gestion *transactionnelle* d'importants volumes d'information. Leur succès repose en grande partie sur une séparation entre la définition *logique* de l'information, et son implantation *physique*. La forme physique de l'information correspond à la manière concrète par laquelle elle est stockée. Par opposition, une représentation logique de l'information est une abstraction de son codage physique. Cette séparation (appelée *principe d'indépendance des données*) permet à l'utilisateur de s'affranchir d'une vision codée de l'information, pour se focaliser sur une représentation qui reflète sa vision de la réalité ; elle rend de plus possible le développement de *techniques de gestion et d'optimisation* transparentes pour l'utilisateur.

E. F. Codd [17] a précisé la séparation physique/logique en proposant le paradigme relationnel en tant que *modèle de données logique* de représentation de l'information. De manière informelle, le *modèle relationnel* repose sur la *logique du premier ordre*, où :

- l'information est structurée sous la forme d'*ensembles de n-uplets*[1] ;
- l'information est manipulée à l'aide de langages de haut niveau.

Le succès (académique et commercial) du modèle relationnel est en partie dû à cette définition mathématique. Ce modèle à pu être implanté efficacement et a constitué un cadre de recherches important pour le développement des bases de données.

Lors de son apparition, le modèle relationnel a parfaitement répondu aux besoins en gestion transactionnelle de l'information. Depuis, ces besoins ont évolué : l'accroissement des volumes de données a induit des changements dans la vision de l'information et suscité une sophistication des traitements. Plus précisément, on souhaite actuellement disposer d'un point de vue global pour manipuler des sources de données *hétérogènes*, sur lesquelles des traitements *analytiques* (traitements *OLAP*, pour "On-Line Analytical Processing") sont associés aux traitements transactionnels. Codd [18] propose que l'information soit représentée et manipulée *selon plusieurs dimensions* et à des *niveaux de granularité* différents.

Le modèle relationnel est sémantiquement pauvre (le seul moyen de représentation est la relation), et impropre à modéliser directement de l'information structurée de manière multidimensionnelle. Il en est de même pour les nombreux travaux qui ont proposé d'étendre ce modèle (e.g., à la prise en compte de *collections*), ou de le dépasser (e.g., le modèle objet), car aucun n'est motivé par les spécificités analytiques à mettre en oeuvre. Modéliser une information complexe dans un modèle de données logique non adapté à la structure de cette information entraine des codages non naturels et

1. Nous emploierons l'angliscisme "tuples" dans la suite de ce mémoire.

oblige le plus souvent à exprimer les manipulations de données spécifiques dans des langages externes, par exemple des langages de programmation [36]. Il devient nécessaire de trouver une représentation logique claire de cette *information multidimensionnelle*, tant pour faciliter sa manipulation par l'utilisateur, que pour amener les mécanismes de la base de données à tirer profit de sa structure.

L'objectif principal de ce travail est la proposition d'un niveau logique simple et intuitif pour bases de données intégrant traitements analytiques et information structurée de manière multidimensionnelle. Pour ce faire, nous proposons une progression selon trois points :

1. établir une grille d'étude synthétisant les formalisations de traitements analytiques de données multidimensionnelles ;
2. contribuer à l'étude de ces traitements par la proposition d'un langage défini formellement ;
3. montrer l'intérêt pratique du langage en présentant une application concrète.

2 Travail réalisé

La première partie de ce travail est un état de l'art original portant sur la manipulation de données multidimensionnelles à vocation analytique dans les bases de données. La contribution principale de cette synthèse est de proposer un recensement et une classification des opérations élémentaires propres aux domaines de l'analyse en ligne de données (les traitements *OLAP*). Notre but est de faire ressortir, via des exemples de traitements typiques, les spécificités des différentes approches formelles. Cette synthèse a été tenue à jour lors de l'avancement de nos travaux.

La deuxième partie du travail constitue une réponse aux conclusions de l'état de l'art, et propose, à travers les définitions formelles d'un langage, un unique formalisme enrichissant l'étude de ces manipulations et intégrant des aspects jusqu'alors disséminés sur plusieurs propositions. Son objectif initial est de montrer qu'un cadre simple inspiré de Datalog est suffisant pour spécifier les traitements complexes des systèmes OLAP. Nous nous sommes inspirés de techniques provenant des bases de données et de la programmation logique pour élaborer [25] puis étendre [27, 26, 28] un langage noyau. Au moment de leur élaboration, les langages décrits dans [25] et [27] étaient à notre connaissance les seuls langages non algébriques proposés pour le domaine OLAP. Ces propositions sont étayées par des définitions formelles (sémantique déclarative et contrepartie opérationnelle équivalente), l'études des liens avec le cadre Datalog classique, et l'illustration du langage sur des exemples de requêtes typiques.

La dernière partie de ce travail montre comment exprimer dans notre cadre la catégorie de traitements multidimensionnels que l'on trouve dans les interfaces de type tableur, utilisées fréquemment comme *front-ends* des systèmes OLAP [18]. Cette application originale peut aussi être vue comme une contribution à l'amélioration de la qualité des développements à base de tableurs [8]. En effet, la sémantique précise du langage a permis l'élaboration et l'implantation d'une technique de compilation de règles pour développer de façon quasi-automatique des feuilles de calcul [10].

Les trois chapitres qui structurent le corps de ce mémoire reflètent la démarche adoptée, et présentent les principaux résultats obtenus :

chapitre II la description et la classification des opérations élémentaires de l'analyse en ligne de données, et l'étude de leurs premières formalisations ;

chapitre III les caractéristiques de notre cadre formel enrichissant les travaux existants ;

chapitre IV l'application de notre cadre à la génération de feuilles de calcul.

La conclusion et les perspectives de ce travail sont présentées au chapitre V.

Ce travail a été effectué en liaison avec le projet européen 22469 "Foundations of Data Warehouse Quality (DWQ)". Le document [27] a notamment été joint aux délivrables d'octobre 1997 du projet.

Chapitre II

Synthèse

1 Introduction

Les bases de données ont été développées et utilisées pour des applications commerciales dont le but est de gérer d'importants volumes d'informations selon des *processus transactionnels en ligne* (*OLTP* pour "On-Line Transactional Processing"). Ces processus ont des propriétés spécifiques [14] :

- ils sont nombreux et principalement concernés par la mise à jour des données ;
- ils portent sur un nombre de tuples relativement restreint (de l'ordre de la dizaine) ;
- ils sont lancés de manière interactive par les utilisateurs ;
- ils sont exécutés de façon concurrente.

A la recherche d'abstractions permettant de détacher la manière de représenter ces données de leur implantation physique, E.F. Codd a proposé le modèle relationnel [17]. Ce modèle de données logique a permis de réconcilier les approches existantes, et a le mieux répondu aux attentes des applications commerciales. Il a de plus comblé le fossé entre la modélisation de données et la logique mathématique [4], et donc permis la mise en oeuvre de techniques (d'implantation, d'optimisation) pouvant être validées de façon formelle.

Si aujourd'hui les bases de données demeurent incontournables dans les entreprises, l'utilisation qui en est faite a évolué. Elles font désormais partie d'applications *d'aide à la décision*, qui sont par nature *semi-automatiques*. Codd ne parle d'ailleurs plus d'utilisateurs, mais d'utilisateurs-analystes [18]. Cette évolution a eu une incidence sur la façon d'envisager informations et traitements, car si les processus sont toujours exécutés de manière interactive, ils sont moins nombreux mais beaucoup plus complexes, et concernent l'exploration de volumes de données aux structures hétérogènes beaucoup plus importants (d'un facteur 1000 par rapport aux volumes de données concernés par les applications OLTP). Il n'est pas rare, par exemple, de vouloir mettre à profit les facilités analytiques du type de celles fournies par un tableur pour étudier des données provenant de téraoctets d'information historique [18, 14]. Cette nouvelle vision a engendré de nouveaux besoins, et a donné naissance aux concepts *d'entrepôt de données* et *d'analyse en ligne de données*.

Entrepôts de données Comme son nom l'indique, un entrepôt de données est un lieu (vaste !) de stockage d'informations. Sa constitution suppose la mise en commun de données de différentes structures, provenant de plusieurs sources. Son rôle est de centraliser des données pour en offrir une *représentation homogène globale*, directement exploitable [48].

Analyse en ligne de données Entre la vue construite de l'information offerte par l'entrepôt de données et les diverses interfaces mises à la disposition de l'analyste vient se greffer toute une catégorie de traitements nécessaires à la prise de décisions (figure II.1). Codd [18] propose de manière informelle une vision unifiée de ces nouveaux types de manipulations, en les regroupant sous le terme *OLAP*, pour "On-Line Analytical Processing", que nous traduisons par analyse en ligne de données [1].

Les traitements mis en oeuvre tiennent compte de l'aspect semi-automatique du processus d'aide à la décision en proposant des manipulations interactives, et ils répondent aux besoins spécifiques en analyse d'information. Ainsi, paraphrasant Codd, Agrawal et al. [7] décrivent ces traitements comme autant d'aptitudes à consolider, visualiser et synthétiser des informations reflétant la vision de l'analyste. Cette vision correspond à une structuration des données selon plusieurs dimensions, pouvant représenter des notions très variées telles que le temps ou la localisation géographique, mais aussi la valeur d'un chiffre d'affaires ou le code identifiant des produits. On parle alors de modélisation et de traitements *multidimensionnels* des données [14].

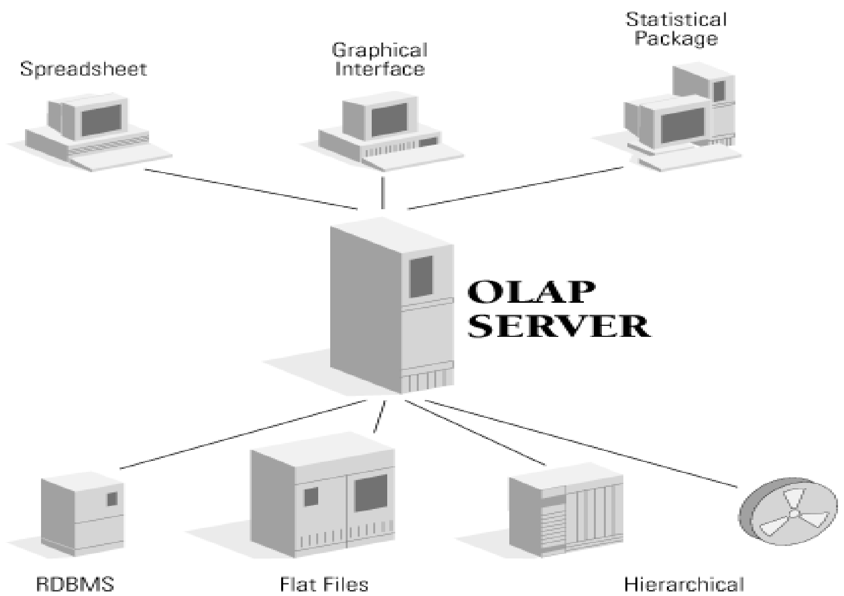

FIG. II.1 - *Rôle du serveur OLAP [18]*

Pour mieux comprendre cette nouvelle terminologie, nous proposons dans ce chapitre un état de l'art du domaine de la manipulation de don-

1. Dans la suite de ce mémoire, les termes OLAP et analyse en ligne de données sont utilisés indifféremment. On évitera la confusion avec le terme "analyse de données".

nées multidimensionnelles utilisée pour l'analyse en ligne de données. Nous structurons notre présentation de la façon suivante :

- en nous basant sur des exemples typiques, nous décrivons la vision de l'analyste et abordons informellement la modélisation de données multidimensionnelles. Nous illustrons alors les principes élémentaires de manipulation de données liés à cette modélisation (section 2) ;
- pour mieux comprendre les spécificités de la manipulation de données multidimensionnelles, nous étudions les façons dont celles-ci ont été formalisées. Ceci nous permet de dégager des pistes d'exploration intéressantes (section 3).

Nous adoptons cette démarche car elle a l'avantage de procéder par affinements successifs de concepts intuitifs, jusqu'à leurs formalisations. Elle respecte de plus la dimension historique : les concepts intuitifs et les systèmes ont existé avant que les notions de base soient formalisées.

2 Manipulations multidimensionnelles

Dans les domaines récents que sont l'analyse en ligne de données, les entrepôts de données et les bases de données multidimensionnelles, les concepts intuitifs se sont propagés rapidement, et n'ont pas toujours fait l'objet de caractérisations formelles. Comme le remarque [46], les termes utilisés pour les opérations OLAP sont plus descriptifs que formels et se prêtent parfois à plusieurs interprétations. En conséquence, les définitions sont souvent imprécises. Une contribution de cette synthèse est de proposer un recensement des manipulations élémentaires fondamentales[2] pour l'analyse en ligne de données, et d'en donner une catégorisation simple. Cette catégorisation servira de base à la présentation des propositions formelles de langages faite dans la section 3. Notre présentation s'appuie sur les travaux faisant état des besoins des utilisateurs-analystes ou décrivant des systèmes [18, 41, 21, 42], et sur des articles de recherche généralistes [14, 46]. Par soucis de clarté et d'objectivité, nous préférons employer tels quels certains termes anglais (sans les traduire). Lors de cette présentation, nous donnons les définitions qui nous semblent consensuelles, ou s'il y a lieu, nos propres définitions.

Comme souligné plus haut, l'analyse en ligne de données est indissociable de la perception de l'information qu'a l'analyste. Dans un premier temps, nous nous attachons à décrire les *spécificités* qu'induit cette perception, pour étudier ensuite leur influence sur la manipulation de donnée. Nous choisissons de présenter trois spécificités primordiales :

- tout d'abord la nature *multidimensionnelle* de l'information ;

2. On notera que les manipulations présentées sont basées sur leurs apparitions récurrentes dans la littérature, et ne sont pas censées recouvrir de manière exhaustive la totalité des traitements OLAP.

– ensuite, sa représentation à *différents niveaux de détail*;

– et enfin, sa manipulation ensembliste.

Pour chacune de ces catégories, nous présentons la manière typique de modéliser l'information, et les *opérations élémentaires* de manipulation qui s'y rattachent. Les deux premières catégories sont liées à l'aspect "analyse". La troisième catégorie est motivée par la prise en compte des facilités transactionnelles inhérentes à la manipulation de données. Les opérations qui s'y rattachent correspondent aux opérations relationnelles de manipulation de données.

2.1 Structure multidimensionnelle

2.1.1 Intuitions

L'*efficacité* de l'analyse basée sur des traitements OLAP est directement reliée à la modélisation de l'information sous une forme conceptuelle proche de la perception qu'en a l'analyste [14, 12]. L'intuition simple à la base de cette modélisation est de considérer une donnée multidimensionnelle comme un point dans un espace à plusieurs dimensions. Dans ce contexte, le terme *dimension* mérite d'être précisé, et [12] en suggère la définition suivante.

Définition 2.1 Une *dimension* est une catégorie linguistique utilisée pour caractériser la structure des données du point de vue de l'analyste. □

La modélisation simpliste sous forme de relation, devient inappropriée. Considérons la relation décrite dans la figure II.2, qui représente des quantités de pièces vendus en 1996 dans quatre régions.

ventes 1996	pièces	régions	quantités
	écrous	est	50
	écrous	ouest	60
	écrous	sud	40
	clous	est	70
	clous	nord	40
	vis	ouest	50
	vis	sud	50
	vis	nord	60

FIG. II.2 - *La relation* ventes 1996

On peut discerner plusieurs catégories linguistiques dans ces données, chacune correspondant à un *attribut* (intitulé de colonne) de la relation : une dimension relative aux pièces, une dimension relative aux régions et une dimension relative aux quantités. Ainsi, comme le montre la figure II.3, les données de la relation peuvent être décrites dans cet espace à trois dimensions.

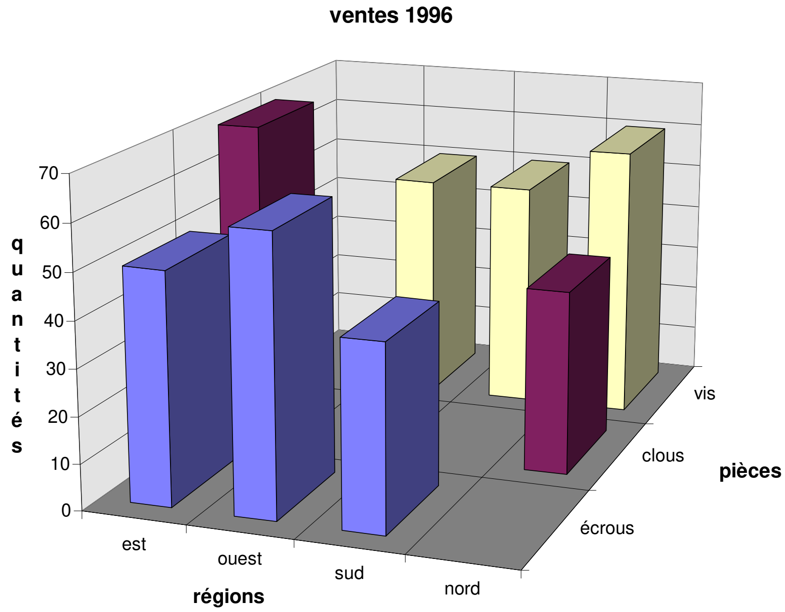

FIG. II.3 - *Représentation multidimensionnelle*

Si l'analyste cherche à représenter les données de la relation de la figure II.2 sous une forme reflétant mieux leur caractère multidimensionnel, il peut aussi choisir de les ranger dans une table selon deux dimensions, par exemple celle des pièces en ligne, et celle des régions en colonnes. Cette représentation, sous forme de *tableau croisé*, est décrite dans la figure II.4.

Pour construire cette représentation tabulaire à partir de la relation *ventes 1996*, on remarque que les tuples sont éclatés pour former les noms de lignes, les noms de colonnes et les valeurs aux intersections de chaque ligne avec chaque colonne. On notera qu'une table permet de représenter toutes les combinaisons des valeurs choisies pour constituer les noms de lignes et les noms de colonnes. Or ces combinaisons peuvent ne pas être connues (e.g., les ventes de vis dans la région est), et il faut trouver un moyen de représenter cette absence (e.g., par une case vide sur la figure II.4).

La métaphore qui s'est rapidement imposée comme généralisation de cette vision est celle du *cube*[3], qui reflète par sa structure la nature multi-

3. Dans la littérature, les termes *cube, hypercube et table multidimensionnelle* sont utilisés de manière interchangeable. Nous proposons dans cette section de relier les termes "cube" et "table" à la représentation de l'information. Ainsi, "table" sera employé pour des représentations bidimensionnelles, et "cube" pour des représentations tridimensionnelles. Nous réservons le terme "hypercube" pour des structures à plus de 3 dimensions.

ventes 1996	est	ouest	sud	nord
écrous	50	60	40	
clous	70			40
vis		50	50	60

FIG. II.4 - *La table bidimensionnelle* ventes 1996

dimensionnelle de l'information.

La figure II.5 montre un cube basé sur un exemple récurrent de la littérature OLAP. On peut considérer qu'il s'agit de l'extension de la table de la figure II.4 à la prise en compte de la dimension des années. La table et le cube présentés sur ces figures nous serviront d'exemples pour notre présentation.

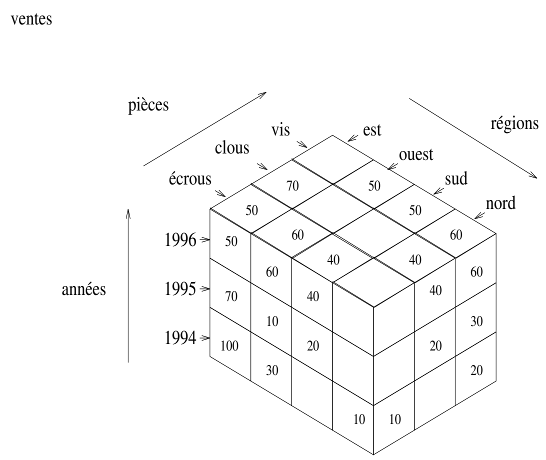

FIG. II.5 - *Le cube* ventes

Un cube est composé d'éléments appelés *cellules*, une cellule pouvant contenir une ou plusieurs valeurs appelées *mesures*. Ces cellules sont repérées par des axes pouvant être nommés. Chaque axe correspond à une dimension, et est gradué par des valeurs appelées *paramètres* ou *membres*. La position d'une cellule dans un cube, donnée par une graduation prise sur chaque axe (i.e., une valeur de membre pour chaque dimension) est appelée *référence*. Une référence identifie donc de manière unique une cellule, ce qui engendre une dépendance fonctionnelle entre les références de cellules et les contenus.

2.1.2 Opérations liées à la structure

Les opérations agissant sur cette structure multidimensionnelle de l'information sont motivées par l'aspect interactif de l'analyse en ligne de données, et le soucis d'offrir des possibilités d'animation de la représentation. Elles illustrent de plus l'importance des liens entre la manipulation des données et la représentation du cube à l'écran. Elles sont particulièrement utiles pour spécifier des représentations bidimensionnelles ou tridimensionnelles d'hypercubes. Si l'on s'en tient à la métaphore du cube, une description imagée de ces traitements consiste à faire pivoter un cube, le couper en tranches, interchanger ou combiner coordonnées et/ou contenus.

Nous choisissons de regrouper ces opérations sous le nom d'opérations de *restructuration*, car elles permettent de manière *relative*[4], de réorienter la vue multidimensionnelle ou d'en changer l'agencement, en jouant sur les *positions des membres et des mesures.* De manière plus précise :

Définition 2.2 Nous appelons opération de restructuration, ou simplement *restructuration*, toute opération élémentaire bijective de changement de point de vue sur un cube. □

Ainsi, tout cube obtenu par restructuration d'un cube initial contient les informations nécessaires et suffisantes pour régénérer le cube initial par restructuration réciproque. Nous illustrons ces opérations de restructuration sur le cube de la figure II.5, en faisant ressortir les concepts importants qu'elles suggèrent (e.g., l'incidence sur le nombre de dimensions requises pour présenter le résultat).

"Rotate" La première opération élémentaire que nous abordons est connue sous le nom de "pivot" ou "rotate", et consiste à faire effectuer à un cube une rotation autour d'un des trois axes passant par le centre de deux faces opposées, de manière à présenter un ensemble de faces différent. La figure II.6 en donne une idée, en partant du cube de la figure II.5 que l'on fait pivoter autour d'un des axes horizontaux.

Cette opération est typiquement tridimensionnelle, et n'a pas d'incidence sur le nombre de dimensions de la représentation du cube résultant. Elle est motivée par la représentation adoptée pour le cube, qui s'attache à rendre l'aspect tridimensionnel d'un cube dans un plan. Seules trois faces étant représentées, l'opération "rotate" s'apparente à une sorte d'opération de sélection qui ne serait pas guidée par un choix de membres, mais par un choix de faces.

"Switch" Une deuxième opération, appelée "switch", consiste à interchanger la position des membres d'une dimension, comme le montre la figure II.7 sur le cube de la figure II.5 pour "nord" et "sud".

4. i.e., en fonction d'un état précédent.

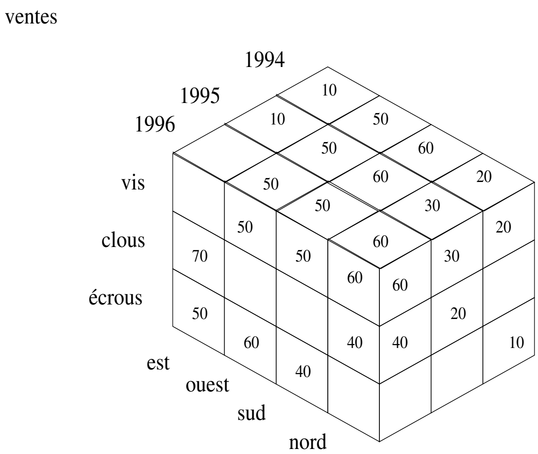

FIG. II.6 - *"Rotate"*

"Split" L'opération suivante consiste à présenter chaque "tranche" du cube, et de passer d'une représentation tridimensionnelle d'un cube à sa représentation sous la forme d'un ensemble de tables. La figure II.8 en donne un aperçu, pour un découpage selon les régions du cube de la figure II.5. De manière générale, cette opération permet de réduire le nombre de dimensions d'une représentation. Elle est appelée "split", et sa généralisation permet par exemple de découper un hypercube de dimension 4 en cubes. On notera que le nombre de cubes ou tables résultant d'un "split" dépend d'informations contenues dans le cubes de départ (e.g., le nombre de membres d'une dimension) et n'est pas connu à l'avance. L'opération réciproque n'a pas d'appellation établie dans la littérature OLAP, et permet par exemple de recomposer la dimension des régions du cube de la figure II.5 à partir des tables de la figure II.8.

"Nest" L'opération suivante, appelée "nest", permet d'imbriquer des membres, comme le montre la figure II.9 à partir du cube de la figure II.5. L'un des intérêts de cette opération est qu'elle permet de grouper sur une même représentation bidimensionnelle toutes les informations (mesures et membres) d'un cube ou hypercube, quelque soit le nombre de ses dimensions. L'opération réciproque, "unnest", reconstitue une dimension séparée à partir des membres imbriqués.

"Push" La dernière opération de restructuration consiste à combiner les membres d'une dimension aux mesures du cube, et donc de faire passer des membres comme contenus de cellules. Elle est appelée "push". La figure II.10 en présente un exemple, à partir du cube de la figure II.5 sur lequel on effectue un "push" des années.

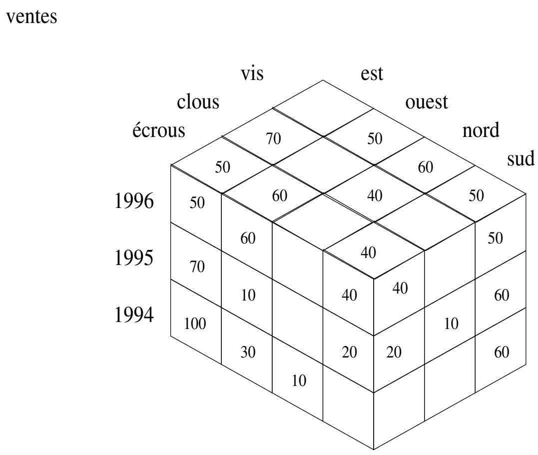

FIG. II.7 - *"Switch"*

ventes est	1996	1995	1994
écrous	50	70	100
vis		10	10
clous	70	50	40

ventes ouest	1996	1995	1994
écrous	60	10	30
vis	50	50	50
clous		10	40

ventes sud	1996	1995	1994
écrous	40	20	
vis	50	60	60
clous		10	

ventes nord	1996	1995	1994
écrous			10
vis	60	30	20
clous	40	20	

FIG. II.8 - *"Split"*

L'opération réciproque, appelée "pull", permet de changer le statut de certaines mesures d'un cube en membres, et de constituer une nouvelle dimension pour la représentation du cube, à partir de ces nouveaux membres. Par exemple, pour reconstruire la représentation de la figure II.5 à partir de celle de la figure II.10, on effectue un "pull" des années. On constate que ces deux dernières opérations permettent de traiter de façon similaire mesures et membres.

Un cas particulier de l'opérations "pull" se présente lorsqu'au plus une mesure est contenue dans chaque cellule. Dans ce cas, le contenu de chaque cellule du cube résultant est utilisé pour représenter la validité des combinaisons des membres entre eux, et donc différencier les combinaisons valides des combinaisons non valides. La figure II.11 montre le résultat d'un "pull" des ventes appliquée à la table de la figure II.4, où les cellules grisées représentent des combinaisons valides, et les cellules blanches des combinaisons

ventes		1996	1995	1994
écrous	est	50	70	100
	ouest	60	10	30
	nord			10
	sud	40	20	
vis	est		10	10
	ouest	50	50	50
	nord	60	30	20
	sud	50	60	60
clous	est	70	50	40
	ouest		10	40
	nord	40	20	
	sud		10	

FIG. II.9 - *"Nest"*

ventes	est	ouest	nord	sud
écrous	1996 50 1995 70 1994 100	1996 60 1995 10 1994 30	1994 10	1996 40 1995 20
vis	1995 10 1994 10	1996 50 1995 50 1994 50	1996 60 1995 30 1994 20	1996 50 1995 60 1994 60
clous	1996 70 1995 50 1994 40	1995 10 1994 40	1996 40 1995 20	1995 10

FIG. II.10 - *"Push"*

non valides.

2.2 Granularité

2.2.1 Intuitions

Le deuxième aspect de la vision de l'analyste est de *hiérarchiser* l'information en différents niveaux de détails, appelés niveaux de granularité dans la littérature OLAP. Un niveau est un ensemble nommé de membres. En général, le niveau le plus bas est celui de l'entrepôt de données. Des opérations d'agrégation successives sur ces données (via des sommations ou des calcul de moyenne, par exemple) offrent de nouveaux points de vue de moins en moins détaillés de l'information, et constituent autant de niveaux supérieurs. Ces niveaux de granularité sont décrits au moyen d'un graphe,

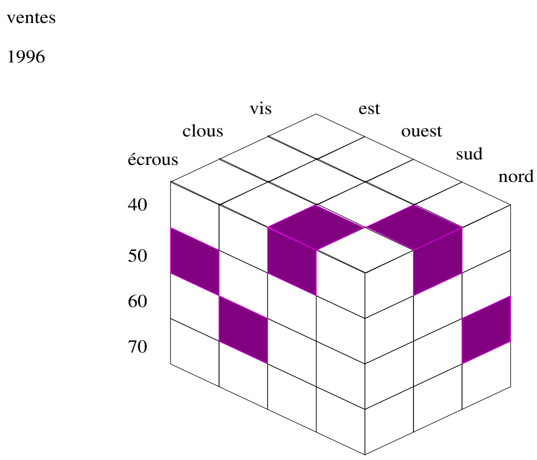

FIG. II.11 - *Cube sans mesure*

ou d'un ensemble de graphes, qui indique comment les membres d'un niveau sont regroupés pour former un membre d'un niveau supérieur. Le plus souvent, chaque graphe est associé à une dimension, et représente une *hiérarchie*. Chaque niveau de la hiérarchie correspond à une *agrégation* des données associées au niveau immédiatement inférieur.

Considérons la figure II.12. Trois graphes décrivent les différents niveaux de granularité des trois dimensions du cube de la figure II.5. Les noms de dimensions et les noms de niveaux apparaissent en caractère gras. L'usage est de donner le nom de la dimension au plus haut niveau de la hiérarchie. Ainsi, le premier graphe nous apprend que les données décrites au niveau des 4 régions (identifiées par les membres "est", "ouest", "sud", "nord") sont obtenues à partir de données décrites au niveau inférieur des villes (identifiées par les membres "lyon", ..., "lille"). Les pièces et les années font respectivement l'objet d'un seul regroupement.

2.2.2 Opérations associées à la granularité

Les opérations associées aux hiérarchies spécifient la navigation de façon relative entre les différents niveaux.

"Roll-up" La première opération consiste à représenter les données d'un cube à un niveau de granularité supérieur selon une dimension. On parle de "roll-up" d'un niveau à un autre. L'opération de "roll-up" est paramétrée par une fonction d'agrégat (e.g., somme, moyenne) indiquant comment calculer les données d'un niveau à partir du niveau immédiatement inférieur (dans nos exemples, nous utiliserons par défaut la fonction somme). Par exemple, la figure II.13 présente le cube de la figure II.5 sur lequel un "roll-

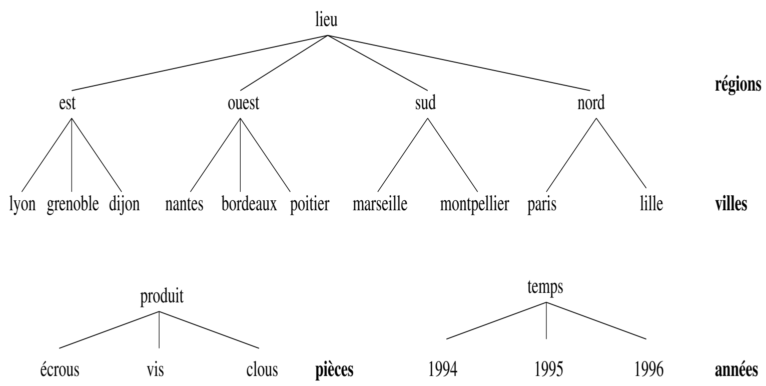

FIG. II.12 - *Les hiérarchies des dimensions*

up" a été effectué sur la hiérarchie des années, depuis le niveau détaillant les années jusqu'au niveau résumant l'ensemble des ventes pour toutes les années (le niveau le plus haut), en sommant les ventes de toutes les années pour chaque pièce et chaque région.

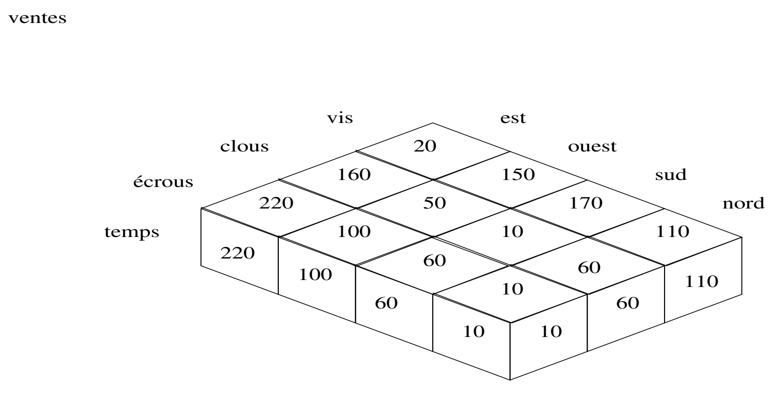

FIG. II.13 - *"Roll-up"*

Remarquons que cette opération est indépendante de la manière de présenter les données. Ainsi, on peut vouloir représenter les données de différents niveaux sur le même cube, comme le montre la figure II.14, où l'unique membre du niveau le plus haut de la hiérarchie *temps* est représenté sur le même axe que les membres du niveau des années.

L'opération qui décrit la consolidation d'un cube (i.e., calcul de tous les agrégats suivants tous les niveaux de toutes les dimensions) s'appelle "data cube" [22], et peut être vue comme la généralisation de l'opération "roll-up". La figure II.15 montre le résultat d'un telle opération appliquée au cube de la figure II.5 (e.g., la somme des ventes en 1996 pour toutes les pièces et

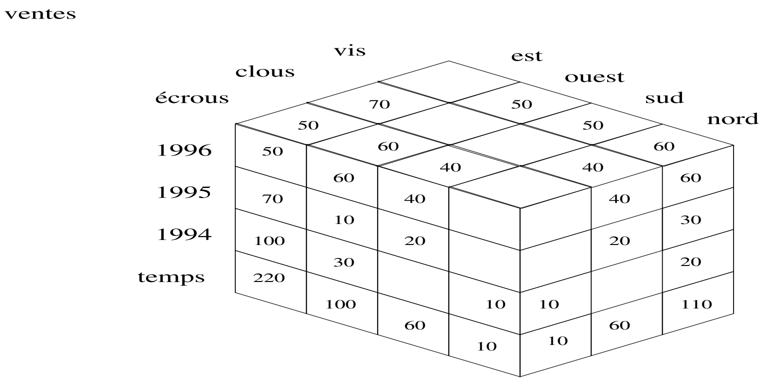

FIG. II.14 - *Représentation cubique du "roll-up"*

toutes les régions est de 420).

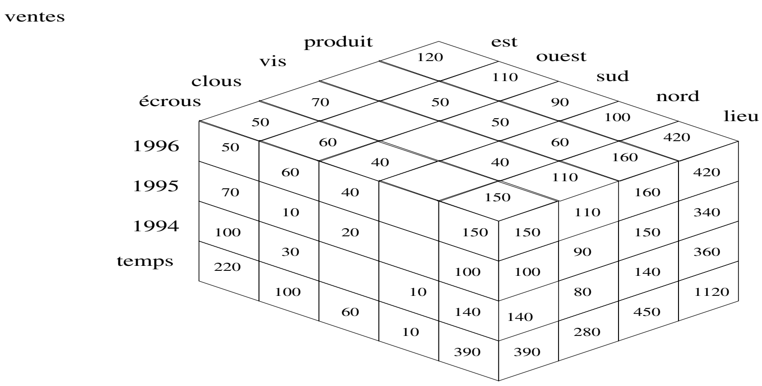

FIG. II.15 - *Consolidation du cube* ventes

"Drill-down" Une deuxième opération, appelée "drill-down", consiste à représenter les données d'un cube à un niveau inférieur, et donc sous une forme plus détaillée. Elle peut être vue comme l'opération réciproque du "roll-up". Ainsi, un "drill-down" sur la table de la figure II.5 du niveau des régions au niveau des villes permet d'obtenir le cube de la figure II.16.

On remarque que contrairement aux opérations de restructuration, les opérations "roll-up" et "drill-down" nécessitent des informations non contenues dans le cube pour passer d'une représentation initiale à une représentation de granularité différente. Par exemple, un "roll-up" nécessite de connaître la fonction d'agrégat appliquée, et un "drill-down" nécessite la connaissance des données du niveau inférieur, ou la connaissance d'une distribution des données du niveau courant sur le niveau inférieur.

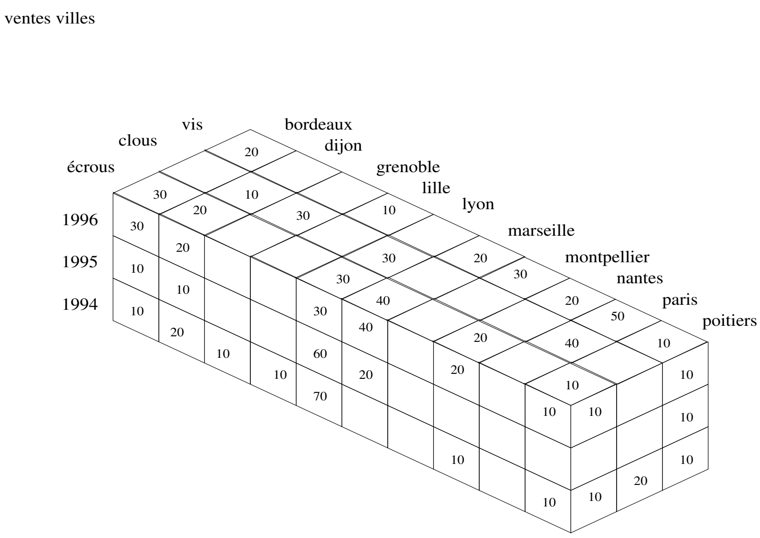

FIG. II.16 - *Le cube* ventes villes

2.3 Adaptation de la modélisation relationnelle

2.3.1 Intuitions

La modélisation relationnelle propose de différencier *l'instance* d'une relation (c'est à dire la liste des tuples composant cette relation) de son *schéma* (sa structure). Cette dissociation amène à regrouper en deux ensembles distincts les constantes utilisées pour modéliser une base de données relationnelle : on trouve d'une part l'ensemble des *méta-données*, qui contient les noms de relations et les noms d'attributs, et d'autre part l'ensemble des *données* qui sont les valeurs stockées sous forme de tuples. Ces méta-informations sont utilisées par les langages de manipulation relationnels pour l'extraction d'informations, ou à la combinaison des informations de deux relations.

Si on se réfère à la table de la figure II.4 construite à partir de la relation de la figure II.2, l'adaptation de cette dissociation aux informations multidimensionnelles consiste à voir d'une part les noms de cubes, les noms de dimensions, et les noms de niveaux comme des méta-données, et d'autre part les membres et les mesures comme des données.

Nous envisageons maintenant l'extension des opérations de manipulation de données telles qu'on les trouve dans le cadre relationnel[5], où elles sont bien connues et définies, à la prise en compte de l'information multidimensionnelle. Si l'on s'en tient à la vision imagée des données sous forme de cubes, la correspondance de ces opérations s'exprime en terme d'extraction

5. Un rappel de l'algèbre relationnelle est fait en annexe.

de sous-cubes, de combinaison de cubes entre eux, mais aussi en terme de formation de groupements, de spécification de niveau et de classification.

2.3.2 Adaptation des manipulations classiques

Sélection La version relationnelle de l'opération de sélection permet de ne considérer qu'un sous-ensemble des données d'une relation, en spécifiant sur les données des conditions d'appartenance à ce sous-ensemble. Ce principe appliqué à un cube consiste à définir un sous-ensemble de ses données. Deux cas se présentent alors : la sélection porte sur les mesures du cubes, ou sur ses membres. La figure II.17 montre un exemple de sélection sur le cube de la figure II.5 de manière à ne retenir que les quantités vendues supérieures ou égales à 50. La figure II.18 montre un exemple de sélection sur le cube de la figure II.5 de manière à ne retenir que les quantités de clous et d'écrous vendues en 1996 et 1995 dans les régions nord et sud. C'est aussi une sélection qui permet d'extraire du cube de la figure II.5 la table de la figure II.4.

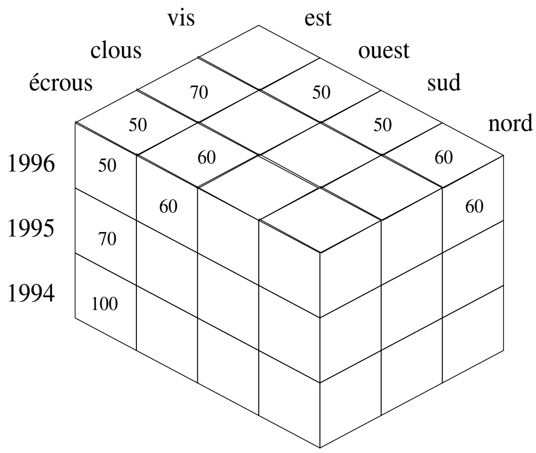

Fig. II.17 - *Sélection sur les mesures*

Projection En algèbre relationnelle, l'opération de *projection* vise à restreindre l'ensemble des méta-données (les attributs) d'une relation. Etendue à un cube, cette opération porte donc soit sur les dimensions, soit sur les niveaux de granularité.

Dans le premier cas, elle se rapproche d'une opération de projection au sens géométrique : un cube de dimension n est projeté sur $n-p$ dimensions, $1 \leq p \leq n$. Or le cube de $n-p$ dimensions résultant d'une telle projection doit conserver l'unicité de l'association référence-contenu des cellules. Ce qui se produit par exemple dans le cas où les p dimensions exclues ne possèdent

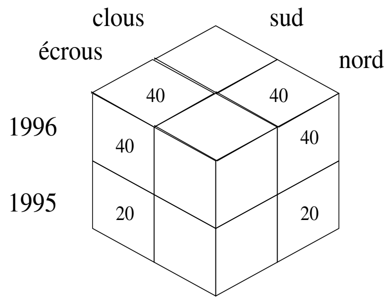

FIG. II.18 - *Sélection sur les membres*

chacune qu'un seul membre, ou bien lorsque les membres du cube sont agrégés au niveau le plus haut dans la hiérarchie de chacune des p dimensions. Ces p dimensions ne sont alors plus utiles pour la représentation du résultat, comme le montre la figure II.19 dans le cas du cube de la figure II.5 ayant subit une opération de projection sur les dimensions *lieu* et *produit*, les mesures ayant été agrégées selon la dimension *temps* (avec un principe similaire à une opération de "roll-up").

ventes 94-96	est	ouest	sud	nord
écrous	220	100	60	10
clous	160	50	10	60
vis	20	150	170	110

FIG. II.19 - *Une projection sur les dimensions* lieu *et* produit

Dans le deuxième cas, si un cube contient des données de différents niveaux d'une même hiérarchie, la projection sur un niveau permet de représenter le cube seulement à un niveau choisi. Au contraire du premier cas, le nombre de dimensions utile pour la représentation ne change pas. Ainsi le cube de la figure II.14 peut être obtenu à partir du cube de la figure II.13 en projetant au plus haut niveau de la dimension "temps". On notera que la combinaison des opérations de sélection et de projection est souvent regroupée sous le terme "slice-and-dice" dans la littérature OLAP.

Jointure La *jointure* naturelle relationnelle permet d'associer les données de relations différentes ayant certaines méta-données en commun. Nous illustrons l'extension de cette opération sur les tables des figures II.4 et II.20, pour une jointure selon la dimension des pièces. Le but de cette opération est ici de représenter sur une même table la quantité de pièces vendue et leur prix unitaire. Le résultat est décrit dans la figure II.21.

prix	1996
écrous	1
clous	0.7
vis	0.8

FIG. II.20 - *La table* prix

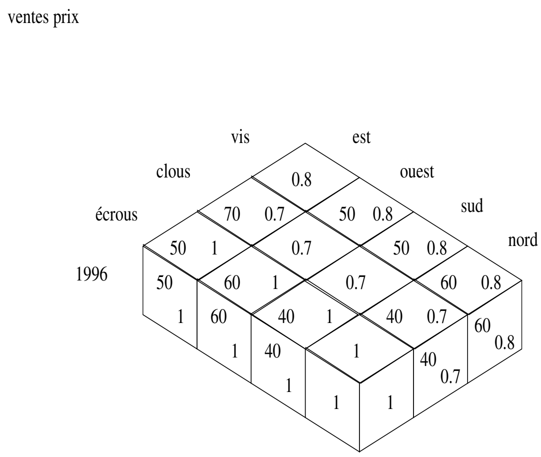

FIG. II.21 - *Jointure*

Autres opérations Les autres opérations classiques de manipulation de données sont l'opération de renommage et les opérations ensemblistes d'union, d'intersection et de différences. Dans le cas de l'algèbre relationnel, les opérations ensemblistes agissent sur des relations ayant le même ensemble de méta-données. L'extension de ces opérations pour un modèle multidimensionnel est triviale. Nous donnons simplement une illustration de l'opération d'union de deux tables bidimensionnelles. La table de la figure II.23 représente l'union de la table de la figure II.22 avec celle de la figure II.4. On prendra garde à ne pas confondre l'opération d'union, qui n'influe pas sur le nombre de dimensions de la représentation, avec l'opération réciproque du "split" rencontrée plus haut.

autres ventes1996	est	ouest	sud	nord
écrous				50
clous		60	60	
vis	70			
boulons		10	10	

FIG. II.22 - *La table* autres ventes1996

ventes 1996 complètes	est	ouest	sud	nord
écrous	50	60	40	50
clous	70	60	60	40
vis	70	50	50	60
boulons		10	10	

FIG. II.23 - *Union*

Nous présentons maintenant la mise en oeuvre de ces opérations élémentaires au travers d'exemples de traitements typiques.

2.4 Exploitation des opérations élémentaires

Les idées de base de l'analyse en ligne de données ont rapidement donné lieu à l'implantation de systèmes. Codd et al. [18] proposent douze règles destinées à juger si un produit implémentant les fonctionnalités OLAP répond effectivement aux besoins des utilisateurs-analystes. Ayant des préoccupations similaires, les rédacteurs du rapport OLAP [41] font une critique constructive de ces règles et affinent les critères de jugement. Ces critères, dont le but initial est l'évaluation de systèmes, nous servent pour présenter les aspects pratiques de la modélisation et la manipulation de données dédiée à l'analyse en ligne. Pour cette présentation, nous nous appuyons sur la description des opérations élémentaires faite dans la section précédente. Les requêtes présentées dans cette section ont été choisies pour leur caractère typique. Elles se réfèrent aux cubes et tables décrits plus haut, et seront utilisées dans la section 3 pour illustrer les manipulations de données permises par les langages formels actuels pour OLAP.

2.4.1 Modélisation conceptuelle

Le critère primordial proposé par Codd est que la vue conceptuelle offerte à l'utilisateur-analyste reflète l'aspect multidimensionnel des données, et permette la définition d'un nombre suffisant [6] de dimensions et de niveaux d'agrégation.

Les dimensions doivent être "équivalentes" (*symétriques*), tant au niveau de la structure qu'au niveau des capacités opérationnelles. En effet, comme illustré lors de la présentation des opérations "push" et "pull", chaque donnée d'une dimension peut apparaître comme membre ou comme mesure, et à ce titre aucun des deux statuts ne doit être privilégié. On dit que membres et mesures doivent être traités *symétriquement*. Ainsi, toute dimension peut faire l'objet d'agrégation sur les données (mesures ou membres) qui la com-

6. par rapport au domaine d'application (la modélisation ne devant pas être limitée par des restrictions dues au produit implantant les spécificités OLAP).

posent : on peut par exemple vouloir compter les membres d'un niveau d'une dimension, comme le montre la requête 2.1.

(2.1) *Quel est le nombre de villes où il a été vendu moins de 20 unités de l'une des 3 pièces en 1996 ?*

Concernant la granularité, certains traitements, comme celui illustré par la requête 2.2, requièrent la définition de multiples hiérarchies par dimension, ou de hiérarchies dites "en diamant" (un exemple est donné dans la figure II.24). Ces hiérarchies modélisent plusieurs possibilités de groupement. [34] recense des conditions pour obtenir des consolidations pertinentes et donne des éléments de méthode quant à la conception des hiérarchies.

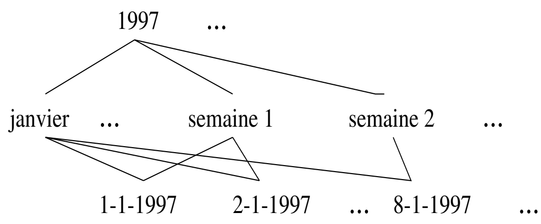

FIG. II.24 - *Exemples de hiérarchie particulière*

(2.2) *Calculer le total des ventes par mois et par semaines.*

Chaque dimension doit pouvoir comprendre des membres ou des mesures calculés à partir d'autres données pouvant provenir du même niveau, comme l'illustre la requête 2.3. Ces membres ou ces mesures sont appelés "calculated members" ou "calculated measures".

(2.3) *Calculer le chiffre d'affaires réalisé par pièce, région et année, en multipliant le prix de ventes par le nombre d'articles vendus. On désire faire apparaître ce chiffre comme mesure dans le cube ventes.*

2.4.2 Manipulations

L'interface offerte à l'utilisateur reflète la vision conceptuelle de l'information, et selon Codd [18], elle doit comporter des capacités

- de représentation synthétique et flexible, et
- de manipulation et d'animation du modèle de données.

Ces capacités portent sur toutes les différentes sources de données intégrées (*condition d'accessibilité*), sans toutefois que l'utilisateur ait à se préoccuper de la provenance des informations (*condition de transparence*).

Pour Codd et les auteurs du rapport OLAP, un modèle multidimensionnel est manipulable de manière plus intuitive qu'un modèle monodimensionnel, et cette manipulation suppose la définition de traitements complexes, nécessitant différentes combinaisons des opérations élémentaires. C'est le cas lors de la spécification de changements de point de vue sophistiqués basés sur la métaphore du cube. La requête 2.4 en est un exemple (les tables à obtenir sont représentées dans la figure II.25).

(2.4) *Présenter les informations du cube ventes concernant les vis et les clous, sous forme de tableau croisé, les années en ligne, les pièces imbriqués avec les régions en colonne. Séparer le résultat en deux tables selon que les quantités vendues sont inférieures ou supérieures à 30.*

ventes	est	ouest	sud	nord	
inf30	vis	clous	clous	vis	clous
1995	10	10	10	30	20
1994	10			20	

ventes	est	ouest		sud	nord	
sup30	clous	vis	clous	vis	vis	clous
1996	70	50		50	60	40
1995	50	50		60		
1994	40	50	40	60		

FIG. II.25 - *Les tables* ventes inf30 *et* ventes sup30

La spécification des manipulations doit faire preuve de flexibilité par rapport au modèle de données utilisé. On veut pouvoir spécifier des traitements quelque soit la dimension, ou le niveau de granularité, comme le montre la requête 2.5.

(2.5) *Présenter la quantité totale de pièces vendues pour la période 1994-1996 dans chaque ville.*

La flexibilité concerne aussi l'aspect granulaire. Les groupements (en vue d'agréger les données) peuvent être définis de manière *statique*, en fixant la hiérarchie associée à chaque dimension, ou *dynamique*, c'est à dire par requête. Une définition dynamique des groupements permet de spécifier des agrégats autres que ceux prédéfinis par la construction des hiérarchies, pour des traitements ponctuels (*agrégats ad-hoc*). Cette possibilité est illustrée par la requête 2.6 où l'année 1995 est présente dans les deux groupes constitués.

(2.6) *Calculer la moyenne glissante des ventes par région et par pièce, pour une fenêtre de deux années consécutives.*

Enfin, un dernier traitement typique dans le domaine de l'aide à la décision est la spécification d'estimations, comme l'indique la requête 2.7.

(2.7) *Calculer les prévisions de ventes pour les années 1997 à 2000, avec comme hypothèse un accroissement annuel des ventes de 10 pourcent.*

Si [18] met l'accent sur la nécessité de fournir des outils de manipulation intuitifs, [12] remarque que plusieurs langages de manipulation de données à différents niveaux d'abstraction seront nécessaires. D'un côté, l'utilisateur final doit trouver une interface conviviale à partir de laquelle il manipulera par "clic-souris" des métaphores graphiques. D'autre part, l'utilisateur avisé désireux de spécifier des traitements complexes privilégiera un langage déclaratif de haut niveau. Cette idée est aussi suggérée par les auteurs du rapport OLAP [41]. Nous étudions dans la section suivante les tentatives de définition d'un modèle de données et d'un langage de manipulation de données spécifiques.

3 Langages formels pour OLAP

3.1 Bases de données et informations multidimensionnelles

L'analyse en ligne de données n'est pas sans rappeler le cas des bases de données avant l'avènement du modèle relationnel : les concepts et les systèmes existent, mais sans fondement théorique stable. Une conséquence est que dans les systèmes OLAP actuels, l'interaction se fait principalement à l'aide d'extensions ad-hoc de SQL, et n'est généralement possible que sous la forme d'une seule opération à la fois. Il n'existe pas de véritable langage permettant de spécifier des combinaisons d'opérations [24, 23]. De fait, les langages conçus pour spécifier les traitements liés à l'analyse en ligne de données ne se trouvent ni dans les SGBD transactionnels existants, ni dans les outils d'analyse actuels (e.g., tableurs) [18, 14]. Ces constatations motivent la proposition de nouveaux langages *formels* dédiés aux manipulations de données multidimensionnelles à vocation analytique.

Des travaux ont déjà permi d'étendre le modèle relationnel à la prise en compte d'information multidimensionnelle dans des domaines autres qu'analytique, (e.g., les bases de données spatiales, temporelles ou statistiques). Si, comme le suggèrent [7, 46] l'utilisation de ces extensions est envisageable sur certains points précis (e.g., les structures d'indexation multidimensionnelle des bases de données spatiales, ou le calcul des vues avec agrégations des bases de données statistiques), elle n'est pas souhaitable dans le cas des modèles et langages de manipulation de données car ceux-ci dépendent de la nature de l'information à manipuler, et doivent prendre en compte des opérations spécifiques (e.g., du type de celles offertes par les tableurs) [14]. Une possibilité est alors d'envisager une extension au modèle relationnel vers la prise en compte de traitements analytiques sur de l'information multidimensionnelle implantée dans des relations (on parle de *ROLAP* pour "Relational OLAP"). Par opposition, on peut aussi envisager la conception d'un système où les opérations analytiques sont directement implantées sur des structures spécifiques (e.g., des tableaux multidimensionnels) stockant l'information (on parle de *MOLAP* pour "Multidimensional OLAP").

En nous appuyant sur la description des informations multidimensionnelles et de leurs manipulations faite dans la section 2, nous détaillons les propositions formelles actuelles de modèles de données et de langages de manipulation dédiés à l'analyse en ligne.

Ces propositions sont suffisamment hétérogènes et peu nombreuses pour que nous puissions les présenter une par une, en nous attardant sur leurs spécificités. Pour chaque proposition, nous décrivons le modèle de données et le(s) langage(s) de manipulation de données en nous concentrant sur la structure, la granularité et l'adaptation des concepts issus du modèle relationnel. Nous montrons notamment comment sont modélisés dimensions, membres, mesures, comment sont prises en compte les hiérarchies dans le modèle (si elles le sont), et comment les opérations élémentaires peuvent être *directement* prises en compte par chacun des langages. A titre d'exemple, nous décrivons dans chaque modèle le cube *ventes* introduit dans la figure II.5 de la section 2, page 21. Quand cela est possible, nous incluons l'aspect granulaire lié au cube *ventes* (figure II.12, page 27). Enfin, nous donnons une idée de l'expression des requêtes de la section 2 dans chacun des langages.

Nous choisissons de présenter les langages étudiés dans l'ordre chronologique de leur apparition, depuis les premiers travaux jusqu'aux plus récents. La terminologie employée par les auteurs des différents travaux présentés n'étant pas toujours homogène, nous utilisons dans cette section les termes et les opérations élémentaires introduits dans la section 2 (e.g., membres, mesures, "roll-up"). Nous souhaitons attirer l'attention du lecteur sur les points suivants :

- cette section contient des détails techniques liés à la conception de langages, mais sa lecture n'est pas nécessaire à la compréhension des chapitres III et IV, et un passage direct à la conclusion de ce chapitre est possible ;

- certains aspects des travaux étudiés ont été simplifiés pour permettre une présentation homogène et concise. Cette présentation constitue un point d'entrée sur le domaine, que le lecteur pourra approfondir en se référant aux articles originaux ;

- un rappel des notions de langage de manipulation de données pour bases de données relationnelles comprenant un lexique des termes utilisés dans cette section est donné en annexe.

3.2 Les propositions

3.2.1 Une algèbre pour matrices bidimensionnelles

Nous décrivons le modèle de base de données tabulaire proposé par [24]. Dans ce modèle, les données sont rangées dans des matrices bidimensionnelles. Ces matrices sont manipulées grâce à une algèbre comportant 12 opérateurs plus une instruction d'itération.

Structure multidimensionnelle Soit S un ensemble de symboles contenant une constante particulière $\perp$ pour décrire les valeurs nulles (inapplicables ou inconnues).

Dans ce modèle, une référence de cellule est notée $F_c(i,j)$ où $c \in S$ est le nom du cube, F_c est un nom de fonction et $i,j \in \mathbb{N}$. Le contenu v d'une cellule est un symbole de S, et donc une cellule est notée $F_c(i,j) = v$.

Soient $A = \{0,\ldots,k\}$ et $B = \{0,\ldots,l\}$ deux segments initiaux de $\mathbb{N}$. Dans ce modèle, un cube c est une fonction F_c ayant pour domaine $A \times B$ et pour codomaine S. k et l représentent respectivement le nombre de lignes et le nombre de colonnes de c, et $F_c(i,j), i \in [0,\ldots,k], j \in [0,\ldots,l]$ désigne l'information contenue à l'intersection de la ligne i et de la colonne j.

Pour un cube c de k lignes et l colonnes, chaque $F_c(0,j), 1 < j \leq l$ contient un nom de colonne, et chaque $F_c(i,0), 1 < i \leq k$ contient un nom de ligne. $F_c(0,0)$ contient c. Pour ce modèle, nous considérerons que les membres sont les noms de lignes et les noms de colonnes d'un cube.

Exemple 3.1 Nous décrivons le cube *ventes* de la figure II.5 de la page 21 dans ce modèle. Ce cube peut être décrit par la matrice F_{ventes} de la figure II.26[7]. F_{ventes} est définie comme une fonction de $\{0,\ldots,10\} \times \{0,\ldots,6\}$ dans S, avec par exemple $F_{ventes}(0,0)$ = ventes, $F_{ventes}(0,1)$ = pièces, $F_{ventes}(2,0) = \perp$ et $F_{ventes}(3,4) = 10$. □

Opérations de restructuration Pour animer le modèle de données, l'algèbre comprend cinq opérateurs dits de restructuration, un opérateur de transposition (exprimable par combinaison d'autres opérateurs), et un opérateur dit de suppression de redondance.

Deux opérateurs de restructuration sont utilisés pour passer d'une représentation sous une forme proche de la "relation" (i.e., un ensemble de tuples sans nom de ligne) à une représentation sous forme de tableau croisé (noms de lignes et de colonnes constitués à partir des valeurs de certains champs des tuples). Ces deux opérateurs permettent de coder une forme de "nest"

7. Le codage choisi est arbitraire. Nous aurions pu, comme le suggèrent les auteurs, coder un cube tridimensionnel par un ensemble de matrices, mais ce codage n'est pas satisfaisant car la troisième dimension ne serait alors pas traitée de manière symétrique aux deux autres.

ventes	pièces	années	quantité	quantité	quantité	quantité
régions	$\perp$	$\perp$	est	ouest	sud	nord
$\perp$	écrous	1996	50	60	40	$\perp$
$\perp$	écrous	1995	70	10	20	$\perp$
$\perp$	écrous	1994	100	30	$\perp$	10
$\perp$	clous	1996	70	$\perp$	$\perp$	40
$\perp$	clous	1995	50	10	10	20
$\perp$	clous	1994	40	40	$\perp$	$\perp$
$\perp$	vis	1996	$\perp$	50	50	60
$\perp$	vis	1995	10	50	60	30
$\perp$	vis	1994	10	50	60	20

FIG. II.26 - *Le cube* ventes

(e.g., on peut considérer que les pièces et les années sont imbriquées dans le cube de la figure II.26).

Deux autres opérateurs de restructuration permettent d'exprimer directement l'opération "split" et son opération réciproque.

Le dernier opérateur de restructuration permet d'échanger deux lignes d'un cube (une des lignes est échangée avec la première ligne), et correspond de manière restreinte à l'opération "switch". Comme les membres sont contenus dans une ligne ou une colonne de matrice au même titre que les mesures, les opérations "push" et "pull" sont prises en compte par cet opérateur, mais de manière restreinte, puisque le modèle de données ne permet pas de représenter plusieurs mesures par cellule.

Un opérateur de transposition de matrice classique peut être simulé en combinant d'autres opérateurs du langage. Grâce à cet opérateur, il est possible d'exprimer pour toute opération concernant les lignes (resp., colonnes) une opération similaire sur les colonnes (resp., lignes).

Un opérateur de suppression de redondance à pour rôle de condenser la représentation et offrir ainsi une mise en forme plus conviviale des données (e.g., en supprimant les combinaisons contenant le symbole $\perp$). Il permet par exemple de passer de la représentation de gauche de la figure II.27 à celle de droite.

t	1	2	3
a	10	$\perp$	$\perp$
a	$\perp$	20	$\perp$
a	$\perp$	$\perp$	30

t	1	2	3
a	10	20	30

FIG. II.27 - *Suppression de redondance*

Granularité La notion de granularité n'est prise en compte ni dans le modèle de données ni dans le langage. Ainsi, modéliser les niveaux autres que ceux décrits dans le cube *ventes* reviendrait à décrire d'autres cubes, sans possibilité d'établir de liens entre les différentes cubes représentant les différents niveaux.

Méta-données et données L'ensemble S des symboles utilisés est en fait l'union de deux ensembles disjoints N de noms et V de valeurs (plus la constante $\perp$). Intuitivement, les noms de N correspondent aux méta-données relationnelles et les valeurs de V correspondent aux données relationnelles. Pour l'exemple 3.1, les symboles *ventes, régions, pièces, années et quantité* sont choisis dans N, et les autres symboles (*est*, *clous*, 50, ...) sont choisis dans V.

Opérations classiques 6 opérateurs de l'algèbre relationnel classique (union, différence, produit cartésien, renommage de noms de colonnes, sélection et projection) sont incorporés à l'algèbre de [24].

A l'exception de l'union, ces opérateurs fonctionnent soit exclusivement en ligne, soit exclusivement en colonne, ce qui simplifie leur définition. Par exemple, la sélection permet de sélectionner les lignes d'un cube dont les données satisfont des critères, et la projection permet de sélectionner un ensemble de colonnes en fonction de leur nom. En combinant ces opérateurs avec l'opérateur de transposition, il est possible d'exprimer les opérations de sélection et de projection sur les lignes comme sur les colonnes d'un cube.

Le produit cartésien, l'union, la différence et l'intersection sont étendus de façon triviale à la prise en compte de structures bidimensionnelles.

Deux opérateurs supplémentaires permettent de créer de nouvelles valeurs, par ajout d'une nouvelle ligne et/ou colonne à un cube, contenant des valeurs aléatoires deux à deux distinctes. Les auteurs précisent que ces opérateurs ont été rajoutés dans le but d'obtenir un langage capable d'exprimer toutes les transformations sur les cubes[8].

Expression de requêtes Un programme algébrique est constitué d'instructions d'itération et d'instructions d'affectation de la forme :

$$c \leftarrow \langle opérateur \rangle \ \langle arguments \rangle$$

où c est le nom du cube résultat, $\langle opérateur \rangle$ est un nom d'opérateur algébrique, et $\langle arguments \rangle$ sont les arguments nécessaires à l'opérateur. Le langage ainsi défini est clos et complet pour le modèle de données : il permet de spécifier toutes les transformations d'une base de données tabulaire en une autre base de données tabulaire.

8. L'ajout de ce type d'opérateurs est une manière classique d'accroître l'expressivité d'un langage de requête [4, 5].

Exemple 3.2 Nous traduisons dans l'algèbre de [24] la requête 2.4 de la section 2 (les autres requêtes faisant intervenir des fonctions d'agrégat et la granularité, elles ne peuvent être exprimées directement). Nous partons du cube de la figure II.26, pour spécifier la construction d'un cube qui contient les quantités vendues de clous et de vis inférieures à 30, sous forme de tableau croisé, les années en ligne, les pièces imbriquées avec les régions en colonnes. On utilise l'opérateur de transposition de manière transparente pour simplifier la lecture. Ainsi, suivant les conventions de [24], si c est un cube, O est un opérateur et p une liste de paramètres pour O, alors $O_p(c)$ désigne l'opération O appliquée sur les lignes de c, et $O^p(c)$ désigne l'opération O appliquée sur les colonnes de c.
Détaillons la construction de la requête. En premier lieu, on sélectionne (opérateur SEL) les pièces *vis* et *clous*, avec la requête :

$$m1 \leftarrow SEL_{pièces=clous \vee pièces=vis}(ventes)$$

Le cube obtenu est représenté dans la figure II.28.

m1	pièces	années	quantité	quantité	quantité	quantité
régions	⊥	⊥	est	ouest	sud	nord
⊥	clous	1996	70	⊥	⊥	40
⊥	clous	1995	50	10	10	20
⊥	clous	1994	40	40	⊥	⊥
⊥	vis	1996	⊥	50	50	60
⊥	vis	1995	10	50	60	30
⊥	vis	1994	10	50	60	20

FIG. II.28 - *Le cube* m1

La prochaine étape consiste à sélectionner les quantités inférieures à 30. L'opérateur de sélection fonctionnant en ligne, il nous faut d'abord restructurer le cube $m1$ pour n'avoir qu'une quantité par ligne (sinon une ligne de $m1$ sera sélectionnée si l'une des quantités est inférieure à 30). Pour ce faire, nous employons l'opérateur $FOLD$ pour passer à une représentation où les régions constituent une colonne. Ensuite, il est possible de sélectionner les quantités désirées (figure II.29) :

$$m2 \leftarrow FOLD_{by\ régions\ on\ quantité}(m1)$$
$$m3 \leftarrow SEL_{quantité \leq 30}(m2)$$

Le cube $m3$ de la figure II.29 peut alors être condensé grâce à l'opération $PURGE$, pour n'obtenir qu'une seule colonne *quantité* :

$$m4 \leftarrow PURGE(m3)$$

Ensuite, on restructure ($UNFOLD$) le cube pour faire passer les noms de pièces comme noms de colonnes, puis les noms de régions comme noms de colonnes (figure II.30) :

$$m5 \leftarrow UNFOLD_{by\ pièces\ on\ quantité}(m4)$$
$$m6 \leftarrow UNFOLD_{by\ région\ on\ quantité}(m5)$$

m3	pièces	années	régions	quantité	quantité	quantité	quantité
$\perp$	clous	1995	ouest	$\perp$	10	$\perp$	$\perp$
$\perp$	clous	1995	sud	$\perp$	$\perp$	10	$\perp$
$\perp$	clous	1995	nord	$\perp$	$\perp$	$\perp$	20
$\perp$	vis	1995	est	10	$\perp$	$\perp$	$\perp$
$\perp$	vis	1995	nord	$\perp$	$\perp$	$\perp$	30
$\perp$	vis	1994	est	10	$\perp$	$\perp$	$\perp$
$\perp$	vis	1994	nord	$\perp$	$\perp$	$\perp$	20

FIG. II.29 - *Le cube* m3

m6	années	quantité	quantité	quantité	quantité	...
régions	$\perp$	est	est	ouest	ouest	...
pièces	$\perp$	vis	clous	vis	clous	...
$\perp$	1995	$\perp$	$\perp$	$\perp$	10	...
$\perp$	1995	$\perp$	$\perp$	$\perp$	$\perp$	...
$\perp$	1995	$\perp$	$\perp$	$\perp$	$\perp$	...
$\perp$	1995	10	$\perp$	$\perp$	$\perp$	...
$\perp$	1995	$\perp$	$\perp$	$\perp$	$\perp$	...
$\perp$	1994	10	$\perp$	$\perp$	$\perp$	...
$\perp$	1994	$\perp$	$\perp$	$\perp$	$\perp$	...

FIG. II.30 - *Le cube* m6

L'opérateur $PURGE$ peut encore être utilisé à ce stade pour condenser la représentation (figure II.31).

$$m7 \leftarrow PURGE_{années}(m6)$$
$$m8 \leftarrow PURGE^{pièces}(m7)$$

m8	années	quantité	quantité	quantité	quantité	quantité
régions	$\perp$	est	ouest	sud	nord	nord
pièces	$\perp$	vis	clous	clous	vis	clous
$\perp$	1995	10	10	10	30	20
$\perp$	1994	10	$\perp$	$\perp$	20	$\perp$

FIG. II.31 - *Le cube* m8

La mise en forme finale consiste tout d'abord à interchanger ($SWITCH$) la ligne 0 (contenant les noms de colonne) avec la ligne 1 (celle des régions) pour nommer les colonnes par les régions. L'ex ligne 0 peut alors être supprimée en projetant ($PROJ$) sur *pièces* et $\perp$. (figure II.32). Les requêtes employées sont :

$$m9 \leftarrow SWITCH_{régions}(m8)$$

$m10 \leftarrow PROJ_{pièces,\perp}(m9)$

m10	$\perp$	est	ouest	sud	nord	nord
pièces	$\perp$	vis	clous	clous	vis	clous
$\perp$	1995	10	10	10	30	20
$\perp$	1994	10	$\perp$	$\perp$	20	$\perp$

FIG. II.32 - *Le cube* m10

On fait de même avec les colonnes pour nommer les lignes par les années et obtenir le cube de la figure II.33, en employant les requêtes:

$m11 \leftarrow SWITCH^{\perp}(m10)$

$ventes\ inf30 \leftarrow SEL^{\perp \neq pièces}(m11)$

ventes inf30	est	ouest	sud	nord	nord
$\perp$	vis	clous	clous	vis	clous
1995	10	10	10	30	20
1994	10	$\perp$	$\perp$	20	$\perp$

FIG. II.33 - *Le cube* ventes inf30

On notera l'emploi de la sélection sur les colonnes qui permet de supprimer la colonne contenant le symbole *pièces*. □

L'algèbre de [24] aide à mieux saisir le concept de restructuration de structures bidimensionnelles et s'avère une base prometteuse pour étudier les possibilités d'optimisation liées à ce concept. Cependant, certaines opérations comme "nest" ou "push" n'ont pas de réelle correspondance directe et doivent faire l'objet d'un codage. L'exemple de requête donné illustre le caractère constructif de la spécification de requêtes, qui oblige à suivre attentivement l'évolution de la structure du cube afin de pouvoir composer correctement les opérateurs.

Une particularité de ce modèle est la possibilité de représenter et manipuler les cellules contenant les membres comme les cellules contenant les mesures. Ceci permet de manipuler dans les cubes à la fois les données et les méta-données, même si ces constantes ne sont pas traitées de manière uniforme. En effet, les méta-données sont les constantes de N, qui sont les seules constantes utilisables pour paramétrer les opérateurs de l'algèbre, et les données sont les constantes de V.

Les principales limitations de ce travail dans un contexte OLAP sont la restriction à un modèle bidimensionnel et l'impossibilité de décrire des groupements ou d'opérer des agrégations.

3.2.2 Une algèbre pour hypercubes

Nous décrivons maintenant le modèle logique d'hypercube proposé par [6, 7]. Dans ce modèle, l'information est structurées en hypercubes pouvant contenir des tuples. Ces hypercubes sont manipulés à l'aide d'une algèbre comprenant 6 opérateurs, dont certains sont paramétrés par des fonctions externes au langage.

Structure multidimensionnelle Soit un ensemble de noms de dimension D. A chaque dimension $d \in D$ est associé un domaine de valeurs dom_d. Dans ce modèle, les membres et les mesures sont pris parmi les domaines de chaque dimension. Soit deux constantes particulières notées 0 et 1.

Soit $d_1, \ldots, d_n, d'_1, \ldots d'_k \in D$. Une référence de cellule est un tuple $\langle v_1, \ldots, v_n \rangle$ avec $v_i \in dom_{d_i}, i \in [1, \ldots, n]$. Le contenu d'une cellule est soit la constante 0, soit la constante 1, soit un tuple $\langle v'_1, \ldots, v'_k \rangle$, où chaque $v'_i \in dom_{d'_i}, i \in [1, \ldots, k]$.

Un cube c à n dimensions est une fonction F_c associant à chaque tuple $\langle v_1, \ldots, v_n \rangle$ de membres (appartenant à $dom_{d_1} \times \ldots \times dom_{d_n}$) l'un des éléments suivants :

- un tuple $\langle v'_1, \ldots, v'_k \rangle$ de mesures appartenant à $dom_{d'_1} \times \ldots \times dom_{d'_k}$ si la cellule de coordonnées $\langle v_1, \ldots, v_n \rangle$ contient $\langle v'_1, \ldots, v'_k \rangle$;

- la constante 0 si la cellule de référence $\langle v_1, \ldots, v_n \rangle$ n'existe pas pour c ;

- la constante 1 si la cellule de référence $\langle v_1, \ldots, v_n \rangle$ existe mais ne contient pas de mesure.

Exemple 3.3 Nous décrivons le cube *ventes* de la figure II.5 de la page 21 dans ce modèle. Soit les dimensions *années*, *pièces*, *régions* et *quantité* dont les domaines sont : $dom_{années} = \{1994, 1995, 1996\}$, $dom_{pièces} = \{écrous, clous, vis\}$, $dom_{régions} = \{est, ouest, nord, sud\}$, et $dom_{quantités} \subseteq \mathbb{N}$.
Le cube *ventes* est défini par trois dimensions pour les membres (*années, pièces et régions*), une dimension pour les mesures (le tuple $\langle quantité \rangle$) et la fonction F_{ventes} de $dom_{années} \times dom_{pièces} \times dom_{régions}$ vers $dom_{quantité} \cup \{0, 1\}$. Une référence du cube *ventes* est $(1995, écrous, ouest)$. La fonction F_{ventes} associe références et contenus. Par exemple, $F_{ventes}(1995, écrous, ouest) = \langle 10 \rangle$, et $F_{ventes}(1996, clous, sud) = 0$[9]. □

9. que l'on ne confondra pas avec $F_{ventes}(1996, clous, sud) = \langle 0 \rangle$!

Opérations de restructuration Le modèle de données employé (notamment la possibilité de constituer des tuples de mesures, ou la possibilité de représenter la non validité de combinaison de membres) autorise la prise en compte directe des opérations "push" et "pull" par deux opérateurs. On notera cependant que le cube résultant de l'application de l'opérateur "push" conserve la dimension qui a fait l'objet du "push".

La combinaison de ces deux opérateurs avec les opérateurs classiques permet d'envisager l'expression d'autres opérations de restructuration élémentaires telles que "split" ou "rotate".

Granularité La granularité n'est pas prise en compte dans le modèle de données. L'aspect granulaire est par contre pris en compte par l'utilisation de fonctions externes à l'algèbre, qui permettent de spécifier la construction des groupements et les opérations d'agrégation sur ces groupements.

Ainsi, un opérateur permet d'exprimer les opérations de "roll-up" et de "drill-down" via des fonctions externes.

Schéma et méta-données Les méta-données d'un cube sont les noms de dimensions qui le composent : le schéma d'un cube est constitué des n noms de dimensions des membres $d_1, \ldots, d_n$, et un tuple des k noms de dimensions des mesures.

Exemple 3.4 Le schéma du cube *ventes* décrit dans l'exemple 3.3 est : *années*, *pièces*, *régions* pour les membres, et le tuple $\langle$*quantités*$\rangle$ pour les mesures. □

Opérations classiques Un opérateur correspond directement au premier cas de l'opération de projection décrite dans la section 2, et permet de décrémenter le nombre de dimensions d'un cube. On notera que la dimension enlevée doit avoir un domaine constitué d'un seul élément, pour que le passage d'un cube de dimension n à un cube de dimension $n-1$ préserve la dépendance fonctionnelle entre les membres et les mesures.

Un opérateur permet de faire de la sélection sur les membres ; combiné avec l'opérateur "pull", la sélection sur les mesures est aussi possible.

Un opérateur de jointure permet de combiner deux cubes quelconques entre eux, selon une ou plusieurs dimensions. Cet opérateur est paramétré par des fonctions externes au langage qui permettent d'effectuer des calculs sur les membres des dimensions jointes, et sur les mesures des cubes joints[10].

Les auteurs précisent que les autres opérateurs classiques (e.g., union, différence) peuvent être exprimés en combinant des opérateurs de leur langage.

10. L'opérateur permettant d'exprimer les opérations de "roll-up" et "drill-down" dans cet algèbre est en fait un cas particulier de l'opérateur de jointure.

Expression de requêtes L'algèbre proposée par [6, 7] a délibérément été conçue comme une collection d'opérateurs pouvant être traduits en SQL. L'algèbre est close, et bien que sa puissance d'expression n'a pas été caractérisée précisément, les auteurs observent qu'elle est au moins aussi puissante que l'algèbre relationnelle.

Exemple 3.5 Nous traduisons dans l'algèbre de [6, 7] les requêtes 2.1, 2.3 et 2.5 de la section 2. Nous choisissons de traduire ces requêtes car la traduction des autres requêtes dérive facilement de celles-ci, sauf pour la requête 2.4 qui n'admet pas de correspondance, car l'imbrication des dimensions n'est pas possible dans ce modèle. On notera que les requêtes basées sur des calculs de proche en proche, comme la requête 2.7 qui vise à faire des estimations de ventes d'une année sur l'autre, ne peuvent être prises en compte que par la définition de plusieurs requêtes et leur composition (e.g., une requête par année dans le cas de la requête 2.7).
Pour traduire ces requêtes, nous supposons l'existence du cube *ventes villes* de la figure II.16 de la page 29, qui détaille les quantités de pièces vendues par ville et par année. Ce cube comprend les dimensions *années, quantités* et *pièces* décrites dans l'exemple 3.3, ainsi qu'une dimension *villes* dont le domaine est constitué des noms de villes.
Nous traduisons d'abord la requête 2.5, qui vise à calculer le nombre total de pièces vendues dans chaque ville pour la période 94-96, pour illustrer l'emploi de l'opérateur de "roll-up" *merge*. Cet opérateur permet ici d'opérer deux "roll-ups" sur le cube *ventes villes* en employant la syntaxe suivante :

$$merge(ventes\ villes, \{[années, f_{merge_a}], [pièces, f_{merge_p}]\}, sum)$$

où *sum* est une fonction externe sommant les mesures, et où les fonctions f_{merge_a} et f_{merge_p} décrivent les hiérarchies des dimensions *années* et *pièces* représentées dans la figure II.12 (par exemple, $f_{merge_a}(1996) = temps$, $f_{merge_p}(clous) = produit$).
Nous traduisons maintenant la requête 2.1, pour trouver le nombre de villes où il a été vendu moins de 20 unités de l'une des trois pièces en 1996. Nous détaillons son écriture (pour en faciliter la lecture, nous nommerons les cubes intermédiaires). En premier lieu, il faut faire passer les mesures de quantité en membres pour pouvoir opérer la sélection, en utilisant l'opérateur *pull*.

$$pull(ventes\ villes, quantité)$$

Appelons $c1$ le cube intermédiaire obtenu. On sélectionne (*restrict*) les quantités de $c1$ inférieures à 20, puis l'année 1996.

$$\begin{aligned}&restrict(\\&\quad restrict(c1, quantité, < 20),\\&\qquad années, = 1996),\end{aligned}$$

Appelons $c2$ le cube obtenu. Sur ce cube, deux agrégations (*merge*) successives sont nécessaires. En effet, la dimension des pièces peut contenir plus d'un membre ce qui interdit de supprimer cette dimension en projetant sur la dimension des villes. En agrégeant une première fois toutes les pièces, on obtient un cube détaillant seulement la dimension des villes, sur lequel une opération de comptage permet donc d'obtenir le résultat souhaité.

$$
\begin{array}{l}
merge(\\
\quad merge(c2, \{[pièces, f_{merge_p}]\}, count), \\
\qquad \{[villes, f_{merge_v}]\}, count) \qquad \text{(deuxième merge)}
\end{array}
$$

Dans cette dernière partie, *count* est une fonction externe décomptant les mesures, f_{merge_p} et f_{merge_v} sont des fonctions externes qui définissent la manière de grouper respectivement les pièces et les villes.
Supposons maintenant qu'il existe un cube appelé *prix*, qui décrit le prix de chaque pièce (une représentation d'un tel cube restreint à l'année 1996 est donnée dans la figure II.20 de la page 32). Nous exprimons la requête 2.3 qui vise à calculer le chiffre d'affaires réalisé par pièces, régions et années, en multipliant le prix de ventes par le nombre d'articles vendus (le chiffre d'affaire devant apparaître comme mesure dans le cube *ventes* à côté de chaque quantité vendue). Cette requête nécessite la jointure du cube *ventes* avec le cube *prix*. Lors de cette jointure (*join*), les contenus des cellules sont multipliés. Ensuite on opère une jointure de ce cube avec le cube *ventes*, en employant une fonction externe qui crée un tuple dans chaque cellule contenant à la fois la quantité vendue et le chiffre d'affaire. La requête se traduit par :

$$
\begin{array}{l}
join(\\
\quad join(ventes, prix, [id], mult), \\
\qquad ventes, [id, id, id], f_{tuple})
\end{array}
$$

Ici, *id* désigne la fonction identité, c'est à dire que pour les dimensions jointes, les membres du cube résultat sont identiques aux membres des cubes initiaux, *mult* est la fonction externe qui multiplie les contenus des cellules, et f_{tuple} désigne une fonction externe qui à partir des mesures quantité et chiffre d'affaire des cubes initiaux renvoie un tuple constitué de ces deux mesures. □

L'algèbre de [6, 7] comporte un nombre restreint d'opérateurs, dont certains (e.g., *join*) sont polyvalents. Grâce à la traduction de ses opérateurs en SQL, elle permet d'envisager des possibilités OLAP directement sur des bases de données relationnelles (même si d'après les auteurs, une traduction directe ne semble pas satisfaisante du point de vue efficacité).

Les auteurs motivent le choix du paramétrage d'opérateurs par des fonctions externes au langage par le souci de conserver un nombre restreint d'opérateurs. Ce choix implique que les informations décrites par des fonctions externes (e.g., les groupements) pourraient ne pas entrer en considération lors de l'optimisation des requêtes[11], et suppose la connaissance d'un langage externe pour spécifier les groupements. Néanmoins, on notera que l'absence de restriction sur l'emploi des fonctions permet de modéliser n'im-

11. Toutefois, des recherches à ce sujet commencent à aboutir [15].

porte quel type de hiérarchies (e.g., multiples, en diamant), et n'importe quel type d'agrégats.

Le modèle de [7] dissocie clairement l'information de sa représentation, en considérant l'animation comme étant plutôt du ressort de l'interface graphique. Ainsi, les opérations de restructuration basées sur les positions relatives des dimensions comme "rotate" sont difficiles à spécifier. La position relative des membres n'est pas prise en compte, et l'imbrication n'est pas permise, les opérations "switch" et "nest" n'ont donc pas de sens pour ce modèle.

3.2.3 Deux algèbres pour cubes et relations

Le modèle que nous étudions maintenant, proposé par [35], base un cube sur un ensemble de relations (au sens du modèle relationnel). Dans ce modèle certaines relations, appelées relations de groupement, sont utilisées pour représenter la décomposition de niveaux de granularité en sous-niveaux. Deux algèbres sont proposées pour manipuler les relations et les cubes de ce modèle. La première algèbre (appelée algèbre de groupement) est une extension de l'algèbre relationnelle pour manipuler des relations et des relations de groupement. La deuxième algèbre (appelée algèbre multidimensionnelle) s'adresse plus particulièrement aux cubes, et comprend 6 opérateurs qui sont définis comme des fonctions de cubes vers cubes, ou de relations vers cubes.

Structure multidimensionnelle Soient un ensemble A de noms d'attribut, et un ensemble D de noms de dimension. A chaque attribut $a \in A$ est associé un domaine dom_a. Soit un autre ensemble V, contenant des scalaires (entiers, réels, etc) ainsi qu'une valeur particulière (servant de valeur nulle). Dans ce modèle, les membres sont pris parmi les domaines associés aux attributs, et les mesures parmi V.

Une dimension est définie par un nom de dimension $d \in D$ et une relation (au sens du modèle relationnel) r_d. Soit $d_1, \ldots, d_n \in D$. Une référence de cellule est une mise en relation d'un tuple pris dans chaque relation $r_{d_1}, \ldots, r_{d_n}$, et est notée $(t_1, \ldots, t_n)$ avec $t_1 \in r_{d_1}, \ldots, t_n \in r_{d_n}$. Le contenu d'une cellule est une constante de V.

Un cube c à n dimensions est un ensemble de n relations $r_{d_1}, \ldots, r_{d_n}$, et une fonction F_c ayant pour domaine l'ensemble de références formées à partir des relations $r_{d_1}, \ldots, r_{d_n}$, et pour codomaine V.

Exemple 3.6 Nous décrivons le cube *ventes* de la figure II.5 de la page 21 dans ce modèle. Soit trois dimensions nommées *temps*, *produit* et *lieu*, ainsi que trois attributs, *années, pièces et régions*, avec $dom_{années} = \{1994, 1995, 1996\}$, $dom_{pièces} = \{écrous, clous, vis\}$, et $dom_{régions} = \{est, ouest, sud, nord\}$. Les relations correspondantes sont décrites dans la figure II.34.

Une référence du cube *ventes* est $(1994, \acute{e}crous, est)$. La fonction F_{ventes} fait correspondre à cette référence une constante de l'ensemble V des valeurs scalaires : $F_{ventes}(1995, \acute{e}crous, ouest) = 10$. □

r_{temps}

années
1994
1995
1996

$r_{produit}$

pièces
écrous
clous
vis

r_{lieu}

régions
est
ouest
sud
nord

FIG. II.34 - *Les différentes dimensions du cube* ventes

Opérations de restructuration Trois opérateurs de l'algèbre multidimensionnelle permettent de (re)structurer des cubes.

Deux opérateurs sont dédiés à la construction de cubes à partir de relations. Le premier opérateur permet de générer un cube à partir d'une relation, en spécifiant quel attribut de la relation fait office de mesure. Le deuxième opérateur permet d'ajouter une dimension à un cube.

Un opérateur permet de réarranger les dimensions d'un cube, c'est à dire de transférer un attribut d'une dimension à une autre dimension. Il permet d'exprimer l'opération "nest".

Granularité Le découpage de niveaux en sous-niveaux est décrit par une relation de groupement. Une telle relation comprend des attributs préfixés par un nom de dimension.

Exemple 3.7 La granularité du cube *ventes* est prise en compte par des relations de groupement. Par exemple, la relation *gr* de la figure II.35 représente le regroupement des villes en régions. Cette relation peut être notamment utilisée pour calculer le cube *ventes* à partir du cube *ventes villes* qui détaille les quantités vendues au niveau des villes. □

gr

lieu.région	lieu.villes
est	lyon
est	dijon
⋮	⋮
nord	paris
⋮	⋮

FIG. II.35 - *La relation* gr

Opérations associées à la granularité La granularité est prise en compte soit au niveau des relations de groupement, avec trois opérateurs de l'algèbre de groupement, soit directement au niveau des cubes, avec un opérateur de l'algèbre multidimensionnelle.

Les trois opérateurs de l'algèbre de groupement sont les suivants. Un premier opérateur permet d'ordonner les tuples d'une relation en ordre croissant ou décroissant, avec un principe similaire à la clause "order-by" de SQL. Un second opérateur permet de constituer des intervalles réguliers de tuples sur une relation, qui peuvent se recouvrir. Le résultat de cette opération est une nouvelle relation, dans laquelle les tuples constituant chaque intervalle sont listés (certains tuples sont donc répétés au besoin). Cette relation comporte de plus deux attributs supplémentaires dont les valeurs jouent le rôle d'identifiants de chaque intervalle. Le troisième opérateur de l'algèbre de groupement permet d'appliquer une fonction d'agrégat à un cube, via les groupements définis par une relation de groupement. Cet opérateur permet d'exprimer l'opération "roll-up" à partir de n'importe quel groupement (calculé par requête ou non).

Un opérateur de l'algèbre multidimensionnelle permet d'exprimer plusieurs "roll-ups" sur un cube, sans passer par des relations de groupement. Les groupes sont alors constitués de tuples ayant une valeur commune pour certains attributs $a_1, \ldots, a_n$ dans les dimensions $d_1, \ldots d_n$. Le résultat est un cube possédant comme dimensions $d_1, \ldots d_n$, comme attributs $a_1, \ldots, a_n$, et comme mesures les valeurs de l'agrégat calculé pour chaque groupe.

Schéma et méta-données Les méta-données d'un cube sont les symboles de A qui décrivent les attributs (au sens du modèle relationnel) et les symboles de D qui décrivent les noms des relations sous-jacentes à un cube, i.e., les noms des dimensions.

Le schéma d'un cube c de dimension n est noté $\{(d_1, R_1), \ldots, (d_n, R_n)\}$ où chaque $d_i, i \in [1, \ldots, n]$ est un nom de dimension, et chaque $R_i, i \in [1, \ldots, n]$ est l'ensemble des attributs de la relation r_{d_i} associée à d_i.

Exemple 3.8 Le schéma du cube *ventes* décrit dans l'exemple 3.6 est $\{(temps, \{années\}), (produit, \{pièces\}), (lieu, \{régions\})\}$. □

Opérations classiques Les opérateurs de l'algèbre relationnelle permettent de définir des nouvelles relations de groupement, et sont inclus dans l'algèbre de groupement.

Les opérations classiques sont aussi adaptées aux cubes, via deux opérateurs de l'algèbre multidimensionnelle qui permettent d'effectuer la jointure entre une relation et un cube, et l'union de deux cubes.

Expression de requêtes L'algèbre de groupement proposée par [35] est une extension de l'algèbre relationnelle en vue d'ordonner les tuples d'une relation et d'appliquer des fonctions d'agrégats. Le principe développé par [35] est de constituer à l'aide de cette algèbre des relations de groupements, et d'utiliser ces relations pour agréger et classer les données. Un des rôles de l'algèbre multidimensionnelle est de construire des cubes à partir de relations classiques. La puissance d'expression de ces langages n'a pas été étudiée, mais l'algèbre de groupement est au moins aussi expressive que l'algèbre relationnelle, puisqu'elle en est une extension.

Exemple 3.9 Nous donnons la traduction des requêtes de la section 2 dans le cadre de [35]. Le modèle ne permet pas d'associer plusieurs mesures à une référence, et le langage ne permet pas de distinguer les dimensions d'un cube. En conséquence, les requêtes 2.3 et 2.4 n'admettent pas de traduction. De plus, il n'existe aucun opérateur permettant de faire directement ou indirectement une sélection sur les mesures d'un cube, ce qui rend la requête 2.1 inexprimable.
Soit le cube *ventes villes* (figure II.16 de la page 29), dont le schéma est le même que celui du cube *ventes*, excepté la dimension *lieu* qui comporte un attribut *villes* au lieu d'un attribut *région*. La requête 2.5 vise à calculer le total des ventes de pièces dans chaque ville pour la période 1994-1996. Elle s'exprime simplement en utilisant l'opérateur de l'algèbre multidimensionnelle de "roll-up" sur un cube. Cet opérateur est noté f_D^R, où f est une fonction d'agrégat, R et D décrivent respectivement les attributs et dimensions concernés pour constituer les groupes. Pour constituer un groupe par ville et agréger chaque groupe, la requête s'écrit donc :

$$sum_{lieu}^{villes}(ventes\ villes)$$

La requête 2.6 spécifie le calcul de la moyenne glissante des ventes par région et par pièces, pour une fenêtre de deux années consécutives. Elle est traduite grâce aux opérateurs de l'algèbre de groupement. Le principe est le suivant. Tout d'abord, on constitue une relation de groupement à partir des relations *temps*, *lieu* et *produit* ($\times$ dénote le produit cartésien).

$$temps \times produit \times lieu$$

Ensuite, cette relation est divisée en deux intervalles, un pour les années 1994-1995 et un autre pour les années 1995-1996, à l'aide de l'opérateur ρ permettant de partitionner une relation.

$$\rho_{années,asc}^{1,1,2}(temps \times produit \times lieu)$$

où la constante *asc* désigne le classement des tuples de la relation dans l'ordre croissant (des années), et où la liste de paramètres $1, 1, 2$ désigne la constitution d'intervalles de longueur 2, commençant une position après le début de l'intervalle précédent (premier 1), depuis le premier tuple de la relation (second 1).
Cette relation de groupement est utilisée enfin par l'opérateur d'agrégation [12]

12. de l'algèbre de groupement, différent de celui de l'algèbre multidimensionnelle utilisé

sur le cube *ventes* pour calculer la moyenne de chaque groupe. La requête entière s'écrit donc :

$$avg_{ventes,moyenne}^{fin-années,produit.pièces,lieu.régions}($$
$$\rho_{années,asc}^{1,1,2}($$
$$temps \times produit \times lieu))$$

où l'attribut *fin-années* est produit par l'opérateur ρ et contient un identifiant pour chaque intervalle, *produit.pièces* et *lieu.région* sont les des attributs de la relation de groupement, et *moyenne* est un nouvel attribut pour les valeurs calculées. Ces attributs constituent les attributs du résultat de la requête.
Pour prendre en compte la requête 2.2 qui demande la possibilité de calculer le total des ventes par mois comme par semaine, il suffit de deux relations de groupement, qui détaillent d'une part les groupements des jours en mois, d'autre part les groupements des jours en semaines. Exprimer la requêtes qui calcul le total des ventes par semaine ou par mois dérive facilement de la traduction de la requête 2.6. □

Les deux algèbres proposées par [35] pour manipuler des relations et des cubes mettent l'accent sur l'aspect granularité (définition d'une large gamme de groupements) d'une part, et sur la construction de cubes à partir de relations d'autre part.

Le fait que le modèle se base sur des relations suggère une implantation naturelle dans une base de données relationnelle. Néanmoins, dans ce modèle, les membres et les mesures ne sont pas traités symétriquement. Ainsi les opérations "push" et "pull" n'ont pas de sens. Les auteurs précisent aussi que les dimensions d'un cube ne sont pas ordonnées entre elles, ce qui implique que l'opération "rotate" n'a pas de sens non plus.

3.2.4 Un calcul pour des informations réparties en niveaux

Le modèle que nous présentons maintenant est proposé par [12, 13], et prend en compte l'aspect granulaire de l'information dans le modèle de données. Les cubes sont manipulés à l'aide d'un calcul paramétré par des fonctions interprétées (fonctions scalaires ou fonctions d'agrégat).

Structure multidimensionnelle Soient deux ensembles de constantes N et V. Soit $\mathcal{L} \subseteq N$, un ensemble de niveaux. Un niveau $l \in \mathcal{L}$ ne peut être utilisé que pour une seule dimension et est associé à un ensemble dénombrable $dom_l \subseteq V$, représentant le domaine de valeurs du niveau l.

Soit les niveaux $l_1, \ldots, l_n, l \in \mathcal{L}$ utilisés pour un cube c à n dimensions. Une référence de cellule de ce cube est notée $[a_1 : v_1, \ldots, a_n : v_n]$ avec $a_i \in N$ et $v_i \in dom_{l_i}$ pour $i \in [1, \ldots, n]$. Chaque $a_i, i \in [1, \ldots, n]$ est un label des axes de c. Le contenu d'une cellule est une constante de dom_l.

pour la traduction de la requête 2.5.

Le cube c est une fonction partielle F_c dont le domaine est un ensemble de références et le codomaine est dom_l.

Exemple 3.10 Des exemples de références du cube *ventes* de la figure II.5 de la page 21 dans ce modèle sont : *[ANNEE : 1994, PIECE : vis, REGION : nord], [ANNEE : 1995, PIECE : clous, REGION : sud], [ANNEE : 1995, PIECE : écrous, REGION : ouest]*, où $ANNEE$, $PIECE$, et $REGION$ sont les labels des axes. La fonction associée au cube *ventes* fait correspondre à la dernière référence la mesure 10. □

Opérations de restructuration L'opération "rotate", ainsi que les opérations "push" et "pull" (de manière restreinte, puisque les cellules ne peuvent contenir plusieurs mesures) sont prises en compte en jouant sur la position syntaxique des variables.

Granularité Chaque dimension d d'un cube est définie par un triplet $\langle L, \preceq_L, Roll_L \rangle$, avec $L \subseteq N$ l'ensemble des niveaux de d, $\preceq_L$ un ordre partiel sur ces niveaux, et $Roll_L$ une famille de fonctions décrivant les groupements réalisés lors des "roll-ups".

Les fonctions de $Roll_L$ définissent les groupements en se basant sur l'ordre $\preceq_L$ de la manière suivante : pour chaque niveau $l_1, l_2 \in L$ tels que $l_1 \preceq_L l_2$, ${Roll_L}_{l_1}^{l_2}$ est une fonction de $Roll_L$ ayant pour domaine dom_{l_1} et pour codomaine dom_{l_2}. Pour chaque $l_1, l_2, l_3 \in L$ tels que $l_1 \preceq_L l_2 \preceq_L l_3$, on a ${Roll_L}_{l_1}^{l_3} = {Roll_L}_{l_2}^{l_3} \circ {Roll_L}_{l_1}^{l_2}$.

Exemple 3.11 Trois dimensions sont nécessaires pour modéliser le cube *ventes* de la figure II.5 de la page 21 dans ce modèle. Ces dimensions sont nommées *temps*, *produit* et *lieu*. Détaillons la dimension *lieu*. *lieu* est caractérisée par le tuple $\langle LIEU, \preceq_{LIEU}, Roll_{LIEU} \rangle$ où $LIEU$ est l'ensemble de niveaux $\{villes, régions, lieu\}$, avec $villes \preceq_{LIEU} régions \preceq_{LIEU} lieu$. A chaque niveau de $LIEU$ correspond un domaine. Par exemple, $dom_{régions} = \{est, ouest, nord, sud\}$. Ainsi, une des fonctions de la famille $Roll_{LIEU}$ est ${Roll_{LIEU}}_{villes}^{régions}$ avec par exemple, ${Roll_{LIEU}}_{villes}^{régions}(lyon) = est$. □

Opérations associées à la granularité Les fonctions *Roll* sont utilisées sous forme d'atomes dans les formules du calcul pour sélectionner de manière relative le niveau auquel on désire obtenir le résultat. La navigation dans les niveaux basée sur les opérations "roll-up" et "drill-down" est donc facilement spécifiable. Les fonctions interprétées paramétrant le calcul servent à spécifier des opérations sur les mesures, et peuvent constituer les informations nécessaires à cette navigation (e.g., fonction d'agrégat employée lors d'un "roll-up"). Ces fonctions sont de deux types : fonctions scalaires ou fonctions d'agrégat. Une fonction scalaire est appliquée directement à une variable, alors qu'une fonction d'agrégat est appliquée au résultat d'une requête, en s'appuyant sur les groupements définis par les fonctions *Roll*.

Schéma et méta-données Les méta-données d'un cube sont les noms de dimension, les noms de niveaux, les attributs servant de labels aux axes. Un cube de nom $c \in N$ de dimension n est décrit par un schéma $c[a_1 : l_1, \ldots, a_n : l_n] : l_0$, où les $a_i \in N, i \in [1, \ldots, n]$ et les $l_i \in \mathcal{L}, i \in [0, \ldots, n]$. Les a_i sont des attributs (au sens relationnel du terme), et les niveaux l_i sont utilisés pour indiquer le domaine de valeurs de chaque attribut (i.e., le type de chaque membre).

Exemple 3.12 Dans ce modèle, le cube *ventes* a pour schéma :
ventes[ANNEE : années, PIECE : pièces, REGION : régions] : numérique, où *ANNEE, PIECE, REGION* sont des attributs, *années et pièces* correspondent respectivement à des niveaux des dimensions *temps* et *produit*, et *numérique* est un niveau prédéfini dont le domaine est constitué des nombres rationnels. □

Opérations classiques Les opérations classiques de sélection, jointure, et renomage se font de la même manière qu'en calcul relationnel, et sont possibles à la fois sur les membres et les mesures. Les opérations de projection, d'union et de différence nécessitent la présence de fonctions externes pour maintenir la dépendance fonctionnelle entre mesures et membres.

Expression de requêtes Le calcul proposé par [12, 13] est paramétré par des fonctions interprétées (scalaires et d'agrégat), pour manipuler ces cubes. Une requête est une expression de la forme : $\{x_1, \ldots, x_n : x \mid \Psi(x, x_1, \ldots, x_n)\}$, où $\Psi(x, x_1, \ldots, x_n)$ est une formule du premier ordre dans laquelle $x, x_1, \ldots, x_n$ sont les seules variables libres. Elle est composée d'atomes d'égalité impliquant des cubes, des fonctions *Roll*, et des fonctions interprétées. Les variables ont pour domaine d'interprétation les mesures et les membres (les éléments de V).

Si $\forall v_1, \ldots, v_n \in V$, il existe au plus un m tel que $\Psi(m, v_1, \ldots, v_n)$ alors la fonction F_c décrivant le contenu du cube c est définie de la façon suivante :

$\forall m, v_1, \ldots, v_n \in V, F_c(v1, \ldots, v_n) = m \Leftrightarrow \Psi(m, v_1, \ldots, v_n)$

Si en revanche $\exists m_1, m_2, v_1, \ldots, v_n \in V$ avec $m_1 \neq m_2$ et tels que $\Psi(m_1, v_1, \ldots, v_n)$ et $\Psi(m_2, v_1, \ldots, v_n)$, alors le résultat de la requête n'est pas défini car elle implique l'association de plusieurs contenus différents à la cellule dont la référence est constituée par $(v1, \ldots, v_n)$.

Pour les requêtes ayant un résultat défini, le calcul est clos. Dans le cas général du calcul proposé par [12, 13], déterminer si une requête a un résultat défini est indécidable. Cette propriété devient décidable sous certaines conditions (e.g., si le calcul est positif et existentiel et n'implique que des fonctions *Roll*). La puissance d'expression du langage dépend des fonctions le paramétrant. Les auteurs indiquent par exemple dans [12] quelles sont les fonctions permettant de rendre ce calcul aussi expressif que le calcul relationnel avec fonctions d'agrégat [33].

Exemple 3.13 Nous exprimons dans le calcul de [12, 13] les requêtes de la section 2. Les requêtes 2.4 et 2.3 ne sont pas exprimables, car le langage ne permet pas de définir l'imbrication, et le modèle de données ne permet pas de placer plusieurs mesures dans une cellule.
Exprimer la requête 2.6 nécessite de se servir du calcul pour définir des groupements non décrits de manière statique, et calculer la moyenne glissante des ventes par région et par pièce pour une fenêtre de deux années consécutives. Ainsi, une première requête a pour but de calculer l'agrégat pour les années 1994 et 1995, et une deuxième le même agrégat pour les années 1995 et 1996. Le résultat désiré est l'union des deux tables calculées (naturellement, ce principe est guère satisfaisant puisqu'il faut écrire autant de requêtes que l'on veut constituer de groupe). Détaillons la requête calculant le résultat pour les années 1994 et 1995. On sélectionne tout d'abord les années 1994 et 1995 :

$$\begin{gathered}\{a, p, r : x \mid \\ x = ventes[ANNEE : a, PIECE : p, REGION : r] \wedge \\ (a = 1994 \vee a = 1995)\}\end{gathered}$$

Ensuite, il reste à sommer le résultat de cette sélection :

$$\begin{gathered}\{p, r : z \mid z = sum(a, p, r : x \mid \\ x = ventes[ANNEE : a, PIECE : p, REGION : r] \wedge \\ (a = 1994 \vee a = 1995))\}\end{gathered}$$

Expliquons informellement le principe de cette requête. Soit T l'ensemble des tuples sur a, p, r, x qui satisfont la sous-requête lorsque les constantes c_p, c_r sont substiuées à p et r. Alors (c_p, c_r, c_z) appartient au résultat de la requête si c_z est égal à la somme des dernières composantes des tuple de T.
Pour pouvoir calculer le total des ventes par mois et par semaine (requête 2.2), nous considérons la définition suivante de la dimension *temps*. Elle est modélisée par un triplet $\langle T, \preceq_T, Roll_T \rangle$, où T est l'ensemble de niveaux $\{jours, semaines, mois, années, temps\}$, avec $jours \preceq_T semaines$, $jours \preceq_T mois$, $semaines \preceq_T années$, $mois \preceq_T années$, et $années \preceq_T temps$.
Les domaines associés à chaque niveau sont les suivants :
$dom_{jours} = \{1/1/94, \ldots, 31/12/96\}$,
$dom_{semaines} = \{sem1/94, \ldots, sem52/dec96\}$,
$dom_{mois} = \{jan94, \ldots, dec96\}$,

$dom_{années} = \{1994, 1995, 1996\}$, et
$dom_{temps} = \{temps\}$.
En supposant qu'un cube *ventes jours* contienne les ventes quotidiennes de pièces dans chaque région, nous exprimons la requête 2.2 qui calcule le total des ventes par semaines (la requête calculant le total des ventes par mois s'en dérivant trivialement) :

$$\begin{gathered}\{s, p', r : z \mid z = sum(j, p, r : y \mid \\ y = ventes\ jours[JOUR : j, PIECE : p, REGION : r] \wedge \\ s = Roll_{temps_{jours}^{semaines}}(j)) \wedge p' = Roll_{produit_{pièce}^{produit}}(p))\}\end{gathered}$$

Considérons maintenant que les données de base se trouvent au niveau des villes, et que celles-ci sont décrites dans un cube *ventes villes* (figure II.16 de la page 29). Supposons que l'on cherche à exprimer la requête 2.1 à partir du cube *ventes* pour trouver le nombre de villes où il a été vendu moins de 20 unités de l'une des trois pièces en 1996 (nous utilisons le principe déjà employé pour traduire cette même requête lors de l'exemple 3.5 de la page 46). On cherche tout d'abord le détail des quantités vendues par villes, qu'on sélectionne inférieur à 20. On obtient un cube intermédiaire bidimensionnel représentant les ventes par ville et par pièce satisfaisant ces conditions. On agrège alors en regroupant tous les pays. Il ne reste qu'à compter les mesures présentes dans ce cube en regroupant toutes les villes. La requête s'écrit :

$$\begin{gathered}\{l : z \mid z = count(v, p' : y' \mid y' = count(v, p : y \mid \\ \exists r, x(x = ventes[ANNEE : 1996, PIECE : p, REGION : r] \wedge \\ y = ventes\ villes[ANNEE : 1996, PIECE : p, VILLE : v] \wedge \\ r = Roll_{lieu_{villes}^{régions}}(v) \wedge y < 20) \wedge l = Roll_{lieu_{villes}^{lieu}}(v) \wedge p' = \\ Roll_{produit_{pièces}^{produit}}(p))))\}\end{gathered}$$

La traduction de la requête 2.5 se déduit trivialement des traductions proposées ci-dessus. En revanche, exprimer la requête 2.7 ne peut se faire qu'au prix d'un codage très lourd (i.e., en écrivant autant de requêtes que de groupements). □

Le calcul de [12, 13] offre un langage déclaratif pour spécifier des manipulations OLAP. Un aspect intéressant est la possibilité de décrire dans le modèle les hiérarchies, ce qui évite leur spécification dans un langage externe et autorise la prise en compte des groupements en vue d'optimiser les requêtes. Néanmoins, le caractère statique de cette description rend délicate la spécification des groupements non prévus initialement, et donc des agrégats ad-hoc (comme illustré par la traduction de la requête 2.6). Les opérations comme "switch" ou "nest" n'ont pas de sens dans ce modèle.

3.2.5 Algèbre et calcul pour des cubes vus comme des ensembles de relations

Nous abordons finalement le modèle proposé par [23], dans lequel le contenu d'un cube est dissocié clairement de sa structure multidimensionnelle. Un cube est en effet composé d'un contenu, correspondant au contenu d'un ensemble de relations (au sens du modèle relationnel), et d'une structure, c'est à dire sa représentation (le nombre de ses dimensions, la répartition du contenu en mesures et membres)[13]. L'algèbre proposée par [23] sur ce modèle exploite cette dissociation entre un cube et les relations qui décrivent son contenu. Elle comporte 11 opérateurs, dont certains agissent seulement sur la structure, et les autres seulement sur le contenu des cubes.

Structure multidimensionnelle Les symboles utilisés sont pris dans deux ensembles N de noms, et V de valeurs, auxquels est ajoutée une constante $\perp$ pour décrire les valeurs nulles (inapplicables ou inconnues).

Une dimension est définie par son nom $d \in N$ et par une relation (au sens du modèle relationnel) r_d. Chaque tuple de r_d décrit un membre de d et possède un identifiant unique. Ainsi, chaque relation r_d possède un attribut clé nommé T_{id}, dont les valeurs sont différentes deux à deux pour tout couple de relations différentes.

Une référence de cellule est un produit cartésien d'identifiants de plusieurs dimensions différentes. Une cellule est notée $(v_1, \ldots, v_n)$ avec $v_i \in V, i \in [1, \ldots, n]$.

Le contenu d'un cube c à n dimensions $d_1, \ldots, d_n$ est un ensemble de n relations, $r_{d_1}, \ldots, r_{d_n}$ pour décrire les membres, et une relation r_m pour décrire les mesures.

La relation r_m fait le lien entre les références des cellules de c et leurs contenus. Elle possède ainsi chaque attribut T_{id} de chaque relation $r_{d_i}, i \in [1, \ldots, n]$ pour décrire les références, et d'autres attributs pour décrire les valeurs des mesures.

La structure du cube c est un choix particulier de représentation des données contenues dans les relations $r_{d_1}, \ldots, r_{d_n}, r_m$. Cette représentation est augmentée au besoin par des constantes $\perp$, de manière à décrire toutes les combinaisons des membres contenus dans les relations $r_{d_1}, \ldots, r_{d_n}$.

Exemple 3.14 Nous décrivons le cube *ventes* de la figure II.5 de la page 21 dans ce modèle. Soit les trois relations nommées *temps*, *produit* et *lieu*. Chaque dimension est décrite par une de ces relations, comme le montre la figure II.36. Le contenu du cube *ventes* est fixé par ces trois relations plus

13. On prendra garde à ne pas confondre le terme "mesures", que nous employons pour dénoter l'information représentée à l'intérieur d'une cellule d'un cube, avec ce que [23] appelle "contenu" et qui pour leur modèle englobe ce que nous appelons membres et mesures.

la relation r_m (figure II.37). La structure du cube est précisé par la figure II.38. Une cellule de ce cube est $(1995, \acute{e}crous, ouest, 10)$. □

produit

T_{id}	pièces
p1	écrous
p2	clous
p3	vis

lieu

T_{id}	régions	villes
l1	est	lyon
l2	est	dijon
⋮	⋮	⋮
l7	nord	paris
⋮	⋮	⋮

temps

T_{id}	années
t1	1994
t2	1995
t3	1996

FIG. II.36 - *Le contenu des dimensions du cube* ventes

r_m

$temps.T_{id}$	$produit.T_{id}$	$lieu.T_{id}$	quantité
t1	p1	l1	60
t1	p1	l2	40
⋮	⋮	⋮	⋮
t2	p3	l7	40
⋮	⋮	⋮	⋮

FIG. II.37 - *Le contenu des cellules du cube* ventes

Opérations de restructuration Pour [23], l'animation d'un cube concerne seulement la structure multidimensionnelle de ce cube (e.g., passage d'un cube bidimensionnelle à un cube tridimensionnelle). Les opérations de restructuration modifient donc le choix de représentation du cube et laissent son contenu inchangé.

L'animation des cubes se fait par le biais de deux opérateurs de restructuration. Le premier opérateur permet de faire passer toute une dimension comme mesure. Le deuxième permet de sélectionner un ou plusieurs attributs parmi les mesures et de constituer une dimension à partir de ces attributs. Ces deux opérateurs permettent de décrire directement les opérations "push", "pull", et de manière indirecte l'opération "rotate". Leur composition permet en plus de prendre en compte l'opération "nest".

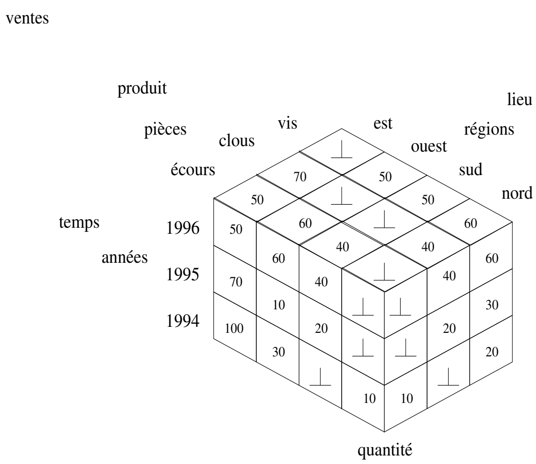

FIG. II.38 - *La structure du cube* ventes

Granularité La granularité n'est pas prise en compte dans le modèle de données. En revanche, l'aspect granulaire est pris en compte à travers deux opérateurs paramétrés par des fonctions externes au langage de requêtes. Ces opérateurs ne changent pas la structure multidimensionnelle d'un cube, mais simplement son contenu (membres et mesures) et ne concernent donc que les relations sous-jacentes.

Un opérateur de classification permet de spécifier des groupements de tuples, et un opérateur de consolidation permet d'appliquer une fonction sur des groupes de tuples.

Schéma et méta-données Les méta-données d'un cube sont des symboles de N qui décrivent les attributs (au sens du modèle relationnel) et les noms des relations sous-jacentes à un cube. Les tuples composant ces relations sont formés de symboles de V.

Le schéma d'un cube c de dimensions n est noté (D, R, att), où $D = \{d_1, \ldots, d_n\}$ est l'ensemble des n noms de dimensions, $R = \{A_1, \ldots, A_m\}$ est un ensemble de nom d'attributs pris dans N, et att est une fonction ayant pour domaine D et pour codomaine les parties de R. La fonction att associe à chaque nom de dimension $d_i, i \in [1, \ldots, n]$ l'ensemble des attributs de la relation $r_{d_i}, i \in [1, \ldots, n]$ pris dans R, excepté l'attribut T_{id}.

Exemple 3.15 Le schéma du cube *ventes* décrit dans l'exemple 3.14 est : (D, R, par) avec $D = \{temps, produit, lieu\}$, $R = \{$*années, pièces, régions, villes, quantité*$\}$, et par exemple *par(temps) = années*. □

Opérations classiques Les auteurs précisent que les opérations classiques de l'algèbre relationnelle opèrent uniquement sur le contenu des cubes et laissent inchangée sa structure multidimensionnelle.

Ainsi les opérations relationnelles classiques de sélection, projection, renommage, union, intersection, différence et produit cartésien sur un cube sont directement traduites en opérations de l'algèbre relationnelle sur les relations sous-jacentes au cube.

Expression de requêtes L'algèbre proposée par [23] pour manipuler des cubes est un langage clos, dont la puissance d'expression n'a pas été caractérisée. [23] propose aussi un calcul équivalent à l'algèbre qui, au contraire de l'algèbre, ne repose pas sur la dissociation forme-contenu. Ce calcul est très proche de celui proposé par [12, 13], privé des fonctions *Roll*. Pour illustrer l'expression de requêtes, nous avons donc choisi de présenter seulement les caractéristiques de l'algèbre, qui nous semblent plus spécifiques.

Exemple 3.16 Nous traduisons dans l'algèbre de [23] les requêtes de la section 2. La requête 2.4 ne peut être traduite car le langage de [23] ne permet pas de différencier les lignes et colonnes d'un cube bidimensionnelle. Commençons par la requête 2.6 visant à calculer la moyenne glissante des ventes par région et par pièce pour une fenêtre de deux années consécutives. La traduction de cette requête illustre l'utilisation des opérateurs K et C, respectivement opérateur de classification et opérateur d'agrégation. La requête s'exprime simplement par :

$$C(K(ventes, f_{gliss}), g_{moy})$$

où f_{gliss} est une fonction externe paramétrant l'opérateur de classification K, et spécifie le regroupement des années 1994 et 1995 d'une part, 1995 et 1996 d'autre part. g_{moy} est une fonction externe paramétrant l'opérateur de consolidation C, et réalisant le calcul de la moyenne des ventes pour ces groupements.

Nous traduisons ensuite la requête 2.3, qui spécifie le calcul du chiffre d'affaires réalisé par pièce, région et année. Pour traduire cette requête, nous nous référons au cube *prix* qui détaille le prix de chaque pièce par année (une représentation d'un tel cube restreint à l'année 1996 est donnée dans la figure II.20 de la page 32). Le principe est le suivant. Afin de faire la jointure des cubes *ventes* et *prix*, on renomme d'abord les attributs *années* et *pièces* du cube *prix* avec l'opérateur de renommage (ρ) :

$$\rho_{ann \leftarrow années}(\rho_{prod \leftarrow pièces}(prix))$$

Ceci fait, on effectue le produit cartésien des cubes *prix* et *ventes*, puis une sélection (σ) pour terminer la jointure des deux cubes :

$$\sigma_{ann=années, prod=pièces}(\\ ventes \times (\rho_{ann \leftarrow années}(\rho_{prod \leftarrow pièces}(prix))))$$

On utilise ensuite l'opérateur de consolidation pour multiplier les mesures entre elles (il s'agit simplement de faire appel à une fonction externe). Ce qui donne finalement :

$$
\begin{array}{l}
C(\\
\quad \sigma_{ann=années,prod=pièces}(\\
\qquad ventes \times (\rho_{ann\leftarrow années}(\rho_{prod\leftarrow pièces}(prix)))), \\
\qquad\quad g_{mult})
\end{array}
\qquad (C)
$$

où g_{mult} est une fonction externe qui multiplie chaque mesure quantité d'une cellule avec la mesure $prix$ de cette cellule. La nouvelle valeur calculée, le chiffre d'affaires, constitue une nouvelle mesure du cube obtenu.
Nous terminons par la requête 2.1 pour trouver le nombre de villes où il a été vendu moins de 20 unités de l'une des trois pièces en 1996 (nous utilisons le principe déjà employé pour traduire cette même requête lors de l'exemple 3.5). Le principe consiste d'abord à sélectionner pour le cube $ventes$ les quantités vendues inférieures à 20 en 1996. On applique ensuite l'opérateur de classification K pour spécifier le regroupement des toutes les pièces, et l'opérateur de consolidation C pour décompter les pièces. On répète les deux dernières opérations pour décompter les villes.

$$
\begin{array}{ll}
C(& \\
\quad K(& \\
\quad\quad C(& \\
\quad\quad\quad K(& \\
\quad\quad\quad\quad \sigma_{quantité<20}(& \\
\quad\quad\quad\quad\quad \sigma_{années=1996}(ventes)), & \\
\quad\quad\quad\quad\quad\quad f_{pièce}), & \text{(premier } K\text{)} \\
\quad\quad\quad\quad\quad\quad\quad g_{pièce}), & \text{(premier } C\text{)} \\
\quad\quad\quad\quad\quad\quad\quad\quad f_{villes}), & \text{(deuxième } K\text{)} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad g_{villes}) & \text{(deuxième } C\text{)}
\end{array}
$$

où $f_{pièce}$ et f_{villes} spécifient les groupements des pièces et des villes, $g_{pièce}$ compte le nombre de ventes pour chaque pièce, et g_{villes} compte le nombre de mesures restantes (soit le nombre de villes).
La traduction des requêtes 2.2, 2.5 et 2.7 se déduit aisément de celles proposées ci-dessus, sachant que traduire la requête 2.7 ne peut se faire qu'à raison d'une requête par année et nécessite l'union des résultats intermédiaires. □

Dans le modèle et l'algèbre que proposent [23], l'idée de dissocier structure et contenu permet une simplicité dans la définition des opérateurs algébriques et suggère une implantation naturelle dans une base de données relationnelle. Cette proximité avec le modèle relationnel permet aussi de profiter directement des travaux existants. Le calcul, détaché de la dissociation contenu/structure, offre une contrepartie déclarative à l'algèbre.

Comme pour [6, 7], la définition des groupements se fait via des fonctions externes au langage. Le concept de structure n'incorpore pas l'ordre des membres, donc la position relatives des cellules n'est pas prise en compte (en conséquence, l'opération "switch" n'a pas de sens pour ce modèle). Seulement deux opérateurs de restructuration sont utilisés pour exprimer quatre opérations élémentaires ("push", "pull", "nest", "rotate"). Cette polyvalence semble préjudiciable à la lisibilité des requêtes, et pourrait gêner un processus d'optimisation basé sur une dissociation des opérations élémentaires.

4 Conclusion

Nous avons présenté une synthèse des travaux concernant la manipulation de données multidimensionnelles utilisée pour l'analyse en ligne des données (manipulations OLAP). La prise en compte de traitements OLAP constituent l'évolution naturelle des bases de données traditionnelles vers les systèmes d'aide à la décision. Nous avons vu que ces traitements supposent la définition de différents points de vue, pour présenter les données à différents niveaux de détail où selon différentes catégories. Ainsi, la modélisation privilégiée et celle du cube de données, dont chaque dimension est hiérarchisée.

Nous avons décrit les opérations élémentaires typiques, et étudié les langages clairement formalisés ayant été proposés, à notre connaissance, pour modéliser les manipulations OLAP. Ces langages, quatre algèbres et deux calculs (dont un équivalent à une des algèbres), couvrent une grande partie des opérations OLAP, tout en gardant leurs propres spécificités. Les principales caractéristiques des travaux étudiés sont résumés sur la figure II.39.

Nous pouvons dégager quelques points communs :

- le souci de rester proche du modèle relationnel, afin de profiter de techniques maîtrisées ou de proposer une implantation à partir de bases existantes. Ainsi [6, 7] proposent une traduction de leurs opérateurs en SQL. [23, 35] basent leurs modèles respectifs sur des relations et utilisent tels quels les opérateurs algébriques relationnels. [35] se distingue de [23] en permettant de manipuler directement les relations qui servent de base à un cube ;

- le choix d'une approche nommée (i.e., la prise en compte de noms d'attributs ou de dimensions dans les requêtes) ;

- l'incorporation de la dépendance fonctionnelle entre références et mesures directement dans la définition d'un cube. On remarquera que chez [6, 7] cette dépendance est maintenue grâce aux fonctions externes paramettrant les opérateur (e.g., lors de "roll-up"), et que pour [23] elle nécessite l'emploi d'identifiants artificiels. Chez [12, 13], cette dépendance est incorporée à la sémantique du langage ;

- la majorité des travaux ne prennent pas en compte l'aspect animation dans sa totalité. Néanmoins, [24] fait exception, et [12, 13], s'il n'en

	Gyssens & al. [24]	Agrawal & al. [6, 7]	Li & Wang [35]	Cabibbo & Torlone [12, 13]	Gyssens & Laksh. [23]
type	algèbre	algèbre	calcul	algèbre calcul	algèbre
symétrie membre / mesures	oui	oui	non	oui	oui
restructuration directe (sans codage)	rotate split switch	pull push	nest	pull push rotate	nest pull push
représentation directe des hiérarchies	non	non	oui	oui	non
construction de hiérarchies par requêtes	non	non	oui	non	non
correspondance donnée avec le relationnel	non	oui	oui	non	oui

FIG. II.39 - *Principales caractéristiques des travaux étudiés*

tiennent pas compte dans le calcul qu'ils proposent, reconnaissent l'importance de l'animation dans un langage de requête ;

– en général, la spécification des hiérarchies est impossible via le langage de requêtes. [6, 7, 23] proposent de spécifier les groupements via des fonctions externes à leur langage respectif. Chez [12, 13] et [35], les hiérarchies sont décrites dans le modèle de données. [35] est le seul travail à proposer de décrire des hiérarchies par requête.

Cette synthèse nous a permis de cerner les caractéristiques fondamentales d'un langage de requêtes pour systèmes OLAP. La section 3 de ce chapitre laisse apparaître que :

1. aucun des langages étudiés ne permet d'exprimer directement des traitements récursifs, et aucun langage à base de règles n'a été proposé ;

2. aucun des langages étudiés ne permet d'exprimer de manière directe la totalité des requêtes de la section 2.4. Notamment, il n'a été proposé aucun langage permettant de décrire de manière précise la structure[14]

14. au sens des matrices bidimensionnelles de [24] où la position des membres et des mesures est prise en compte.

du résultat d'une requête avec agrégats;

3. il n'a été proposé aucun langage déclaratif[15] intégrant des possibilités de spécification de construction de hiérarchies ad-hoc sans recours à des fonctions externes.

Nous pensons que ces points constituent des axes de recherche importants pour la compréhension de la problématique OLAP au niveau logique. Nous proposons dans la suite de ce mémoire d'intégrer ces caractéristiques au sein d'un même formalisme, et nous montrons qu'un cadre basé sur Datalog est particulièrement approprié pour spécifier les manipulations OLAP typiques, qu'elles soient élémentaires ou complexes. Nous montrons notamment qu'un tel cadre offre la possibilité de spécifier la totalité des requêtes de la section 2.4, et reste suffisamment proche du cadre relationnel classique pour envisager l'adaptation des nombreux travaux existants (e.g., optimisation de requêtes récursives, sémantiques sophistiquées pour l'agrégation ou la négation).

15. i.e., ayant une sémantique purement déclarative, à la différence de l'algèbre de [35] dont la sémantique est constructive.

Le chapitre qui suit synthétisant et étendant les travaux publiés dans [25, 27, 26, 28], sa présentation restera en anglais. Il offre un point de vue homogène sur nos réponses aux questions ouvertes du chapitre II.

Chapitre III

Cadre proposé

1 Introduction

On-Line Analytical Processing (OLAP) has emerged to support multidimensional data analysis, by providing manipulations and aggregations of data according to multiple dimensions. Foundations of OLAP languages are now a growing field of interest in the database research community. This work is a contribution in this direction. It introduces an extension of Datalog devoted to the manipulation of data organized in multidimensional cubes. This extension is based on the point of view that a Datalog fact represents an entry (called *cell reference*) in a cube. The resulting language allows to define intuitively relationships between cells, and provides a declarative way to specify:

1. all the basic cube restructuring operations used in OLAP systems, and
2. complex summarizations of data according to ad-hoc aggregation levels.

The semantics of the language has been set up by combining techniques stemming from previous works done in the area of databases and logic programming:

- A higher-order syntax as in Hilog [16] to allow schema browsing;
- A specific relation to specify the aggregation levels, in the spirit of the *isa* hierarchy of F-logic [32];
- The semantics of *monovaluation* as in Datalog with single-valued data functions (Datalog+sdf) [3] to ensure the association of each cell reference with an unique contents.

It should be noticed that our aim is not to provide here a new semantics for aggregation through recursion. Such semantics exist for languages with second order syntax and first order semantics like Hilog [45], and can also be used in our case. For the sake of simplicity, we chose to adopt a semantics in the spirit of *semipositive Datalog*$^{\neg}$ [4], where grouping relationships are computed apart from summarization specifications, and without aggregation through recursion.

The main contributions of this work are the following: Firstly, we define a model-theoretic declarative semantics that allows a high level specification of the operations used in OLAP systems. Secondly, we give an equivalent constructive semantics, that is a basis for the reuse of evaluation methods developed for deductive databases. We also use a version of our language to characterize the class of queries computable in polynomial time on ordered multidimensional databases.

The presentation is organized as follows. Section 2 introduces the multidimensional data model we propose. Motivating examples are given in

Section 3 to illustrate informally the most salient features of the data manipulation language. In Section 4 we define its model-theoretic semantics, and an equivalent fixpoint semantics. We characterize the class of tractable multidimensional manipulations in Section 5. A variety of typical complex OLAP manipulations are considered in Section 6. We relate this work to other proposals and conclude in Section 7.

2 Data model overview

In this section, we outline informally the underlying data model of the language.

Names The constants we use in the model are called *atomic names*. Structured names can be built from atomic names using the constructor "$\cdot$", and are called *nested names*. In the following, we use the term *names* to denote both atomic and nested names.

Cells In this multidimensional data model, data are organized in *cells*. A cell is identified by a *cell reference*, and is associated with a unique *cell contents*. A cell reference is of the form $N(N_1, N_2, \ldots, N_p)$, where N, N_1, N_2, $\ldots, N_p$ are names. N is the *cube* name, and $N_1, N_2, \ldots, N_p$ are the *members*.

A cell contents is a tuple of names. Associations of cells contents with cells references are represented by *ground atoms* of the form:

$$N(N_1, N_2, \ldots, N_p) : \langle N_{p+1}, \ldots, N_{p+q}\rangle$$

where the tuple $\langle N_{p+1}, \ldots, N_{p+q}\rangle$ denotes measures (the cell contents). We call this form of atoms *cell atoms*.

Cubes A *cube* is a set of ground cell atoms having both a common cube name and the same number of members, and in which the same reference does not appear more than once to ensure cell monovaluation.

Database A *multidimensional database* is a set of ground cell atoms in which the same reference does not appear more than once.

It should be noted that our model distinguishes clearly a cell that does not exist (i.e., no ground cell atom with this cell reference in the database), from an existing but empty cell, which is represented by a ground cell atom of the form $N(N_1, N_2, \ldots, N_p) : \langle\rangle$.

Example 2.1 Consider the cube *ventes* of figure II.5 page 21. This cube contains information on product sales in areas, over time. In our model, this cube can be represented by:

$$\{ventes(écrous, 1996, est) : \langle 50\rangle,$$

$$\vdots$$

$$ventes(clous, 1995, nord) : \langle 20 \rangle\} \qquad \square$$

Example 2.2 A particular 2-dimensional representation of a cube can be obtained by using nested names. Consider the cube of figure II.9 page 25 which represents a nested organization of the data of cube *ventes*. Using nested names, this new cube, called *ventes2D* can be described in the model by the set:

$$\{ventes2D(écrous \cdot est, 1996) : \langle 50 \rangle,$$
$$\vdots$$
$$ventes2D(clous \cdot nord, 1995) : \langle 20 \rangle\} \qquad \square$$

3 Motivating examples

In this section, we present informally the syntax and the semantics of the language, and show how basic OLAP operations can be expressed.

3.1 Intuitive syntax and semantics

Rules à la Datalog are used to define new cell references and their associated contents from existing cells. The higher order syntax stemming from Hilog [16] allows variables to range over every atomic names used in cell references or cell contents. It should be noted that variables cannot range over nested names but only over atomic names.

We adopt the following conventions: symbols beginning with an upper-case letter denote variables, and symbols beginning with a lower-case letter or a digit denote constants.

Intuitive meaning Consider the rule $p(X) \longleftarrow q(X, Y), r(Y)$. The standard (Datalog) informal meaning of this rule is *if q(X, Y) holds and r(Y) holds, then p(X) holds.* The basic intuition of our extension is to read such a rule as follows: *if there are two cells of references q(X, Y) and r(Y), then there is a cell of reference p(X).* We also add the handling of cell contents, and then a typical rule will be:

$$p(X) : \langle W \rangle \longleftarrow q(X, Y) : \langle W \rangle, r(Y) : \langle X \rangle.$$

This rule will be informally read: *if there exists a cell of reference q(X, Y) containing W, and if there exists a cell of reference r(Y) containing X, then there exists a cell of reference p(X) containing W.*

3.2 OLAP elementary data manipulations

We now illustrate how rules can specify typical data manipulations used in OLAP systems. We refer to elementary operations introduced in chapter II.

3.2.1 Restructuring cubes

We first show how the basic cube restructuring operations can be trivially specified.

Rotate The following rule can be used to rotate the cube *ventes* of example 2.1, and obtain the cube depicted figure II.6 page 23:

$$rotate(Y, P, R) : \langle S \rangle \longleftarrow \ ventes(P, Y, R) : \langle S \rangle.$$

Split Suppose we want to split the cube *ventes* in order to obtain a cube per region (figure II.8 page 24). We can use a variable ranging over regions to form new cube names. This is shown by the following rule:

$$ventes \cdot R(P, Y) : \langle S \rangle \longleftarrow \ ventes(P, Y, R) : \langle S \rangle.$$

Nest, unnest The nested representation of figure II.9 page 25 can be obtained from the cube *ventes* by the rule:

$$ventes2D(P \cdot R, Y) : \langle S \rangle \longleftarrow \ ventes(P, Y, R) : \langle S \rangle.$$

The inverse operation (unnest) can also be specified easily :

$$ventes(P, Y, R) : \langle S \rangle \longleftarrow \ ventes2D(P \cdot R, Y) : \langle S \rangle.$$

Push, pull Suppose we want to push the years within the cube *ventes* (figure II.10 page 25). This can be done by the following rule, in which $<$ is a built-in predicate having its standard meaning:

$$\begin{aligned} push(P, R) : \langle Y1, S1, Y2, S2, Y3, S3 \rangle \longleftarrow \ & ventes(P, Y1, R) : \langle S1 \rangle, \\ & ventes(P, Y2, R) : \langle S2 \rangle, \\ & ventes(P, Y3, R) : \langle S3 \rangle, \\ & Y1 < Y2, \\ & Y2 < Y3. \end{aligned}$$

The difference made between empty cells and non existing cells can be useful in the case of a pull operation. Suppose we want to pull the measures from the cube *ventes*1996 (figure II.4 page 21) to obtain the cube depicted in figure II.11 page 26. This can be expressed by the following rule:

$$pull(P, S, R) : \langle \rangle \longleftarrow \ ventes1996(P, R) : \langle S \rangle.$$

3.2.2 Summaries specification

We now present how rules can be used to specify summarizations of data at different levels of aggregation. The following examples are based on the 2-dimensional cube named *ventes*96 *villes* that contains information on the 1996 part sales in various cities. This cube is depicted in figure III.1, and can be viewed as a subset of the cube *ventes villes* depicted in figure II.16 page 29.

ventes96 villes	écrous	vis	clous
bordeaux	30		20
dijon	20	10	
grenoble		30	
lille			10
lyon	30	30	
marseille	40		20
montpellier			30
nantes	20		20
paris		40	50
poitiers	10		10

FIG. III.1 - *The cube* ventes96 villes

The specification of summaries requires that the grouping relationship between members is known (e.g., how cities are grouped in regions). Such a relationship is depicted in figure II.12 page 27. In this example, cities can be grouped in regions, and regions can be grouped to form the whole area called *lieu*. In the same way, the different parts can be grouped to form the whole production called *produit*.

The grouping relationship can be specified by rules. This relationship is represented by particular literals of the form $in(\alpha, \beta)$, where α and β are atomic names. For example the grouping relationship of figure II.12 is described by the following set of ground grouping atoms:

$$\{in(\acute{e}crous, produit), \ldots, in(marseille, sud), \ldots, in(sud, lieu)\}$$

Aggregate subgoals Aggregate subgoals used to specify the summaries are of the form: $T = f(N(N_1, \ldots, N_p))$ where T is a constant or a variable, f is an aggregate operator (e.g., *sum, min*) and $N(N_1, \ldots, N_p)$ is a possibly non ground cell reference.

Their intuitive meaning is illustrated by the following example. Consider the ground aggregate subgoal $50 = sum(ventes96Villes(\acute{e}crous, est))$. It holds if the sum of the sales of *écrous* over the cities *in* region *est* is equal to 50.

More precisely, consider the set $detailRef(ventes96Villes(\acute{e}crous, est))$ that contains the references of the existing cells corresponding to the lowest level of description for the sales of *écrous* in region *est*. According to the grouping relationship depicted in figure II.12 page 27, we have:

$$detailRef(ventes96Villes(\acute{e}crous, est)) = \{ventes96Villes(\acute{e}crous, lyon), ventes96Villes(\acute{e}crous, dijon)\}.$$

It should be noticed that $ventes96Villes(\acute{e}crous, grenoble)$ does not belong to this set, since no cell with this reference exists in the database.

Consider $detailCont(ventes96Villes(écrous, est))$ defined as the multi-set formed with the cell contents of the cell references in $detailRef(ventes96 Villes(écrous, est))$. We have:

$detailCont(ventes96Villes(écrous, est)) = \{\langle 30\rangle, \langle 20\rangle\}$.

The semantics of the aggregate subgoal can now be stated more precisely: $50 = sum(ventes96Villes(écrous, est))$ holds if the sum of the elements in $detailCont(ventes96Villes(écrous, est))$ is equal to 50.

It should be noticed that $detailRef$ collects the most detailed existing description according to all coordinates. Consider for example the set $detailRef(ventes96Villes(produit, lieu))$. We have:

$$
\begin{aligned}
&detailRef(ventes96Villes(produit, lieu)) = \\
&\quad \{ventes96Villes(écrous, bordeaux), \\
&\quad ventes96Villes(écrous, dijon), \\
&\quad \quad \vdots \\
&\quad ventes96Villes(clous, nantes), \\
&\quad ventes96Villes(écrous, paris)\}.
\end{aligned}
$$

Hence $sum(ventes96Villes(produit, lieu))$ denotes the total of sales for the whole production over the whole area.

Let us now illustrate the use of aggregate subgoals, using the grouping relationship depicted in figure II.12.

Simple roll-ups Suppose we want to roll-up the data of cube *ventes* to obtain a cube containing the global sales of part *écrous* in the *sud* area. We simply use the following rule:

$total(écrous, sud) : \langle T\rangle \longleftarrow T = sum(ventes(écrous, temps, sud))$.

Suppose now we want to obtain the global sales for each part in each region (figure II.13 page 27). This can be done with the following rule, where we use the in relation to restrict the variables P and R to range respectively over parts and regions:

$$
\begin{aligned}
ventes94/96(P, temps, R) : \langle T\rangle \longleftarrow\ & T = sum(ventes(P, temps, R)), \\
& in(P, produit), in(R, lieu).
\end{aligned}
$$

It is to be noted that, following the same principle, the cube *ventes* can be obtained by rolling-up the data of cube *ventes villes* to the different regions:

$$
\begin{aligned}
ventes(P, R, Y) : \langle T\rangle \longleftarrow\ & T = sum(ventesVilles(P, V, Y)), \\
& in(P, produit), in(R, lieu), \\
& in(Y, années).
\end{aligned}
$$

Multiple roll-up Computing the 1996 global sales by region from the cube *ventes96 villes* can be done by rolling-up simultaneously from the part names to the whole production, and from the cities to the regions:

$$ventesRégion(R, total) : \langle T \rangle \longleftarrow \; T = sum(ventes96Villes(produit, R)), \\ in(R, lieu).$$

A graphical counterpart of the cube *ventesRégion* is represented in figure III.2.

ventes Région	total
est	120
ouest	110
sud	90
nord	100

FIG. III.2 - *The cube* ventesRégion

3.2.3 Relational operations

We simply illustrate here a combination of selection and projection called *slicing-dicing* in the OLAP literature.

Suppose we are interested in a subset (a slice) of the cube *ventes*, that contains only the data for the year 1996 (figure II.4 page 21). We can use the following rule that both selects year 1996, and projects on the product and region dimensions (a similar rule can be used to obtain the cube *ventes96 villes* from the cube *ventes villes*):

$$ventes1996(P, R) : \langle S \rangle \longleftarrow \; ventes(P, 1996, R) : \langle S \rangle.$$

The other relational operations are taken into account straightforwardly.

4 Formal syntax and semantics

In this section, we formally present the syntax and the semantics of the language.

4.1 Syntax

Constants and variables Let $\mathcal{D}$ be a countably infinite set of constants called atomic names, and $\mathcal{V}$ be a countably infinite set of variables, disjoint from $\mathcal{D}$.

Aggregates An aggregate operator f is a partial mapping from multisets of tuples over $\mathcal{D}$ to a single value in $\mathcal{D}$. Let $\mathcal{AGG}$ be a set of aggregate operators.

Rule-based language The syntactical expressions allowed in the rule-based language are:

$$
\begin{array}{lcl}
atomicName & := & c \mid v \\
name & := & atomicName \mid name \cdot name \\
contents & := & \langle name, \ldots, name \rangle \\
reference & := & name(name, \ldots, name) \\
cellAtom & := & reference : contents \\
groupingAtom & := & in(atomicName, atomicName) \\
atom & := & cellAtom \mid groupingAtom \\
aggregateSubgoal & := & atomicName = f(reference) \\
literal & := & atom \mid aggregateSubgoal \\
body & := & literal, \ldots, literal \mid \epsilon \\
head & := & atom \\
rule & := & head \longleftarrow \ body
\end{array}
$$

where $c \in \mathcal{D}, v \in \mathcal{V}$, and $f \in \mathcal{AGG}$.

Definitions We note $ref(A)$ the reference part of a cell atom or aggregate subgoal A, and $val(A)$ the contents part of a cell atom A. Let var be a computable function that assigns to each syntactical expression the subset of $\mathcal{V}$ corresponding to the set of variables occurring in this expression. var is extended in a straightforward manner to sets of syntactical expressions. A ground name (resp. reference, literal) is a name n (resp. reference rf, literal l) for which $var(n) = \emptyset$ (resp. $var(rf) = \emptyset$, $var(l) = \emptyset$).

Range restricted rule A range restricted rule is a rule $r = A \longleftarrow B_1, \ldots, B_n$ where:

- $var(A) \subseteq var((\{B_1, \ldots, B_n\}))$, and
- let $\mathcal{A}g$ be the set of references occurring in aggregate subgoals in r, and $\mathcal{B}$ be the set of atoms occurring in the body of r, then $var(\mathcal{A}g) \subseteq var(\mathcal{B})$.

Restructuring programs A restructuring program is a set of range restricted rules having *no aggregate subgoal in their body.*

Summarizing programs A summarizing program is a set of range restricted rules having *no grouping atom in their head.*

OLAP programs An OLAP program is a pair $\langle R, S \rangle$, where R is a restructuring program, and S is a summarizing program. Informally, the semantics of an OLAP program $\langle R, S \rangle$ will be to consider R and S in two stages. The restructuring program R will be used first, and then S will be applied.

4.2 Semantics

In this section, we give a declarative model-theoretic semantics and an equivalent fixpoint-based constructive semantics. Our presentation is inspired by that of both [4] and [3].

Input The semantics of a program (restructuring, summarizing or OLAP program) is given with respect to a set of ground atoms describing the *extensional* part of the database. Following [4], we call this set an *input*.

4.2.1 Model-theoretic semantics

Grouping relationship Let J be a set of ground atoms. Then in_J is the relation corresponding to the description of the grouping atoms within J. More precisely, we define the relation in_J by: $\forall x, y \in \mathcal{D}, in_J(x, y) \iff in(x, y) \in J$.

Extending the grouping relationship over references Let J be a set of ground atoms. We define a relation noted $<_J$ over references in the following way: for all references rf and rf', with $rf = n(n_1, \dots, n_p)$ and $rf' = n(n'_1, \dots, n'_p)$, then $rf <_J rf' \iff rf \neq rf'$ and $\forall i \in [1, \dots, p]$, either $in_J(n_i, n'_i)$ holds, or $n_i = n'_i$.

Informally, we have $rf <_J rf'$ when rf references a cell at a strictly lower level of detail than rf', with respect to the grouping relationship defined in J.

Interpretation with respect to an input Let I be an input. A set J of ground atoms is an *interpretation* with respect to I if the following conditions are satisfied:

- For all A_1, A_2 cell atoms in J, $ref(A_1) = ref(A_2) \implies A_1 = A_2$, where "=" is the syntactical equality. This *consistency* criterion is drawn from the semantics of Datalog with *single-valued data functions* [3]. It guarantees that a cell is associated with an unique cell contents;
- The transitive closure of in_J is irreflexive. This ensures that the grouping relationship has no directed cycle;
- $I \subseteq J$ (i.e., if something holds in the input, it also holds in the interpretation).

Most detailed information The satisfaction of a ground aggregate subgoal of the form $k = f(n(n_1, \ldots, n_p))$ depends on the most detailed information available for $n(n_1, \ldots, n_p)$ *in the input*. This is formalized by two functions: *detailRef* and *detailCont*. $detailRef(n(n_1, \ldots, n_p))$ gives the set of cell references of the most detailed information available for $n(n_1, \ldots, n_p)$ in the input, and $detailCont(n(n_1, \ldots, n_p))$ gives the multiset of the contents of these cells. More formally: let J be an interpretation with respect to an input I. Let B be a ground aggregate subgoal. We define the set $detailRef_I^J(B) = \{ref(A) \mid A \in I, ref(A) <_J ref(B)\}$, and the multiset $detailCont_I^J(B) = \{\!\!\{ k \mid A \in I, ref(A) \in detailRef_I^J(B), k = val(A) \}\!\!\}$.

Valuation A valuation ν is a total function from $\mathcal{V}$ into $\mathcal{D}$. ν is extended to be the identity on $\mathcal{D}$. ν is also extended in a straightforward manner to names, literals, and rules.

Satisfaction Let J be an interpretation with respect to an input I. J satisfies a ground literal B with respect to I, denoted $J \models_I B$, iff

- B is a ground atom and $B \in J$, or
- B is a ground aggregate subgoal of the form $k = f(n(n_1, \ldots, n_p))$, such that $detailRef_I^J(B) \neq \emptyset$, $f(detailCont_I^J(B))$ is defined, and such that $f(detailCont_I^J(B)) = k$.

Remark A It should be noticed that for the sake of simplicity our semantics restricts the use of aggregate subgoals to the input. This simplifies the semantics but limits the expressiveness of a summarizing program. Our aim was not to provide a new semantics for aggregation through recursion, and such existing semantics (e.g., [45]) can be adapted if needed. □

Let $r = A \longleftarrow B_1, \ldots, B_n$ be a rule, and I be an input. An interpretation J satisfies r with respect to I, denoted $J \models_I r$ iff for each valuation ν we have:

- $J \models_I \nu(A)$, or
- $\exists B_i, i \in [1, \ldots, n], J \not\models_I \nu(B_i)$.

Model of a restructuring or summarizing program An interpretation J is a model of a restructuring or summarizing program P with respect to an input I, denoted $J \models_I P$, iff $\forall r \in P, J \models_I r$.

Remark B It should be noticed that even simple programs may have no model:

- Inconsistency may arise as it is the case in other languages that allow some kind of monovaluation (e.g., Datalog with single-valued data functions [3], COL [2]). For example, the following program defines two different cell contents for the same cell reference and thus it has no model:

$a(b,c) : \langle e \rangle \longleftarrow\ .$
$a(b,c) : \langle d \rangle \longleftarrow\ .$

- Inconsistency may also arise from the specification of the grouping relationship. As in other languages that handle some kind of hierarchy (e.g., the *subclass* relationship in F-logic [32]), a program like:

  $in(a,b) \longleftarrow\ .$
  $in(b,a) \longleftarrow\ .$

  will have no model, since it specifies a directed cycle in the grouping relationship.

□

Remark C We insist on the fact that the valuations map variables of $\mathcal{V}$ only to constants of $\mathcal{D}$. They do not map variables of $\mathcal{V}$ to names constructed with "$\cdot$". This guarantees that if a program admits a model then it admits also a finite model. Consider the following program:
$a(b,c) : \langle e \rangle \longleftarrow\ .$
$a(X \cdot b, c) : \langle e \rangle \longleftarrow\ a(X,c) : \langle e \rangle.$
$\{\ a(b,c) : \langle e \rangle,\ a(b \cdot b, c) : \langle e \rangle\ \}$ is a finite model of the program since no valuation can map X to $b \cdot b$. The infinite interpretation $\{\ a(b,c) : \langle e \rangle,$ $a(b \cdot b, c) : \langle e \rangle,\ a(b \cdot b \cdot b, c) : \langle e \rangle,\ a(b \cdot b \cdot b \cdot b, c) : \langle e \rangle, \ldots\ \}$ is also a model of this program, but not a minimal one. □

Lemma 4.1 *Let P be a restructuring or summarizing program and I be an input. If P admits a model with respect to I, then P admits a finite model including I.*

Proof. For a program P and an input I, let $adom(P,I)$ be the set of constants from $\mathcal{D}$ in P and I. Let *maxpoint* be the maximal number of occurrences of the constructor "$\cdot$" in any given *name* used in P and I. Then we define $Ref(P,I)$ to be the set of all *reference* expressions constructed from constants in $adom(P,I)$ using at-most *maxpoint* occurrences of "$\cdot$". As both P and I are finite, so is $Ref(P,I)$.
Now let P be a program and I an input such that P admits a model M with respect to I. Let $M' = \{A \mid A \in M \text{ and } ref(A) \in Ref(P,I)\}$. $Ref(P,I)$ is finite, so M' contains a finite number of *references*. And M' is consistent since M is consistent. So $card(M')$ can not be greater than the number of different references occurring in M'. Thus M' is finite.

Moreover, M' contains I, since $\forall A \in I, ref(A) \in Ref(P,I)$ and $I \subseteq M$. Whence M' is a finite interpretation including I.
We next show that M' is a model of P. Let $r = A \longleftarrow B_1, \ldots, B_n$ be a rule in P, and ν be a valuation such that $\nu(B_1), \ldots, \nu(B_n) \in M'$. Since $\nu(B_1), \ldots, \nu(B_n) \in M'$, each atomic name appearing in $\nu(B_1), \ldots, \nu(B_n)$ belongs to $adom(P,I)$. As r is range-restricted, each constant in $\nu(A)$ also belongs to $adom(P,I)$, and hence $ref(\nu(A)) \in Ref(P,I)$. Moreover, $\nu(B_1)$, $\ldots$, $\nu(B_n) \in M$ because $M' \subseteq M$ and since $M \models_I r$, we have $\nu(A) \in M$. As $ref(\nu(A)) \in Ref(P,I)$, $\nu(A) \in M'$. This means that M' is a model of P. Whence, there exists a finite model of P containing I. $\square$

Semantics of a restructuring or summarizing program For a restructuring or summarizing program P and an input I, the semantics of P on I is the unique minimal model of P with respect to I, if it exists. It is denoted $P(I)$.

Theorem 4.2 *Let P be a restructuring or summarizing program and I be an input. If P admits a model with respect to I, then $P(I)$ exists and is finite.*

Proof. Let P be a program and I an input such that P admits a model with respect to I. We define $\mathcal{X}$ to be the set of all models of P including I. $\mathcal{X}$ is non-empty, so we define $J = \cap\mathcal{X}$. By lemma 4.1 J is finite.
We first show that $J \models_I P$. Let $A \longleftarrow B_1, \ldots, B_n$ be a rule in P and ν a valuation such that $\nu(B_1), \ldots, \nu(B_n) \in J$. $\forall K \in \mathcal{X}$, since $J \subseteq K$, we have $\nu(B_1), \ldots, \nu(B_n) \in K$. Thus, $\nu(A) \in K$, because K is a model of P. This holds for all $K \in \mathcal{X}$. Hence, $\nu(A) \in J$. So J satisfies any rule in P.
Moreover, J includes I, because I is included in all $K \in \mathcal{X}$. By construction, J is the minimal model of P including I. Then $P(I) = J$. $\square$

Semantics of an OLAP program Let $Q = \langle R, S \rangle$ be an OLAP program, and I be an input. The semantics of Q on I is $S(R(I))$ if it exists. It is denoted $Q(I)$.

4.2.2 Fixpoint semantics

Immediate consequence operator Let P be a restructuring or summarizing program, I be an input, and J be an interpretation with respect to I. A ground atom A is an immediate consequence for J and P with respect to I if either $A \in J$, or there exists a rule $r \in P$ and there exists a valuation ν with $\nu(r) = A \longleftarrow B_1, \ldots, B_n$, such that $\forall i \in [1, \ldots, n], J \models_I B_i$.

For a restructuring or summarizing program P and an input I, we define the immediate consequence operator $T_{P,I}$ to be a partial mapping from interpretations of P with respect to I to interpretations of P with respect to I, such that, for an interpretation J:
$T_{P,I}(J) = \{A \mid A$ is an immediate consequence for J and P with respect to $I\}$ if this set is an interpretation with respect to I. Otherwise, $T_{P,I}(J)$ is undefined.

We have the following properties:

Lemma 4.3 (Monotonicity) *Let I be an input and J, K be two interpretations with respect to I such that $J \subseteq K$, and $T_{P,I}(K)$ is defined. Then $T_{P,I}(J)$ is defined and is included in $T_{P,I}(K)$.*

Proof. Let I be an input and J, K be two interpretations such that $J \subseteq K$, and $T_{P,I}(K)$ is defined. Let A be an immediate consequence for J and P. We show that $A \in T_{P,I}(K)$. Since A is an immediate consequence for J and P, at-least one of the two following cases applies:

- $A \in J$. Then $A \in K$, and thus $A \in T_{P,I}(K)$,
- there exists a rule $A \longleftarrow B_1, \ldots, B_n$ in P and a valuation ν such that $\forall B_i, i \in [1, \ldots, n], \nu(B_i) \in J$. As $J \subseteq K$, we have $\forall B_i, i \in [1, \ldots, n], \nu(B_i) \in K$, and thus $A \in T_{P,I}(K)$.

Since $T_{P,I}(K)$ is defined, $T_{P,I}(K)$ is an interpretation with respect to I. So the set of all immediate consequences for J and P is also an interpretation with respect to I. Then $T_{P,I}(J)$ is defined and we have $T_{P,I}(J) \subseteq T_{P,I}(K)$.
$\square$

Lemma 4.4 *J is a model of P with respect to an input I iff $T_{P,I}(J) \subseteq J$.*

Proof. Let I be an input, J be an interpretation with respect to I, and P be a program. We first prove $J \models_I P \implies T_{P,I}(J) \subseteq J$.
Suppose $J \models_I P$. We define $conseq(P, J) = \{A \mid A$ is an immediate consequence for J and P with respect to $I\}$ [1]. Given any A in $conseq(P, J)$, at-least one of the two cases applies:

- $A \in J$, by definition of the immediate consequence,
- there exists a rule $r \in P$ noted $A' \longleftarrow B_1, \ldots, B_n$ and a valuation ν such that $\forall B_i, i \in [1, \ldots, n], \nu(B_i) \in J$, and $\nu(A') = A$. Since J is a model of P, then $J \models_I r$, and we have $A \in J$.

1. Note that $conseq(P, J)$ is always defined, which is not the case for $T_{P,I}(J)$.

Thus $conseq(P, J) \subseteq J$. Moreover, as J is an interpretation with respect to I, $conseq(P, J)$ is also an interpretation with respect to I. Hence $T_{P,I}(J)$ is defined by $T_{P,I}(J) = conseq(P, J)$; and we have $T_{P,I}(J) \subseteq J$.

We now prove $T_{P,I}(J) \subseteq J \implies J \models P$.
Suppose $T_{P,I}(J) \subseteq J$. Given r any rule $A \longleftarrow B_1, \ldots, B_n$ in P, and any valuation ν, if $\nu(B_1), \ldots, \nu(B_n) \in J$, then $\nu(A) \in T_{P,I}(J)$. So $\nu(A) \in J$, because $T_{P,I}(J) \subseteq J$, and then $J \models_I r$.
Hence, $J \models P$. □

Lemma 4.5 *Each fixpoint of $T_{P,I}$ is a model for P.*

Proof. We recall that J is a fixpoint of an operator T iff $T(J) = J$.
Let J be a fixpoint of $T_{P,I}$. That is, for the program P, $T_{P,I}(J) = J$. By lemma 4.4, we have:

$$T_{P,I}(J) = J \implies J \models_I P.$$

□

Theorem 4.6 *Let P be a restructuring or summarizing program and I be an input such that $P(I)$ exists. Then $T_{P,I}$ has a unique minimal fixpoint, that equals $P(I)$.*

Proof. Let P be a program and I an input such that P admits a unique minimal model $P(I)$ with respect to I. We first show that $P(I)$ is a fixpoint of $T_{P,I}$. We show that $T_{P,I}(P(I)) = P(I)$ by showing that $T_{P,I}(P(I)) \subseteq P(I)$ and $P(I) \subseteq T_{P,I}(P(I))$.

- by lemma 4.4 $T_{P,I}(P(I)) \subseteq P(I)$ because $P(I)$ is a model of P;
- we have $T_{P,I}(P(I)) \subseteq P(I)$. Since $T_{P,I}(P(I))$ is defined, then by lemma 4.3 $T_{P,I}(T_{P,I}(P(I)))$ is also defined, and $T_{P,I}(T_{P,I}(P(I))) \subseteq T_{P,I}(P(I))$. Thus by lemma 4.4, $T_{P,I}(P(I))$ is also a model of P. As $P(I)$ is the minimal model of P, $P(I) \subseteq T_{P,I}(P(I))$.

Hence $P(I)$ is a fixpoint of $T_{P,I}$.
By lemma 4.5, each fixpoint of $T_{P,I}$ is a model of P. As each model of P contains $P(I)$, each fixpoint of P contains $P(I)$. This proves that $P(I)$ is the minimal fixpoint of $T_{P,I}$. □

Naive evaluation procedure Let P be a program and I an input, then let

- $T_P^0(I) = I$,
- $T_P^{n+1}(I) = T_{P,I}(T_P^n(I))$, if defined.

We have the following result:

Theorem 4.7 *Let P be a restructuring or summarizing program and I an input such that $P(I)$ exists. Then the sequence $T_P^0(I)$,$T_P^1(I)$, $T_P^2(I), \ldots$ reaches a fixpoint after a finite number N of steps, with $T_P^N(I) = P(I)$.*

Proof. Let P be a program and I an input such that $P(I)$ exists. We first show by induction on i that $\forall i \geq 0, T_P^i(I)$ is defined and $T_P^i(I) \subseteq P(I)$.

- For $i = 0, T_P^i(I) = I$ is defined and $I \subseteq P(I)$.
- Suppose for $i = n, T_P^n(I)$ is defined and $T_P^n(I) \subseteq P(I)$. Since $T_{P,I}(P(I))$ is defined (by theorem 4.6) and $T_P^n(I) \subseteq P(I)$, by lemma 4.3 $T_P^{n+1}(I)$ is also defined and $T_P^{n+1}(I) \subseteq T_{P,I}(P(I))$. And since $P(I) = T_{P,I}(P(I))$ (by theorem 4.6), we have $T_P^{n+1}(I) \subseteq P(I)$.

Then, $\forall i \geq 0, T_P^i(I)$ is defined and is included in $P(I)$. Moreover, since T_P is monotone (lemma 4.3) and $P(I)$ is finite (theorem 4.2), the sequence $\{T_P^i(I)\}_i$ reaches a fixpoint after a finite number N of steps.
$T_P^N(I)$ is a fixpoint of T_P and is included in $P(I)$. As $P(I)$ is the minimal fixpoint of $T_{P,I}$ (theorem 4.6), we have $T_P^N(I) = P(I)$. □

Let $Q = \langle R, S \rangle$ be an OLAP program and I be an input. Theorem 4.7 defines a constructive semantics for $R(I)$, and then for $S(R(I))$ (i.e., $Q(I)$). This provides a straightforward naive evaluation procedure for OLAP programs.

5 Expressive power

In this section, we give the principle of the translation of programs written in our language into Datalog programs. We show that a version of our language, without the grouping and summarizing constructs but extended with a restricted form of negation, is sufficient to characterize the class of programs on ordered multidimensional databases computable in polynomial time. The result relies on the fact that the expressive power of this language is equivalent to that of *semipositive Datalog*$^\neg$ on ordered databases [40].

In this section, we only consider ordered databases. These databases correspond to programs for which the constants involved in the input and in the rules are totally ordered. This order can be accessed using 3 specific predicates: $min/1$, $max/1$ and $succ/2$. $min(x)$ holds iff x is the smallest constant, $max(x)$ holds iff x is the largest constant, and $succ(x, y)$ holds iff y is the immediate successor of x with respect to the total ordering.

We use a version of the language, where rules are constructed according to the following syntactical expressions:

$$
\begin{array}{lcl}
atomicName & := & c \mid v \\
name & := & atomicName \mid name \cdot name \\
contents & := & \langle name, \ldots, name \rangle \\
reference & := & name(name, \ldots, name) \\
atom & := & reference : value \\
literal & := & atom \mid min(atomicName) \\
 & & \mid max(atomicName) \\
 & & \mid succ(atomicName, atomicName) \\
 & & \mid \neg literal \\
body & := & literal, \ldots, literal \mid \epsilon \\
head & := & atom \\
rule & := & head \longleftarrow body
\end{array}
$$

where c belongs to a countably infinite and set of constants, and v belongs to a disjoint countably infinite set of variables.

The use of aggregate functions would have lead us to take into account the computational complexity of these functions. However, they are of no use to characterize the class of queries computable in polynomial time on ordered databases. For the sake of simplicity, we simply drop the grouping and summarizing possibilities. So program written in this version are not divided into restructuring and summarizing parts.

For this language, the formal definitions of section 4 can be adapted easily, the most salient differences are the following:

A range restricted rule is a rule $r = A \longleftarrow B_1, \ldots, B_n$ where:

- $var(A) \subseteq var(\{B_1, \ldots, B_n\})$,
- let $\mathcal{N}eg$ be the set of negative literals occurring in the body of r, and $\mathcal{P}os$ be the set of positive literals occurring in the body of r. Then $var(\mathcal{N}eg) \subseteq var(\mathcal{P}os)$.

A program P is set of range restricted rules.

Let P be a program and $r \in P$. Let I be an input and J be an interpretation with respect to I. J satisfies r with respect to I, denoted $J \models_I r$, iff for each valuation ν, with $\nu(r) = A \longleftarrow B_1, \ldots, B_n$ we have:

1. $A \in J$, or
2. $\exists B_i, i \in [1, \ldots, n]$, B_i is a positive literal, and $B_i \notin J$, or

3. $\exists B_i, i \in [1, \ldots, n]$, B_i is a negative literal of the form $\neg C$ and $C \in I$.

Let P be a program, I be an input and J be an interpretation with respect to I. A ground atom A is an immediate consequence for J and P with respect to I if either $A \in J$, or there exists a rule $r \in P$ and there exists a valuation ν with $\nu(r) = A \longleftarrow B_1, \ldots, B_n$, and:

- $\forall i \in [1, \ldots, n]$, if B_i is a negative literal of the form $\neg C$, then $C \notin I$, and
- $\forall i \in [1, \ldots, n]$, if B_i is a positive literal, then $B_i \in J$.

The theorems of section 4 can be adapted easily.

Here again, the semantics of a program in this version is given with respect to a set of ground atoms called the input, that represents the extensional part of the database. Intuitively, negative literals are given truth values only with respect to the input. This yields a form of negation in the spirit of *semipositive Datalog*$^\neg$ [4] (the formal syntax and semantics for this version of our language, as well as examples of queries, can be found in [27]).

We have the following results:

Proposition 5.1 *On ordered databases semipositive Datalog*$^\neg$ *can be simulated within our language.*

Crux Each *semipositive Datalog*$^\neg$ atom $Q(X_1, \ldots, X_n)$ can be represented in our language by an atom $Q(X_1, \ldots, X_n) : \langle\rangle$. The representation of a *semipositive Datalog*$^\neg$ program and input is then straightforward. $min/1$, $max/1$ and $succ/2$ need no particular encoding.

Proposition 5.2 *On ordered databases, our language can be simulated within semipositive Datalog*$^\neg$ *.*

We illustrate the encoding on the following example. Consider the program: $P = \{T \cdot X(Y) : \langle Z\rangle \longleftarrow T(X, Y) : \langle Z\rangle, \neg X(Z) : \langle Y\rangle\}$, and the input $I = \{a(b, c) : \langle b\rangle\}$.

The atoms in I or in P are encoded using two special predicate names: *edb* and *idb*, used respectively for the extensional and intensional part of the database, and three additional new constants not occurring in P or I: *nest*, *memb* and *cont*. For example, the head of the rule $T \cdot X(Y) : \langle Z\rangle$ is encoded as $idb(T, nest, X, memb, Y, cont, Z)$. The constant *nest* indicates that the variable X is nested with the variable T to obtain the table name, *memb* indicates that Y is a member, and *cont* indicates that Z is the cell contents. Following this principle I is encoded as:

$$edb(a, memb, b, memb, c, cont, b).$$

The encoding of rules reflects the definition of the immediate consequence operator. In this example the rule is encoded:

$$idb(T, nest, X, memb, Y, cont, Z) \longleftarrow edb(T, memb, X, memb, Y, cont, Z),\\ \neg edb(X, memb, Z, cont, Y).$$

In fact the positive literals in the body of the original rule may hold either in the idb or in the edb part. Thus we also need the following rule:

$$idb(T, nest, X, memb, Y, cont, Z) \longleftarrow idb(T, memb, X, memb, Y, cont, Z),\\ \neg edb(X, memb, Z, cont, Y).$$

In our language some programs violate cell monovaluation and have no model. They are encoded as *semipositive Datalog*$^\neg$ programs having a minimal model in which the fact *panic* holds. To detect cell multivaluation in our example, we need to add the following rule:

$$panic \longleftarrow idb(T, memb, X, memb, Y, cont, Z),\\ idb(T, memb, X, memb, Y, cont, W),\\ different(W, Z).$$

where $different$ is defined by:

$$lesserThan(X, Y) \longleftarrow succ(X, Y).$$
$$lesserThan(X, Y) \longleftarrow lesserThan(X, Z), succ(Z, Y).$$
$$different(X, Y) \longleftarrow lesserThan(X, Y).$$
$$different(X, Y) \longleftarrow lesserThan(Y, X).$$

We can now state the following result:

Theorem 5.3 *Our language expresses exactly the class of programs computable in polynomial time on ordered databases.*

Proof. By the two previous propositions, and since *semipositive Datalog*$^\neg$ expresses exactly the class of programs computable in polynomial time on ordered databases (with $min/1$, $max/1$ and $succ/2$) [40]. □

Remark D Although the results presented in this section show that programs in some version of our language can be encoded as *semipositive Datalog*$^\neg$ programs, this encoding is neither natural nor suggestive of specific evaluation technics. Our framework gives a direct syntax and semantics for multidimensional table manipulation in deductive databases, and as in the case of Hilog (that can be encoded in Prolog [16]) this direct syntactical and semantical representation of specific concepts provides an easily understandable formulation level and a better basis for evaluation. □

6 Applications of the language

In this section, we illustrate the applicability of our language to specify in a concise yet readable way various complex manipulations arising in OLAP applications. We consider that the language incorporates the following standard logic programming extensions: Built-in predicates like $\geq$ or $\times$, the assignment instruction *is* and anonymous variables noted "_". We also allow classical Datalog atoms to occur in the body of the rules, with their usual semantics.

6.1 Description of the database

The cubes we use in this section are the following: The cube *ventes*, depicted in figure II.5 page 21, that contains yearly part sales in 4 regions, the cube *ventes jours* that contains daily part sales in various cities, the cube *ventes villes*, depicted in figure II.16 page 29, that contains yearly part sales in the different cities, the cube *ventes*96 *villes*, depicted in figure III.1 page 72, which is the restriction of the previous cube to year 1996, and the cube *prix* that contains yearly prices for the different parts (the restriction of the cube *prix* to year 1996 is depicted in figure II.20 page 32).

The hierarchies used are the ones depicted in figure II.12 page 27, except for the time hierarchy. In this hierarchy, the days constitute the most detailed level. Days can be grouped to form months or weeks, and both months and weeks can be grouped to form years. The years can be grouped to form the whole period, called *temps*.

6.2 Typical manipulations

We now express on this database typical non-trivial queries (e.g., sales estimates or complex roll-ups), including queries 2.3, 2.4, 2.6 and 2.7 of chapter II. The expression of queries 2.2, 2.1 and 2.5 is not given since it can be derived straightforwardly from the examples presented in this section.

Restructuring When querying the database, the syntax of our language allows to specify the structure of the resulting cube. For instance, query 2.4 aims at selecting *vis* and *clous* sales, and representing these sales in two tables (whether the sales are greater than 30 or not). In the resulting tables, we want the years as row names, and the regions nested with the parts as column names. This query can be expressed with the following program:

$sel(vis) \longleftarrow .$

$sel(clous) \longleftarrow .$

$$\begin{aligned} ventesInf30(Y, R \cdot P) : \langle S \rangle \longleftarrow\ & ventes(P, Y, R) : \langle S \rangle, \\ & S \leq 30, \\ & sel(P). \end{aligned}$$

$$\begin{aligned} ventesSup30(Y, R \cdot P) : \langle S \rangle \longleftarrow\ & ventes(P, Y, R) : \langle S \rangle, \\ & S > 30, \\ & sel(P). \end{aligned}$$

where the intermediate relation *sel* is used for the selection condition.

Calculated measures/members Expressing query 2.3 (multiplying the sales results with the corresponding part prices and representing the result together with the sales results) can be done with the following rule:

$$\begin{aligned} ventesPrix(P, Y, R) : \langle S, C \rangle \longleftarrow\ & ventes(P, Y, R) : \langle S \rangle, \\ & prix(P, Y) : \langle Pr \rangle, \\ & C\ is\ S \times Pr. \end{aligned}$$

Suppose we want to express the query 2.6 (computing for every pair of consecutive years the sales average). The same principle as above can be used to compute the average, and a nesting operation can be used to create new members (from the year names), as illustrated by the following rule:

$$\begin{aligned} résultat(Y \cdot Y', moyenne) : \langle M \rangle \longleftarrow\ & ventes(P, Y, R) : \langle S1 \rangle, \\ & Y'\ is\ Y + 1, \\ & ventes(P, Y', R) : \langle S2 \rangle, \\ & M\ is\ (S1 + S2)/2. \end{aligned}$$

Recursion can be used to express query 2.7 (computing the sales estimates up to year 2000 with a 10 percent increase) in a concise way:

$$\begin{aligned} ventes(P, Y', R) : \langle S' \rangle \longleftarrow\ & ventes(P, Y, R) : \langle S \rangle, \\ & Y'\ is\ Y + 1, \\ & S'\ is\ S + S/10, \\ & Y' \leq 2000, \\ & Y \geq 1996. \end{aligned}$$

Drill-down Navigation towards more detailed data can also be specified by rules. Consider the cube *ventes* extended with the cells specified by the previous rule. Let us consider the sales estimates by region for year 2000. We want to extract the corresponding estimates for the cities, with the hypothesis that the region estimates is equally divided for each city belonging to the region. This can be expressed by first computing the number of cities per region (using the cube *ventes*96 *villes*) for each part sold in 1996, with the rule:

$$
\begin{aligned}
nbVilles96(P,R):\langle N\rangle \longleftarrow\ & N = count(ventes96Villes(P,R)),\\
& in(R, lieu),\\
& in(P, produits).
\end{aligned}
$$

and next by selecting year 2000 in cube *ventes* and drilling down to cities, with the rule:

$$
\begin{aligned}
estim2000Villes(P,C):\langle S'\rangle \longleftarrow\ & ventes(P,2000,R):\langle S\rangle,\\
& in(C,R),\\
& nbVilles96(P,R):\langle N\rangle,\\
& S'\ is\ S/N.
\end{aligned}
$$

Ad-hoc grouping Consider a new cube named *responsables*, in which a cell of reference $responsables(P,R)$ contains the name of the person responsible for part P in region R (a person may be responsible for several parts in several regions). This cube is depicted in figure III.3.

responsables	est	ouest	sud	nord
vis	john	john	bob	mike
clous	bob	mike	bob	mike
écrous	john	john	bob	john

FIG. III.3 - *The cube* responsables

For each person, we want to sum up the 1996 sales concerning the parts he is responsible for (according to the parts and the regions). The corresponding groupings can not be computed using the hierarchies described in section 6.1. So we need to specify an ad-hoc grouping hierarchy.

First, we define how parts are grouped by responsible names:

$$in(P,N) \longleftarrow responsables(P,R):\langle N\rangle.$$

And we specify the way regions may be grouped by responsible name:

$$in(R,N) \longleftarrow responsables(P,R):\langle N\rangle.$$

Then we simply summarize the sales for each responsible names appearing in the cube *responsable* (the resulting cube is depicted in figure III.4):

$$
\begin{aligned}
respR\acute{e}sult96(total,N):\langle S\rangle \longleftarrow\ & S = sum(ventes(N,1996,N)),\\
& responsables(\_,\_):\langle N\rangle.
\end{aligned}
$$

respRésult96	john	bob	mike
total	160	140	170

FIG. III.4 - *The cube* respRésult96

Roll-up to all possible levels A single rule can specify the summaries of the daily part sales in cube *ventes jours* at all grouping levels. Moreover, the resulting summaries can be placed within the cube *ventes jour* itself. Suppose we want to include into the cube *ventes jours* for each part P, the summaries of the sales in *lyon* at three different levels: at the month level, at the year level and at the *temps* level (grouping all years). This can be done with the single rule:

$$\begin{aligned} ventesJours(P, T, lyon) : \langle S \rangle \longleftarrow\ & S = sum(ventesJours(P, T, lyon)),\\ & in(P, product),\\ & in(\_, T). \end{aligned}$$

Each value of T according to each instantiation of $in(\_, T)$ gives rise to a different grouping. Hence, summaries are specified at all grouping levels for the time hierarchy, except for the lowest level (i.e., the days level). It should be noticed that for any valuation ν such that there is no most detailled information for $\nu(ventesJours(P, T, lyon))$ then $\nu(S = sum(ventesJours(P, T, lyon)))$ is not satisfiable (see the paragraph satisfaction of page 77 for formal details). So the previous rule will not specify summaries with T bound to *est*, *ouest*, *sud*, *nord*, *lieu* or *produit*.

Cube operator The cube operator proposed by Gray & al. [22] is a n-dimensional generalization of the SQL group-by operator. Intuitively, the query:

select ...
from ...
cube by a, b, c

computes the group-by aggregates for each possible subset of $\{a, b, c\}$. This operator has in turn been generalized by Shukla & al [47] towards cubes with aggregation hierarchies. In this case, the cube operator computes the group by aggregates for each possible subset of the given set of attributes at each possible *level* of aggregation in the hierarchies. This principle is illustrated in figure III.6 for the 2-dimensional cube c containing daily sales by cities (figure III.5).

c	jour_1	jour_2	$\ldots$	jour_q
ville_1	v_{11}	v_{12}	$\ldots$	v_{1q}
ville_2	v_{21}	v_{22}	$\ldots$	v_{2q}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
ville_p	v_{p1}	v_{p2}	$\ldots$	v_{pq}

FIG. III.5 - *The cube* c

In our language, this generalized cube operator applied to cube c can simply be specified by:

c	jour_1	...	mois_1	...	jour_q	mois_n	année_1	...
ville_1	v_{11}	...	$\sum$	...	v_{1q}	$\sum$	$\sum$	...
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	...
région_1	$\sum$	...	$\sum$	...	$\sum$	$\sum$	$\sum$	...
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	...
ville_p	v_{p1}	...	$\sum$	...	v_{pq}	$\sum$	$\sum$	...
région_m	$\sum$	...	$\sum$	...	$\sum$	$\sum$	$\sum$	...
lieu	$\sum$	...	$\sum$	...	$\sum$	$\sum$	$\sum$	...

FIG. III.6 - *Result of the cube operator applied on* c

$$
\begin{aligned}
c(P,T,R):\langle S\rangle \longleftarrow\ & S = sum(c(P,T,R)),\\
& level(P),\\
& level(T),\\
& level(R).
\end{aligned}
$$

where *level* may be defined by the two following rules:

$$level(X) \longleftarrow in(X,Y).$$
$$level(Y) \longleftarrow in(X,Y).$$

7 Conclusion

We proposed an extension of datalog devoted to the specification of data manipulations in OLAP systems. We formally defined its model-theoretic semantics, and an equivalent fixpoint semantics. We described a version of the language that expresses exactly the class of programs computable in polynomial time on ordered databases. We illustrated on typical examples its capabilities of handling standard restructuring and summarizing operations on multidimensional cubes, and we showed its ability to express more complex treatments.

This work allows us to extend the table of figure II.39 page 63. It provides a simple framework for the integration of multidimensional data and rules, and allows a clear and concise specification of complex operations used in OLAP systems. Our proposition can serve as a basis for the reuse in these systems of the works developed for deductive databases (e.g., optimization methods, complex semantics). It can also be a starting point to study some open problems of the OLAP field (e.g., semantics of recursion through hierarchies).

Figure III.7 summarize the main interests of our framework. The prominent features are:

- It can be used for describing precisely the structure of a query result. Therefore, the restructuring elementary operations can be specified intuitively;

– It allows both static and dynamic hierarchy definitions without the need of an external language, and

– It remains close to the standard Datalog framework.

	Gyssens & al. [24]	Agrawal & al. [6, 7]	Li & Wang [35]	Cabibbo & Torlone [12, 13]	Gyssens & Laksh. [23]	Our proposal
type	algebra	algebra	calculus	algebra calculus	algebra	rule-based
member measure / symetry	yes	yes	no	yes	yes	yes
direct restructuring	rotate split switch	pull push	nest	pull push rotate	nest pull push	nest split rotate pull push
direct representation of hierarchies	no	no	yes	yes	no	yes
dynamic hierarchies definition	no	no	yes	no	no	yes
correspond. with the relational model	no	yes	yes	no	yes	yes

FIG. III.7 - *Comparison with related works*

This chapter illustrated the use of our language in the (general) context of OLAP systems. The following chapter shows that this language can be used as well in the context of spreadsheet programs to improve their global quality.

Une application directe du cadre proposé dans le chapitre précédent est la spécification de requêtes dans les systèmes OLAP. Nous avons choisi de présenter dans ce mémoire une application plus originale de ce cadre. Le chapitre qui suit décrit l'utilisation du cadre formel pour la spécification et la génération de feuilles de calcul dans un tableur. Il est basé sur le matériel de [8], article en cours de révision pour la revue TSI, dont une partie à été publiée dans [10].

Chapitre IV

Application

1 Introduction

Les logiciels appelés tableurs sont apparus au début des années 80 sur les premiers micro-ordinateurs. Ils apportaient une réponse originale à de nombreux problèmes de calculs et, via la métaphore de la *feuille de calcul*, proposaient une interface particulièrement conviviale. Ils ont rencontré très tôt un énorme succès, notamment auprès des utilisateurs finaux qui voyaient là un moyen rapide de spécifier certains problèmes tout en obtenant immédiatement leurs résolutions informatiques [29].

Aujourd'hui, l'ubiquité des tableurs est flagrante. Ils sont toujours utilisés pour le prototypage rapide de calculs simples mais servent également de support à de véritables applications pour l'aide à la décision dans des domaines complexes. Ils apparaissent actuellement comme étant des clients privilégiés dans les nouvelles architectures des systèmes dédiés à l'analyse des données [18]. La qualité globale de ce type de systèmes d'information dépend alors pour partie de celle des applications développées avec des tableurs.

1.1 Tableurs et qualité

De fait, il y a une dimension d'ingénierie peu compatible avec les approches empiriques si naturelles aux utilisateurs de la première heure. Les applications à base de tableurs doivent être traitées avec le même soin que les autres logiciels : il faut disposer de méthodes pour assister les développeurs sur tout le cycle de vie des feuilles de calcul (spécification, conception, implantation, test et maintenances) et donc maîtriser a priori des critères de qualité comme par exemple la correction mais aussi la lisibilité ou l'évolutivité.

Ce problème de la qualité des feuilles de calcul est peu abordé dans la littérature, mais il suffit d'avoir des discussions informelles avec des informaticiens pour que de multiples anecdotes le fassent surgir. Ainsi, comme universitaires, nous sommes habitués à détecter incorrections ou inconsistances dans des feuilles de calcul diverses, qu'elles soient à finalité comptable ou plutôt dédiées à la scolarité, qu'elles soient le résultat de développements personnels ou collectifs. L'impact économique de ces défauts est déjà énorme et on le sait en forte croissance. Ainsi, dans [20], les auteurs déclarent que 25 % des modèles extraits d'une dizaine d'organisations comportaient des erreurs logiques.

Parmi les quelques travaux publiés sur la correction des feuilles de calcul, des études expérimentales ont mis en évidence la qualité médiocre de celles-ci à travers l'analyse des erreurs qu'elles contiennent. Ainsi, [11] est une étude expérimentale avec une analyse qualitative et quantitative des erreurs produites par des utilisateurs confirmés (1 à 5 années d'expérience). Ceux-ci, au nombre de 9, avaient été soumis à plusieurs problèmes, qu'ils avaient jugés plus simples que ceux qu'ils avaient habituellement à résoudre. Il leur était demandé de réaliser 3 modèles à l'aide du tableur Lotus 1-2-3, à partir d'énoncés d'une demi-page chacun, en mettant explicitement l'accent sur la qualité et la précision de la solution, non sur la rapidité d'exécution. Chaque problème fut résolu en une moyenne de 41 minutes, les participants déclarant avoir confiance en leurs solutions. Les résultats furent les suivants : 40 % des modèles développés produisaient des résultats faux, et, en intégrant les omissions (résultats attendus absents), on parvenait à 63 % de modèles insatisfaisants. De plus, il n'a pas été observé de corrélation entre la qualité des modèles et le degré de confiance annoncé par les participants. Selon [29], cette étude confirme qu'il faut situer la difficulté principale non pas dans la maîtrise de l'outil, mais plutôt dans *l'élaboration des principes de l'application à développer*. Le lecteur trouvera dans [39] une synthèse sur les études expérimentales dans le domaine de la qualité des feuilles de calcul, ainsi qu'une taxinomie des erreurs rencontrées.

1.2 Tableurs et niveau logique

Ces études témoignent de l'absence de méthodes pour la conception des feuilles de calcul. Notre travail apporte des éléments pour asseoir un tel cadre, en proposant d'abord une analyse des spécificités de ce type de développement, et surtout des éléments de solution pour mieux maîtriser la fiabilité a priori des feuilles de calcul durant tout le cycle de vie. Pour améliorer l'état de l'art, nous pensons qu'il faut d'abord travailler à des niveaux d'abstraction plus élevés. Comme dans [30, 29], notre idée est de considérer des concepts qui ont fait leurs preuves dans le domaine des bases de données. Nous considérons principalement deux axes de travail :

- la recherche d'un niveau logique ;
- la proposition d'un Langage de Manipulation de Données (LMD) pour feuilles de calculs.

Le premier point est assez consensuel et il existe des propositions [30, 29]. Cependant, nous verrons que celles-ci n'intègrent pas à un niveau logique la nature multidimensionnelle des données. Nous nous en inspirons néanmoins pour nous concentrer sur les aspects liés aux manipulations de données, au sens employé dans le domaine des bases de données, telles que l'interrogation et la restructuration des informations contenues dans une feuille de calcul.

Nous nous basons sur le travail décrit dans le chapitre III et dérivons un LMD textuel, que nous baptisons DaTabLog, pour définir formellement la manipulation de données dans un tableur. La syntaxe de ce langage est décrite dans la section 3. Celui-ci dispose d'une sémantique déclarative intuitive, ce qui est important pour faciliter l'écriture mais aussi la lisibilité des définitions, et d'une sémantique opérationnelle naïve utilisable dans une phase de prototypage. L'intérêt du langage va au-delà des étapes de définition et de prototypage, et nous montrons comment automatiser en partie la génération d'une feuille de calcul à partir de ses spécifications écrites en DaTabLog, en réduisant ainsi de façon importante les causes d'erreurs possibles.

Nous supposons dans notre présentation que le lecteur est familier du paradigme feuille de calcul, et qu'il a déjà utilisé un des tableurs du marché comme par exemple Excel [1]. La suite de ce chapitre est organisée de la façon suivante. Dans la section 2, nous établissons un diagnostic pour expliquer la mauvaise qualité des applications à base de tableurs et présentons les solutions actuelles. La section 3 présente brièvement DaTabLog et introduit une technique permettant d'obtenir le squelette d'une feuille de calcul à partir de spécifications DaTabLog, et ainsi d'automatiser pour partie le développement de ces feuilles.

2 Un diagnostic

Les travaux abordant le problème de la qualité des applications à base de tableurs sont peu nombreux. Néanmoins, les plus significatifs ([30, 29, 44]) s'accordent pour situer l'origine de la plupart des défauts des développements à base de tableurs dans *l'amalgame fait entre d'une part, la définition des relations entre les données et d'autre part, la définition du placement de ces données dans la feuille de calcul.* Cette section précise cette position en donnant tout d'abord quelques définitions préliminaires, avant de l'argumenter à partir d'exemples d'erreurs typiques et de présenter les solutions proposées actuellement.

2.1 Tableurs et manipulations de données

La caractéristique essentielle des tableurs est de proposer à l'utilisateur une feuille de calcul i.e., un ensemble de *cases* organisées dans une grille à deux dimensions (e.g., dans Excel., les lignes sont nommées par les nombres 1, 2, 3 ... et les colonnes par les lettres A, B, C ...). On utilise les feuilles de calcul pour stocker soit des valeurs (e.g., des textes, des nombres ...) soit des formules (e.g., des expressions arithmétiques portant sur les contenus d'autres cases). Les tableurs proposent aussi des possibilités de gestion de données héritées des bases de données relationnelles (traitements de tables à une dimension), et permettent aussi des manipulations développées spécifiquement pour les besoins en analyse des données. Ils offrent ainsi une vision multidimensionnelle de ces données. Supposons que la figure IV.1 soit un extrait de feuille de calcul décrivant des ventes de pièces.

On peut considérer qu'il s'agit d'une table relationnelle : les noms qui figurent dans les colonnes de la première ligne définissent les noms des attributs et les lignes suivantes correspondent aux tuples, et dans ce cas appli-

pièces	régions	quantités	années
écrous	est	50	1996
écrous	ouest	60	1996
écrous	sud	40	1996
clous	est	70	1996
clous	nord	40	1996
clous	est	50	1995
clous	ouest	10	1995
clous	sud	10	1995
clous	nord	20	1995
écrous	est	70	1995
écrous	ouest	10	1995
écrous	sud	20	1995

FIG. IV.1 - *Enregistrements de ventes dans une feuille*

quer sur cette table certaines des opérations de l'algèbre relationnelle, telles que des sélections [1]. Mais cette feuille de calcul peut aussi être considérée comme étant un ensemble de points dans un espace à quatre dimensions, chaque point étant défini par ses coordonnées selon les dimensions *pièces, régions, quantité* et *années*. Dans Excel, la prise en compte du caractère multidimensionnel de ces données apparaît explicitement lors de la construction de *Tableaux Croisés Dynamiques* (TCD). Un TCD permet l'affichage de diverses présentations synthétiques, avec par exemple des imbrications dans la désignation des lignes et des colonnes. Ainsi, à partir des données de la figure IV.1, nous pouvons obtenir le TCD de la figure IV.2 où les dimensions *pièces* et *régions* sont imbriquées puis "croisées" avec la dimension *années*. Pour les détails de telles constructions, nous renvoyons le lecteur à la documentation du logiciel Excel [1].

<table>
<tr><td colspan="2">ventes</td><td>1996</td><td>1995</td></tr>
<tr><td rowspan="3">écrous</td><td>est</td><td>50</td><td>70</td></tr>
<tr><td>ouest</td><td>60</td><td>10</td></tr>
<tr><td>sud</td><td>40</td><td>20</td></tr>
<tr><td rowspan="4">clous</td><td>est</td><td>70</td><td>50</td></tr>
<tr><td>ouest</td><td></td><td>10</td></tr>
<tr><td>nord</td><td>40</td><td></td></tr>
<tr><td>sud</td><td></td><td>10</td></tr>
</table>

FIG. IV.2 - *Une synthèse des ventes*

En fait, les fonctionnalités pour la manipulation de données dans les feuilles de calcul sont nombreuses. Mais si on se réfère aux concepts de modèle logique et de modèle physique utilisés dans le domaine des bases de données, ces manipulations ne considèrent que le niveau physique de la feuille de calcul, c'est à dire la grille vue comme support de stockage des données. Il n'y a pas de réelle description de manipulations à un niveau logique plus abstrait. Nous illustrons maintenant comment ces manipulations à un niveau d'abstraction insuffisant constituent une source d'erreurs importante.

2.2 Quelques erreurs typiques

	A	B	C	D	E	F	G	H
1	code	nom	gr	type	exam	td	tp	moy
2	1A127	aaa	1	n	10	10	10	10
3	1A452	bbb	2	n	11	7	15	11
4	1A371	ccc	1	r	8	8	14	9.5
5	1A214	ddd	2	n		10	13	11.5
6	1A509	eee	2	n	15	11	12	13.3
...	...	...	...	...	...	...	...	...

FIG. IV.3 - *Un extrait d'une feuille de calcul*

La figure IV.3 montre un extrait d'une feuille de calcul utilisée pour une gestion de scolarité. La seule vue d'une telle feuille ne suffit pas pour comprendre et le problème et sa résolution. Nous sommes pourtant dans un cas extrêmement simple mais le principe de calcul sous-jacent n'y est pas explicite. Précisons les données du problème : des étudiants sont identifiés par un *code*, ils ont un *nom*, sont répartis en différents groupes *gr* et sont soumis à diverses évaluations au cours d'un semestre. Pour chaque étudiant, on dispose également de son *type* (redoublant ou non selon la présence de *r* ou *n*) et des évaluations obtenues aux différentes épreuves (examen semestriel *exam*, note de travaux dirigés *td* et note de travaux pratiques *tp*). Pour l'instant, le seul résultat qui nous intéresse est la moyenne calculée en appliquant les coefficients 2 pour la note d'examen et 1 pour les autres. Enfin, une règle de scolarité stipule qu'une absence excusée à l'examen conduit à une absence de note (cas de *ddd*) et ceci sans aucune pénalité pour le calcul de la moyenne.

De fait, la logique et la documentation d'une feuille de calcul sont souvent enfouies dans les cases et les formules et sont difficilement accessibles aux utilisateurs. En tout état de cause, seule l'inspection de toutes les cases permet de comprendre la nature des traitements et cette granularité est préjudiciable à la compréhension des abstractions du domaine. Cette difficulté se rencontre a priori, lorsque la conception se fait directement au niveau du support d'implantation, mais aussi a posteriori, lors des phases de validation et de maintenance.

Par exemple à ce niveau, il est réaliste d'imaginer que l'utilisateur a fourni une formule spécifique pour le calcul de la moyenne de l'étudiant *ddd* ayant été absent et excusé. Cette règle est alors explicitement présente mais dans la seule case référencée $H5$ qui contiendra la formule $(F5+G5)/2$. Les autres cases de cette colonne contiennent alors la formule du cas général. Ainsi, $H2$ contient $(2 \times E2 + F2 + G2)/4$ pour calculer la moyenne de l'étudiant *aaa*. Ce type de "verrue" a bien évidemment des répercussions importantes lors de la modification des feuilles de calcul et nous illustrons une des erreurs typiques que cela peut engendrer.

Supposons que notre utilisateur souhaite ajouter les résultats du second semestre pour lequel on retrouve les mêmes épreuves dotées des mêmes coefficients. Il peut lui sembler naturel d'effectuer sa saisie et d'utiliser les possibilités de recopie de formules pour réutiliser celles du premier semestre en tirant parti de la translation automatique des références de cases placées dans les formules. La feuille de calcul obtenue pourra ressembler à celle de la figure IV.4 (copie de la colonne H sur la colonne L avec translation des références de cases situées dans les formules de quatre colonnes sur la droite).

	...	E	F	G	H	I	J	K	L
1	...	exam	td	tp	moy	exam	td	tp	moy
2	...	10	10	10	10	13	14	14	13.5
3	...	11	7	15	11	9	12	16	11.5
4	...	8	8	14	8	11	16	13	12.8
5	...		10	13	11.5	9	14	13	**13.5**
6	...	15	11	12	13.3	14	13	15	14
...	...	...	...	...	...	...	...	...	...

FIG. IV.4 - *Prise en compte des deux semestres*

On vient d'introduire une incorrection puisque la case $L5$ contient 13.5 au lieu de 11.3. Le problème provient de la réutilisation d'une formule spécifique pour l'évaluation de la moyenne du premier semestre de l'étudiant *ddd* qui avait été absent lors de l'examen de ce premier semestre. Ainsi, $L5$ contient $(J5+K5)/2$ au lieu de $(2 \times I5 + J5 + K5)/4$.

Considérons un second exemple, dans lequel on désire coder un modèle de calcul légèrement plus compliqué que celui de la feuille représentée figure IV.3. On introduit dans cet exemple une notion de groupe de projet composé d'exactement trois étudiants et l'on ajoute pour cela une colonne *grP* dans laquelle on indique les numéros des groupes de projet des étudiants. On prend en compte deux nouvelles règles de gestion de la scolarité : la note de *tp* est la note obtenue en projet, et les redoublants sont dispensés de projet (leur moyenne étant calculée simplement avec la note de *td* et la note d'examen). La formule qui prend en compte cette dernière règle est d'abord saisie directement dans la case $I4$ sous la forme $(2\times F4+G4)/3)$ puis recopiée dans la case $I9$ par erreur (elle était destinée à la case suivante $I10$). On a désormais deux erreurs que seule une inspection systématique de tous les contenus va pouvoir révéler. En effet, $I9$ contient la formule $(2\times F9+G9)/3$ et la note de projet de cet étudiant n'est donc pas prise en compte. Par contre $I10$ contient la formule générale $(2\times F10+G10+H10)/4$ et va délivrer une moyenne à cet étudiant redoublant en comptant la note 0 pour le projet (compte tenu des hypothèses de valeur par défaut généralement appliquées dans les produits du marché).

	A	B	C	D	E	F	G	H	I
1	code	nom	gr	type	grP	exam	td	tp	moy
2	1A127	aaa	1	n	1	10	10	16	11.5
3	1A452	bbb	2	n	2	11	7	15	11
4	1A371	ccc	1	r		8	8		8
5	1A214	ddd	2	n	1		10	16	13
6	1A509	eee	2	n	3	15	11	8	12.3
7	1A410	fff	1	n	2	12	15	15	13.5
8	1A163	ggg	1	n	2	14	11	15	13.5
9	1A498	hhh	2	n	3	8	13	13	**9.7**
10	1A472	iii	2	r		9	16		**8.5**
11	1A341	jjj	1	n	1	12	17	16	14.3
12	1A769	kkk	2	n	3	17	12	13	14.8

FIG. IV.5 - *Introduction de la notion de projet*

Dans ces deux exemples typiques, les erreurs ont été introduites lors de manipulations de la feuille de calcul à un niveau que nous appellerons niveau physique. Nous utilisons ce terme par analogie avec le domaine des bases de données dans lequel il désigne le niveau où l'on tient compte du placement des données sur leur support de stockage. Ici il s'agit du niveau auquel on prend en compte le placement des données sur leur support de présentation (i.e., la feuille affichée). La cause des erreurs présentées réside dans la nécessité de gérer les formules case par case, en plaçant les formules correctes aux endroits adéquats. Cette source d'erreurs peut être supprimée en permettant de définir la feuille de calcul de façon plus globale, à un niveau plus abstrait. A un tel niveau, le sens et donc la correction d'une formule ne devront plus être liés à son placement dans la feuille. Toujours par analogie avec le domaine des bases de données, nous qualifierons un tel niveau de niveau logique.

Si la description d'une feuille de calcul directement à un niveau physique augmente les risques d'erreurs, elle limite aussi la qualité globale de l'application en rendant très coûteux le codage des dispositifs de contrôles. Regardons de nouveau la feuille de la figure IV.5. Si l'on considère que les étudiants d'un même groupe de projet ont la même note sauf dans des cas exceptionnels, il paraît souhaitable d'offrir au sein de la feuille une zone telle que celle de la figure IV.6 dans laquelle lors de l'utilisation il sera possible de saisir les notes des projets. En effet, en présentant les étudiants non redoublants groupés par projet, on permet de contrôler visuellement que les étudiants d'un même groupe ont (en général) une même note, et on limite ainsi les risques d'erreurs de saisie.

Comment décrire au niveau physique un tel dispositif? Ceci est difficile sur la feuille de la figure IV.5. On peut procéder à des manipulations sur les données (e.g., en réalisant des filtrages et des tris sur la feuille) ou même construire case par case la zone présentée figure IV.6, et enfin placer des formules dans les cases de la colonne *H* de la figure IV.5 pour reprendre les valeurs des notes de projets saisies. La longueur d'un tel processus est suffisante pour que l'on puisse être découragé de le mener à bien. Et de plus ces manipulations, à réaliser au niveau physique, sont elles aussi des sources importantes d'erreurs et influenceront donc de façon très défavorable les coûts de mise au point.

grP		note
1	aaa	16
	ddd	16
	jjj	16
2	bbb	15
	fff	15
	ggg	15
3	eee	8
	hhh	13
	kkk	13

FIG. IV.6 - *Saisie de notes de projet*

2.3 Vers un niveau logique?

A la recherche d'un niveau logique, [30] propose d'identifier dans une feuille de calcul d'une part un schéma S et des données D mais aussi des informations de présentation E et des informations d'implantation B. La partie S est un schéma, indépendant des choix d'implantation, dans lequel on spécifie l'organisation des données et les règles ou contraintes qui définissent les données calculées. [30] considère pour cela une extension naturelle du concept de schéma relationnel. La figure IV.7 fait apparaître un schéma S pour le modèle de calcul implanté dans la figure IV.3.

Relation étudiant		
code :	numérique	% soulignement de la clé % de la relation
nom :	texte	
gr :	numérique	
type :	texte	% n ou r
exam :	numérique + $\{\perp\}$	% nombre ou marque % de valeur non définie
td :	numérique	
tp :	numérique	
moy :	Si exam $\neq\perp$ alors $(2 \times exam + td + tp)/4$ sinon $(td + tp)/2$	

FIG. IV.7 - *Une représentation du schéma de la figure IV.3*

Une seule relation, appelée ici *étudiant*, suffit pour traiter ce problème. Pour les attributs qui correspondent à des valeurs fournies en entrée, on ne donne que leurs types (e.g., *td* est un *numérique*). Pour les attributs dérivés, on indique la formule de calcul. Le lecteur trouvera une présentation du langage de description de cette forme de schémas dans [30].

La partie D est la collection des données proprement dites. Elle consiste en un ensemble de tuples pour chaque relation définie dans le schéma. Dans les tuples une constante particulière notée *c* indique que la valeur de l'attribut est calculée. Nous donnons en figure IV.8 l'extension définie par D pour la relation *étudiant* dans le cas de la feuille de la figure IV.3.

code	nom	gr	type	exam	td	tp	moy
1A127	aaa	1	n	10	10	10	c
1A452	bbb	2	n	11	7	15	c
1A371	ccc	1	r	8	8	14	c
1A214	ddd	2	n		10	13	c
1A509	eee	2	n	15	11	12	c

FIG. IV.8 - *Données du modèle implanté dans la figure IV.3*

La partie E quant à elle, est l'ensemble des informations qui ne sont pas prises en compte dans S ni D et qui interviennent dans la présentation (e.g., les titres, les noms de lignes ou de colonnes, la documentation). Par exemple, pour la feuille de la figure IV.3, il s'agit seulement des en-têtes de colonnes.

Enfin, la partie B fixe l'implantation dans la feuille de calcul des informations définies dans S, D et E. B est donc simplement la définition d'une mise en correspondance des contenus des tuples de D et des informations de E avec des cases de la feuille de calcul.

Le découpage en quatre parties introduit par [30] permet d'avoir une description du schéma et des données à un niveau indépendant des choix de placement physique. Une telle forme de découpage logique a également été proposée par [29], qui introduit le terme *modèle de calcul* pour désigner la définition des données du problème, celle des résultats ainsi que celle des ensembles de règles de calcul qui vont permettre de calculer toutes les valeurs des résultats à partir des valeurs des données initiales. Ce modèle de calcul suppose un ordonnancement des calculs (selon un graphe de dépendance des grandeurs) pour que l'évaluation des différents calculs puisse être automatique à chaque modification d'une donnée. Ce travail étant conceptuellement très proche de celui de [30], nous renvoyons le lecteur à [29] pour sa présentation. L'intérêt essentiel de ces deux approches est d'autoriser une définition des formules de calcul à un niveau plus abstrait que les langages de codage utilisés dans les tableurs, et de permettre une séparation claire entre la structure d'une feuille de calcul et les données qu'elle contient.

Ces propositions considèrent le tableur comme un processeur qui évalue le modèle de calcul implanté dans une feuille de calcul. Elles reposent sur une représentation des données en terme de relations (au sens des bases de données relationnelles) et offrent des possibilités de manipulations de données limitées, basées sur les opérations de l'algèbre relationnelle (e.g., sélection, projection, jointure). Elle ne prennent pas en compte l'aspect multidimensionnel des données et donc la définition à un niveau logique de manipulations sur des données multidimensionnelles. Elles ne permettent donc pas de décrire des constructions importantes telles que celle du TCD de la figure IV.2 ou celle de la zone de saisie proposée figure IV.6.

3 Génération de feuilles de calcul

Dans cette section nous présentons une application concrète d'un langage issu de notre recherche décrite dans le chapitre III. Ce langage est baptisé DaTabLog, et nous montrons comment à partir de spécifications écrites en DaTabLog il est possible de générer de façon semi-automatique une feuille de calcul au niveau physique. L'intérêt immédiat d'une telle génération est de faire disparaître les causes d'erreurs importantes décrites dans la section 2. Cette technique a été implantée en SWI-Prolog [49] pour générer des feuilles de calcul Excel [1].

Nous commençons par décrire rapidement DaTabLog, c'est à dire l'adaptation du cadre présenté chapitre III au domaine des tableurs. Nous exposons ensuite le principe global de la génération, que nous présentons ensuite de façon détaillée sur un exemple simple extrait de celui de la section 2.2. Nous reprenons enfin l'exemple complet de la section 2.2 et montrons que notre approche permet de spécifier ce cas non trivial dans son ensemble.

3.1 Le langage DaTabLog

DaTabLog s'apparente au langage présenté dans le chapitre III, mais dépouillé des sous-buts d'agrégat par soucis de simplicité, et restreint à la manipulation de tables bidimensionnelles dont chaque cellule ne contient qu'une donnée. Il est en revanche étendu aux contraintes sur domaines concrets (e.g., équations linéaires sur les rationnels) et se base sur les expressions syntaxiques suivantes :

$$
\begin{array}{lll}
\textit{référence} & := & nom(nom, nom) \\
\textit{atome} & := & \textit{référence} : nom \mid \textit{référence} = \textit{valeur-concrète} \\
\textit{corps} & := & atome, corps \mid \epsilon \\
\textit{tête} & := & atome \\
\textit{règle} & := & \textit{tête} \longleftarrow \ \ corps /\!\!/ \, contrainte
\end{array}
$$

où les *noms* sont construits comme au chapitre III (avec possibilité d'imbrication), *valeur-concrète* identifie une valeur d'un domaine concret (e.g., un entier) ou une variable, et *contrainte* est une formule issue d'un langage de contraintes, qui respecte les conditions suivantes :

- les ensembles de constantes et variables utilisés sont infinis dénombrables et disjoints ;
- le langage est clos par substitution des constantes aux variables ;
- la satisfiabilité de la conjonction de contraintes dans ce langage est décidable ;
- le langage contient au moins une contrainte satisfiable notée *vrai*.

Afin de simplifier le processus, nous ne considérons ici qu'une forme restreinte de contrainte dans le corps des règles. Nous imposons que la partie contrainte d'un corps de règle soit exprimée sous la forme d'une (et une seule) affectation et de tests de sélection. Ainsi la contrainte $X < 20, X = Y + 1$ sera formulée par la conjonction $X < 20, X \ is \ Y + 1$. Le problème du passage des contraintes aux formes affectations et sélections est un aspect classique des traitements sur des règles sous contraintes (e.g., l'optimisation de programme [31]).

On remarque qu'une cellule peut contenir soit un nom, soit une valeur concrète, et que cette différence est répercutée syntaxiquement au niveau des atomes.

Les sémantiques déclarative et opérationnelle de DaTabLog se déduisent aisément de celles présentées chapitre III. Elles sont décrites formellement dans [25, 8].

3.2 Principe

Nous adoptons une vision simplifiée pour présenter un contexte clair d'utilisation de la génération que nous proposons, mais il convient de noter que la technique décrite dans cette section peut être appliquée dans des cycles de vies plus complexes (e.g., évolution de feuilles existantes, intégration avec une base de données [9]).

Nous séparons clairement les phases de développement et d'utilisation d'une feuille de calcul, et ce, qu'elles soient menées à bien par deux personnes (un développeur et un utilisateur) ou par une seule et même personne. Pour nous, la phase de développement inclut les étapes allant de la spécification à la production d'une feuille de calcul. La phase d'utilisation consiste quant à elle à ouvrir une feuille de calcul existante à partir d'un logiciel du type tableur, puis à entrer des valeurs (des textes ou des nombres) dans certaines cellules et enfin à consulter les valeurs calculées, obtenues dans d'autres cellules.

Ici, une feuille de calcul sera donc simplement composée d'une grille dont les cases sont soit vides, soit contiennent une constante ou une formule de calcul faisant référence à d'autres cases de la grille. Dans ce contexte nous considérons deux types d'entrées. D'une part les entrées que nous appelons *initiales*, qui sont les données connues au moment de la génération de la feuille, et d'autre part les entrées dites *différées* qui sont les données saisies lors de l'utilisation de la feuille.

Plus précisément, le scénario dans lequel nous nous plaçons est le suivant : en fin de conception nous disposons d'une description écrite en DaTabLog, et à partir des entrées initiales nous produisons une feuille de calcul prête à traiter les entrées différées qui seront fournies par l'utilisateur. En ce qui concerne la feuille de calcul cible, nous considérons que nous disposons uniquement des possibilités de descriptions communes aux logiciels de type tableur. Nous n'envisageons pas la production de macro-commandes ou de procédures associées à la grille car actuellement les langages pour l'écriture de ces programmes nous paraissent trop hétérogènes et sont spécifiques à chaque tableur.

Sous ces contraintes, la génération d'une feuille passe par une réponse aux questions suivantes :

- quelles sont les références des cellules qui seront à terme implantées dans la feuille?
- Quelles sont les cellules contenant des entrées différées?
- Quelles sont pour les autres cellules les constantes ou les formules qu'elles contiendront?

Le processus de génération que nous proposons comprend les 4 étapes suivantes :

1. compléter le programme DaTabLog décrivant le modèle de calcul par des règles spécifiant les cellules utilisées pour recevoir les entrées différées ;

2. à partir de ce nouveau programme et de la connaissance des entrées initiales, déterminer l'ensemble des références de cellules utiles. C'est à dire les références de cellules qui contiendront une entrée initiale, une formule ou une entrée différée ;

3. spécialiser le programme obtenu en 1 par rapport aux références déterminées en 2, afin notamment d'exhiber les formules liant les différentes cellules ;

4. implanter la feuille de calcul, à partir des résultats des étapes 2 et 3, et d'un schéma d'implantation fixant la correspondance entre les références de cellules et leurs positions physiques dans la grille.

Ce processus est clairement semi-automatique car il nécessite une intervention du développeur aux étapes 1 et 4. On notera qu'en fait, l'étape 4 peut aussi être réalisée automatiquement en utilisant un placement par défaut des cellules dans la grille, basé par exemple sur l'ordre lexicographique des noms composant les références de cellules.

3.3 Technique de génération

Le cas simple suivant s'inspire de l'exemple donné section 2.2. Il s'agit de produire une feuille de calcul permettant la saisie de notes de travaux dirigés et d'examen final et fournissant la moyenne de chaque étudiant. Cette moyenne est calculée en appliquant un coefficient 2 à leur note d'examen et un coefficient 1 à leur note de travaux dirigés. Pour chaque étudiant le code alphanumérique qui l'identifie ainsi que son nom sont des informations connues a priori, que l'utilisateur n'aura pas à saisir. Si l'on considère seulement 2 étudiants, le squelette du tableau *étudiant* à produire est de la forme décrite figure IV.9.

	A	B	C	D	E
1	étudiant	nom	exam	td	moy
2	1A127	aaa			
3	1A452	bbb			

FIG. IV.9 - *Le squelette de tableau à obtenir*

Nous tenons compte de la règle de scolarité qui stipule qu'une absence excusée à l'examen se traduit par une absence de note sans incidence sur la moyenne (qui pour notre exemple sera alors égale à la note de *td*). Nous prenons donc les conventions de saisie suivantes : une absence excusée se traduit par la présence de *abs* dans la case concernée, et une absence non excusée par la présence d'une note égale à zéro. Dans ces conditions, ces spécifications peuvent être formulées en DaTabLog par le programme P, constitué des deux règles suivantes :

R_1

$$\begin{aligned} \acute{e}tudiant(X, moy) = M \longleftarrow\ & \acute{e}tudiant(X, td) = N1, \\ & \acute{e}tudiant(X, exam) = N2, \\ & M\ is\ (N1 + 2 * N2)/3. \end{aligned}$$

R_2

$$\begin{aligned} \acute{e}tudiant(X, moy) = M \longleftarrow\ & \acute{e}tudiant(X, td) = N1, \\ & \acute{e}tudiant(X, exam) : abs, \\ & M\ is\ N1. \end{aligned}$$

3.3.1 Etape 1

La première étape du processus de génération nécessite l'intervention du développeur pour spécifier toutes les références de cellules qui seront remplies de manière différée. Dans notre exemple, ce sont les références des cellules contenant les notes de *td* et d'examen, alors que celles décrivant les identifiants et les noms des étudiants constituent les entrées initiales. Le développeur indique donc comment obtenir les références des cellules des entrées différées en utilisant éventuellement les entrées initiales. Pour notre exemple, cette spécification est réalisée par ajout au programme des règles suivantes, dans lesquelles une constante spéciale notée @ marque le contenu des cellules de l'entrée différée :

RD_1

$$\acute{e}tudiant(X, td) = @ \longleftarrow \acute{e}tudiant(X, nom) : N.$$

RD_2

$$\acute{e}tudiant(X, exam) = @ \longleftarrow \acute{e}tudiant(X, nom) : N.$$

RD_3

$$\acute{e}tudiant(X, exam) : @ \longleftarrow \acute{e}tudiant(X, nom) : N.$$

La lecture déclarative d'une règle comme RD_1 est : si il existe une cellule de référence $\acute{e}tudiant(X, nom)$ contenant un nom N alors il existe une cellule de référence $\acute{e}tudiant(X, td)$, mais dont la valeur sera fournie ultérieurement par l'utilisateur de la feuille de calcul. On remarque que le concepteur peut spécifier de façon explicite la nature des entrées différées : constantes concrètes (éléments du domaine du langage de contraintes) ou noms. Ainsi

RD_1 stipule que les notes de *td* sont des constantes concrètes, alors que RD_2 et RD_3 indiquent que l'information entrée pour un examen final peut être soit une donnée concrète (une note), soit un nom (la constante *abs*).

3.3.2 Etape 2

Lors de la deuxième étape, nous nous intéressons à la dénotation du programme P pour un ensemble d'entrées initiales I et un ensemble d'entrées différées D. Cette dénotation est le modèle minimal de P incluant I et D. D ne sera connu que lorsque les cellules le composant seront saisies par l'utilisateur de la feuille de calcul, mais il est déjà possible de déterminer une *approximation par excès* de cette dénotation. En effet, soit P' le programme P augmenté des règles fournies dans l'étape 1. Une telle approximation peut être obtenue en calculant le modèle minimal de P' incluant I, en tenant compte du fait que la constante @ représente des valeurs non connues de la manière suivante :

1. si @ apparaît en membre droit d'une expression *is*, alors ce membre se réécrit en @ (e.g., $M\ is\ (N1 + 2 \times @)/3$ se réécrit en $M\ is\ @$). Pour simplifier, nous supposons que le membre gauche d'une expression *is* est toujours une variable libre lors de l'évaluation ;

2. si @ apparaît dans une expression de test (e.g., $@ \geq 10$) alors ce test réussit ;

3. @ approxime n'importe quelle autre constante (e.g., si *étudiant*(*aaa*, *exam*) : @ existe alors *étudiant*(*aaa*, *exam*) : *abs* existe aussi).

Pour l'exemple, considérons les entrées initiales I suivantes :

$\mathsf{I} = \{\acute{e}tudiant(1A127, nom) : aaa, \acute{e}tudiant(1A452, nom) : bbb\}$

et le programme P' composé des règles R_1, R_2, RD_1, RD_2 et RD_3.

Le modèle minimal de P' incluant I est :

$\{\acute{e}tudiant(1A127, nom) : aaa,$
$\acute{e}tudiant(1A452, nom) : bbb,$
$\acute{e}tudiant(1A127, td) = @,$
$\acute{e}tudiant(1A452, td) = @,$
$\acute{e}tudiant(1A127, exam) : @,$
$\acute{e}tudiant(1A127, exam) = @,$
$\acute{e}tudiant(1A452, exam) : @,$
$\acute{e}tudiant(1A452, exam) = @,$
$\acute{e}tudiant(1A127, moy) = @,$
$\acute{e}tudiant(1A452, moy) = @\}$

Cet ensemble, appelons-le A, constitue une approximation du modèle minimal de P pour les entrées initiales I données et pour tout D décrit par P'.

En fait cette approximation n'est correcte que si les deux conditions suivantes sont remplies :

1. P, I et D forment un tout consistant, au sens du critère de consistance décrit dans le chapitre III (une référence est associée à un unique contenu) ;

2. toutes les références de cellules sont déterminées par P' et I. Ceci signifie que si les contenus des cellules peuvent dépendre de D, en revanche les références de cellules doivent toutes se construire à partir des informations connues au moment de la génération. En effet, lorsque la feuille de calcul aura été produite, il ne sera alors plus possible d'ajouter automatiquement de nouvelles cellules dont les références sont calculées à partir des entrées saisies par l'utilisateur. Vérifier que cette condition est satisfaite peut être réalisé automatiquement, en s'assurant qu'aucune référence de cellule apparaissant dans A ne contient la constante @ (i.e., @ n'est utilisée que pour représenter des contenus de cellules).

3.3.3 Etape 3

La troisième étape consiste à spécialiser le programme P par rapport à l'approximation A, afin d'obtenir des règles suffisamment instanciées pour dériver les formules qui seront placées dans la feuille de calcul. Chaque règle de P est remplacée par l'ensemble de ses instances closes valuées par les atomes de A. Comme A approxime le modèle minimal de P par excès, cette spécialisation est correcte.

Avant d'effectuer ce remplacement, nous différencions entre elles les constantes représentant des données non encore connues. En effet, lors des étapes 1 et 2 toutes les constantes inconnues sont traitées de la même manière et une seule constante @ est utilisée pour dénoter n'importe quel contenu inconnu de cellule. A présent, une approximation plus fine est nécessaire. Pour chaque cellule contenant une valeur non encore connue, la référence de cellule est associée à une constante @ qui lui est propre. Nous indiçons simplement la constante @ par un entier différent pour chaque référence de cellule différente. L'ensemble A de l'exemple devient alors:

$\{étudiant(1A127, nom) : aaa,$
$étudiant(1A452, nom) : bbb,$
$étudiant(1A127, td) = @_1,$
$étudiant(1A452, td) = @_2,$
$étudiant(1A127, exam) : @_3,$
$étudiant(1A127, exam) = @_3,$
$étudiant(1A452, exam) : @_4,$
$étudiant(1A452, exam) = @_4,$
$étudiant(1A127, moy) = @_5,$
$étudiant(1A452, moy) = @_6\}$

Remarque A Bien que l'aspect suivant ne soit pas illustré par l'exemple développé ici, il convient de noter que les constantes $@_i$, $i \in \mathbb{N}$ sont seulement des marqueurs, et deux constantes $@_j$ et $@_k$ avec $j \neq k$ peuvent être associées avec la même constante une fois les entrées différées connues. Ainsi, durant le processus d'instanciation, nous permettons le remplacement de deux occurrences de la même variable par deux constantes différentes $@_j$ et $@_k$, et préservons la coréférence en ajoutant un test $(@_j = @_k)$ dans l'instance de la règle concernée. □

Soit $\mathsf{P}_{\mathsf{inst}}$ le programme P où les règles R_1, R_2 sont remplacées par leurs instances closes :

R'$_1$

$$\begin{aligned} \textit{étudiant}(1A127, moy) = @_5 \longleftarrow\ & \textit{étudiant}(1A127, td) = @_1, \\ & \textit{étudiant}(1A127, exam) = @_3, \\ & @_5\ is\ (@_1 + 2 \times @_3)/3. \end{aligned}$$

R"$_1$

$$\begin{aligned} \textit{étudiant}(1A452, moy) = @_6 \longleftarrow\ & \textit{étudiant}(1A452, td) = @_2, \\ & \textit{étudiant}(1A452, exam) = @_4, \\ & @_6\ is\ (@_2 + 2 \times @_4)/3. \end{aligned}$$

R'$_2$

$$\begin{aligned} \textit{étudiant}(1A127, moy) = @_5 \longleftarrow\ & \textit{étudiant}(1A127, td) = @_1, \\ & \textit{étudiant}(1A127, exam) : abs, \\ & @_5\ is\ @_1. \end{aligned}$$

R"$_2$

$$\begin{aligned} \textit{étudiant}(1A452, moy) = @_6 \longleftarrow\ & \textit{étudiant}(1A452, td) = @_2, \\ & \textit{étudiant}(1A452, exam) : abs, \\ & @_6\ is\ @_2. \end{aligned}$$

Les règles obtenues sont suffisamment instanciées pour que l'on puisse associer à chaque référence de cellule apparaissant en tête de règle un pseudo-code permettant de calculer le contenu de la cellule. Par exemple pour la référence $\textit{étudiant}(1A127, moy)$ les deux règles R'$_1$ et R'$_2$ définissant le contenu de cette cellule correspondent au pseudo-code suivant :

SI étudiant(1A127,td) contient une donnée concrète
ET étudiant(1A127,exam) contient une donnée concrète ALORS
étudiant(1A127,moy) contient
(2 × étudiant(1A127,exam)+ étudiant(1A127,td))/3
SINON
SI étudiant(1A127,td) contient une donnée concrète
ET étudiant(1A127,exam) contient le nom "abs" ALORS

étudiant(1A127,moy) contient étudiant(1A127,td)
SINON
étudiant(1A127,moy) contient "non défini"
FIN SI
FIN SI

3.3.4 Etape 4

Nous abordons à présent l'étape 4 du processus qui consiste en la génération de la feuille de calcul au niveau physique. Pour réaliser cette implantation, il est nécessaire de répondre aux quatre questions suivantes :

1. quel est l'emplacement physique dans la grille correspondant à chacune des références de cellules ?
2. Quelles sont les cellules servant d'entrées différées ?
3. Quelles sont les cellules contenant une constante connue ? Et quelles sont ces constantes ?
4. Quelles sont les cellules contenant une formule ? Quelles sont ces formules ?

Le point 1 ne semble pas pouvoir être résolu automatiquement de façon satisfaisante en général, même si l'on peut imaginer diverses stratégies de placement par défaut [27]. Le développeur doit donc intervenir pour spécifier le placement physique en décrivant par exemple des contraintes de positionnement ou encore en indiquant des emplacements de façon interactive par l'intermédiaire d'un outil adéquat. Dans l'exemple traité, la spécification du placement peut être réalisée en fixant les positions des lignes et des colonnes pour le tableau *étudiant* sur une grille telle que celle de la figure IV.9.

Pour le point 2, en revanche, la détermination des cellules utilisées pour la saisie des entrées différées peut s'effectuer automatiquement en considérant que ce sont les cellules dont le contenu n'est ni connu ni défini par une des règles. Il s'agit donc des cellules de A contenant une constante de la forme $@_i$ et dont la référence de cellule n'apparaît en tête d'aucune règle de $\mathsf{P}_{\mathsf{inst}}$. Dans l'exemple présenté, ce sont les quatre cellules de références :
$\acute{e}tudiant(1A127, td)$, $\acute{e}tudiant(1A452, td)$,
$\acute{e}tudiant(1A127, exam)$, $\acute{e}tudiant(1A452, exam)$.

En ce qui concerne le point 3, l'ensemble des cellules dont le contenu est connu de façon certaine, alors que D n'est pas encore donné, correspond au modèle minimal de P pour I fourni et pour D vide. Pour notre exemple, l'ensemble de cellules obtenu est réduit à I lui-même : $\{\acute{e}tudiant(1A127, nom) : aaa, \acute{e}tudiant(1A452, nom) : bbb\}$

Enfin, pour le point 4, l'ensemble des cellules contenant une formule se construit à partir de l'ensemble des références de cellules apparaissant dans

A, duquel on élimine les références des cellules contenant des entrées différées ou des constantes (déterminées respectivement aux points 2 et 3). Pour chacune des références ainsi obtenues, la formule à placer dans la case correspondante de la grille est déterminée par le pseudo-code associé à cette référence à la fin de l'étape 3 de la génération. La traduction de ce pseudo-code en formule est facilité car les tableurs existants incluent dans leur langage de description de formules une fonction permettant d'indiquer des évaluations conditionnelles. Par exemple, dans le cas d'Excel [1], il s'agit de la fonction SI utilisant trois paramètres: $SI(Exp1; Exp2; Exp3)$ où $Exp1$ est une expression de type booléen, et où $Exp2$ et $Exp3$ sont des expressions sans restriction de type. La valeur de l'expression $SI(Exp1; Exp2; Exp3)$ est celle de $Exp2$ si $Exp1$ vaut le booléen *vrai*, ou celle de $Exp3$ dans le cas contraire.

Dans notre exemple, l'ensemble des références des cellules contenant une formule est: $\{étudiant(1A127, moy), étudiant(1A452, moy)\}$. Si l'on tient compte des positions imposées par la figure IV.9, dans la case E2 nous plaçons la traduction (dans le langage d'Excel) du pseudo-code associé à la référence $étudiant(1A127, moy)$, c'est à dire la formule:

=SI(ET(ESTNUM(D2);ESTNUM(C2));
*(2*C2+D2)/3;*
SI(ET(ESTNUM(D2);C2="abs"); D2; "non défini"))

dans laquelle ET dénote l'opérateur booléen de conjonction et l'appel ESTNUM(X) vaut *vrai* si et seulement si la case X contient une constante numérique. De même nous plaçons dans la case E3 (cellule de référence $étudiant(1A452, moy)$) la formule:

=SI(ET(ESTNUM(D3);ESTNUM(C3));
*(2*C3+D3)/3;*
SI(ET(ESTNUM(D3);C3="abs"); D3; "non défini"))

3.4 Exemple complet de spécifications de feuille de calcul

Cette section décrit un cas complet d'utilisation de notre méthode pour produire une feuille de calcul utilisée pour une gestion de scolarité.

Nous reprenons l'exemple de la section 2.2 en considérant les règles de gestion suivantes:

1. pour chaque semestre, la moyenne est calculée en appliquant les coefficients 2 pour la note d'examen, 1 pour la note de *td* et 1 pour la note de *tp* (obtenue lors d'un projet). Les redoublants ne réalisant pas de projet, leur moyenne est calculée simplement avec les notes de *td* et d'examen;

2. une absence excusée à l'examen conduit à une absence de note (marquée "abs") sans aucune pénalité pour le calcul de la moyenne, alors que pour une absence non excusée la note sera de zéro;

3. les groupes de projet sont constitués différemment d'un semestre sur l'autre.

On désire produire les squelettes de 3 tableaux, nommés *étudiant, grP s1* et *grP s2* , décrits figure IV.10. Dans le tableau *étudiant*, chaque ligne est identifiée par le code de l'étudiant. Le tableau nommé *grP s1* est structuré de manière à regrouper les étudiants par numéro de projet, et servira de grille de saisie pour chaque note de projet au premier semestre. Le tableau nommé *grP s2* joue le même rôle pour le second semestre. Ces notes seront ensuite "propagées" dans le tableau *étudiant* comme notes de *tp*. On remarquera que lors de la structuration du tableau *grP s1* (resp., *grP s2*), seuls les non redoublants sont regroupés en projet.

étudiant	nom	type	s1					s2		
			grP	exam	td	tp	moy	grP	exam	...
1A127	aaa	n	1					1		
1A452	bbb	n	2					1		
1A371	ccc	r								
1A214	ddd	n	1					2		
1A509	eee	n	3					1		
1A410	fff	n	2					2		
1A163	ggg	n	2					3		
1A498	hhh	n	3					2		
1A472	iii	r								
1A341	jjj	n	1					3		
1A769	kkk	n	3					3		

grP s1		note
1	aaa	
	ddd	
	jjj	
2	bbb	
	fff	
	ggg	
3	eee	
	hhh	
	kkk	

grP s2		note
1	aaa	
	bbb	
	eee	
2	ddd	
	fff	
	hhh	
3	ggg	
	jjj	
	kkk	

FIG. IV.10 - *Les tableaux* étudiant, grP s1 *et* grP s2

On suppose que les informations concernant le code de chaque étudiant, son nom, son type (*r* ou *n* pour redoublant ou non) et son numéro de projet pour chaque semestre sont obtenues auprès d'une base de données contenant une relation *scolarité* de la forme *scolarité(code, nom, type, grPs1, grPs2)*, avec *code* étant une clé primaire et *nom* une clé secondaire. On considère que les données de cette base font partie de l'entrée initiale et qu'elles peuvent être accédées en utilisant des atomes Datalog classiques dans le corps des règles DaTabLog.

Voici la description du programme DaTabLog correspondant aux spécifications ci-dessus.

Extraction des données de la base de données relationnelle :
$étudiant(C, nom) : N \longleftarrow scolarité(C, N, T, G1, G2).$
$étudiant(C, type) : T \longleftarrow scolarité(C, N, T, G1, G2).$
$étudiant(C, s1 \cdot grP) : G1 \longleftarrow scolarité(C, N, T, G1, G2).$
$étudiant(C, s2 \cdot grP) : G2 \longleftarrow scolarité(C, N, T, G1, G2).$

Spécification des entrées différées :
$étudiant(C, s1 \cdot exam) = @ \longleftarrow étudiant(C, nom) : Nom.$
$étudiant(C, s1 \cdot exam) : @ \longleftarrow étudiant(C, nom) : Nom$
$étudiant(C, s1 \cdot td) = @ \longleftarrow étudiant(C, nom) : Nom$
$étudiant(C, s2 \cdot exam) = @ \longleftarrow étudiant(C, nom) : Nom$
$étudiant(C, s2 \cdot exam) : @ \longleftarrow étudiant(C, nom) : Nom$
$étudiant(C, s2 \cdot td) = @ \longleftarrow étudiant(C, nom) : Nom$
$grP \cdot S(NumProjet \cdot Nom, note) = @ \longleftarrow$
$étudiant(C, S \cdot grP) : NumProjet,$
$étudiant(C, type) : n,$
$étudiant(C, nom) : Nom.$

Moyenne du semestre S pour un non redoublant (sans absence) :
$étudiant(C, S \cdot moy) = X \longleftarrow étudiant(C, S \cdot exam) = Y,$
$étudiant(C, S \cdot td) = Z,$
$étudiant(C, S \cdot tp) = W,$
$étudiant(C, type) : n,$
$X\ is\ (2 \times Y + W + Z)/4.$

Moyenne du semestre S pour un redoublant (sans absence) :
$étudiant(C, S \cdot moy) = X \longleftarrow étudiant(C, S \cdot exam) = Y,$
$étudiant(C, S \cdot td) = Z,$
$étudiant(C, type) : r,$
$X\ is\ (2 * Y + Z)/3.$

Moyenne du semestre S pour un non redoublant (absent à l'examen) :
$étudiant(C, S \cdot moy) = X \longleftarrow étudiant(C, S \cdot exam) : abs,$
$étudiant(C, S \cdot td) = Z,$
$étudiant(C, S \cdot tp) = W,$
$étudiant(C, type) : n,$
$X\ is\ (W + Z)/2.$

Moyenne du semestre S pour un redoublant (absent à l'examen) :

$$\begin{aligned} \acute{e}tudiant(C, S \cdot moy) = X \longleftarrow\ & \acute{e}tudiant(C, S \cdot exam) : abs, \\ & \acute{e}tudiant(C, S \cdot td) = Z, \\ & \acute{e}tudiant(C, type) : r, \\ & X\ is\ Z. \end{aligned}$$

Propagation de la note de projet du semestre S comme note de tp :

$$\begin{aligned} & \acute{e}tudiant(C, S \cdot tp) = NoteProjet \longleftarrow \\ & \qquad grP \cdot S(NumProjet \cdot Nom, note) = NoteProjet, \\ & \qquad \acute{e}tudiant(C, nom) : Nom. \end{aligned}$$

Nous avons utilisé sur ces spécifications une implantation prototype écrite en SWI-Prolog [49] de la technique présentée dans la section 3. Ceci nous a permi de générer automatiquement une feuille de calcul Excel opérationnelle correspondante, en éliminant le risque d'introduction manuelle d'erreurs lors du codage au niveau physique de la feuille de calcul.

4 Conclusion

Des études expérimentales ont montré que les applications à base de tableurs contenaient un nombre d'erreurs très important [11, 20, 39]. Cet état de fait est d'autant plus inacceptable que ces applications sont utilisées de façon intensive comme support d'aide à la décision [18]. Dans le but d'améliorer la qualité de ces développements, les travaux pionniers de [30, 29] isolent clairement les niveaux physique et logique dans les feuilles de calcul. Ils conduisent à mieux séparer les activités de conception et de codage, et permettent de définir précisément à un niveau logique la structuration des informations. En revanche ils ne prennent pas en compte la nature multidimensionnelle de l'information, inhérente à ce type d'applications, ni a fortiori sa manipulation.

Le cadre présenté au chapitre III offre un modèle de données simple et classique, qui respecte les propositions de [30, 29], et les enrichit à la prise en compte, à un niveau abstrait, de la description des manipulations de données fondamentales utilisées dans les applications à base de tableurs (e.g., restructurations de tables, sélections d'informations, calculs numériques).

Le langage DaTabLog possède une sémantique déclarative simple qui permet une lecture et une écriture facile des requêtes. Les programmes DaTabLog admettent une traduction en Datalog, étendu avec un langage de contraintes (une idée d'un tel codage est suggérée dans la section 5 du chapitre III). Néanmoins, cette traduction n'est ni naturelle, ni suggestive d'une technique d'évaluation spécifique, alors qu'une syntaxe et une sémantique directe offrent un niveau de formulation intuitif et concis.

Contrairement aux travaux présentés dans le chapitre II, le cadre que nous proposons s'adapte de manière naturelle au domaine des tableurs. Ces travaux n'ont pas été effectués dans le but de décrire les manipulations de données dans les feuilles de calcul, et leur utilisation dans ce contexte reste difficile. En effet, ils n'offrent pas de gestion directe, à un niveau logique, de certains aspects essentiels tels que l'imbrication des noms de lignes ou de colonnes, ou encore l'utilisation conjointe de tables et de formules de calcul sur un domaine concret.

Le travail que nous avons présenté est, à notre connaissance, la première formalisation d'un langage déclaratif issu du paradigme relationnel permettant de décrire à un niveau logique les manipulations de données utiles dans les applications à base de tableurs. Sans toutefois constituer actuellement une véritable méthode de conception, ce travail fournit des éléments importants permettant de limiter les sources d'erreurs lors du développement d'applications à base de tableurs. Tout d'abord dans la phase de spécification, DaTabLog autorise une description claire et précise de ce que l'on veut réaliser. De plus dans la phase d'implantation, nous avons montré que la génération de la feuille de calcul au niveau physique pouvait être réalisée de façon quasi-automatique, à partir de ses spécifications écrites en DaTabLog. La technique sous-jacente, présentée dans la section 3, repose sur l'utilisation d'un opérateur de conséquence immédiate approximant (au sens de [19]) celui associé à DaTabLog [8, 25]. Cette technique a été implantée en SWI-Prolog [49] pour générer des feuilles de calcul opérationnelles dans Excel [1]. Notons aussi qu'une sémantique opérationnelle naïve, sous la forme d'un calcul de plus petit point fixe tel que décrit au chapitre III, permet “d'exécuter” directement les spécifications à des fins de prototypage. Cette sémantique opérationnelle a d'ailleurs fait l'objet d'une implantation sous la forme d'un méta-interprète réalisé dans l'environnement ECLiPSe de l'ECRC, qui génère le modèle minimal d'un ensemble de clauses DaTabLog, à partir de la technique d'évaluation ascendante décrite dans [37].

Cette partie ne constitue qu'une première étape dans notre réponse au problème de qualité dont souffrent les applications à base de tableurs. Certains développements ont déjà fait l'objet d'investigations :

- DaTabLog admet une sémantique opérationnelle naïve qui peut donner lieu à des travaux d'optimisation du coût d'évaluation des requêtes, ou de vérification statique de propriétés. Par exemple [43] étudie des possibilités d'interprétation abstraite de ce langage ;

- DaTabLog est un langage de haut niveau, mais textuel. S'il est parfaitement adapté aux besoins d'un développeur ou d'un utilisateur avisé désirant spécifier des traitements complexes, il ne peut se substituer à un langage de type “clic-souris” intégré à une interface conviviale opérant sur des métaphores graphiques. [38] et [9] sont des propositions pour prendre en compte les aspects langages pour utilisateurs ou la manipulation de graphiques dans les tableurs.

Si d'autres aspects restent à prendre en compte (tels que le support des agrégats lors de la génération de feuilles de calcul), il semble possible dès à présent de dépasser le niveau langage pour intégrer notre proposition dans une perspective plus large, et proposer une véritable méthode d'élaboration d'applications à base de tableurs. [9] est un premier pas dans cette direction, en envisageant l'apport de DaTabLog dans le contexte des systèmes d'information intégrés, et propose l'architecture d'un atelier de développement de feuille de calcul.

Chapitre V

Conclusion

1 Contributions

Nous pensons que la contribution du travail présenté dans ce mémoire peut être vue selon deux axes :

- un approfondissement des connaissances sur les domaines émergeant que sont les systèmes OLAP, les entrepôts de données et les bases de données multidimensionnelles d'une part, et
- une utilisation de Datalog au travers d'une extension originale d'autre part.

Concernant le premier axe, les traitements OLAP sont présentés via une description intuitive des manipulations de données élémentaires, une classification et un éventail de formalisations de ces manipulations, et des exemples typiques les mettant en jeux.

Cette présentation est complétée par le deuxième axe, qui est centré sur la proposition d'un cadre formel intègrant langage de règles, traitements analytiques et données multidimensionnelles. Le langage conçu combine des mécanismes issus de travaux précédents, comme l'extension de Datalog avec des fonctions monovaluées [3], Hilog [16] et la F-logic [32], et est pourvu d'une sémantique déclarative et d'une sémantique opérationnelle de type respectif théorie des modèles et calcul de plus petit point fixe.

Les propriétés de ce cadre formel sont notamment utilisées pour caractériser la classe de manipulations calculables en temps polynomial sur des bases de données multidimensionnelles ordonnées. Elles sont aussi mises à profit pour palier le défaut en niveau logique des logiciels de type tableurs, en servant de base à l'élaboration d'une méthode semi-automatique de génération de feuilles de calcul.

2 Perspectives

Rappelons tout d'abord que le cadre proposé reste une base, et à ce titre, les pistes pour de futures recherches sont nombreuses. Il est certes motivant de poursuivre l'élaboration du cadre au travers de la sémantique du langage, mais l'étude menée au chapitre IV montre que la réflexion sur les tableurs mérite un travail à part entière. Nous développons brièvement ces deux aspects, en nous intéressant notamment à l'aspect visuel.

2.1 Le langage

Poursuivre les travaux sur l'aspect langage permettrait de renforcer son intégration avec les travaux existants, et à terme de l'inclure dans la proposition d'un cadre unifié pour entrepôt de données, comme il en existe actuellement un pour les bases de données relationnelles. A ce titre, le cadre du chapitre III est une base prometteuse pour élargir la classe de traitements pris en compte, en proposant des sémantiques moins naïves, par exemple une sémantique "stratifiée" pour l'agrégation, basée sur les niveaux hiérarchiques.

Du point de vue opérationnel, le travail fourni un cadre approprié pour l'étude de l'optimisation de requête et la proposition de mécanismes d'évaluation tirant profit de la structure de l'information. Une voie originale est de s'appuyer sur l'aspect "représentation" des cubes pour l'optimisation de requêtes. Considérons que la représentation d'un cube soit celle de la figure II.5. Cette représentation dans un plan ne laisse apparaître que 3 des faces du cube. Ainsi, une requête visant à construire ce cube pourra être optimisée en omettant de collecter les cellules qui n'apparaissent pas à l'écran.

2.2 Les tableurs

Le travail du chapitre IV est une base prometteuse pour fiabiliser les applications faisant intervenir des traitements analytiques du type de ceux offerts par les tableurs. Nous voyons deux axes principaux pour poursuivre ce travail :

- considérer le tableur comme une composante d'un système d'information, et fiabiliser ses interactions avec les autres composantes ;
- étendre la prise en compte par le langage des traitements offerts par le tableur.

Dans le premier cas, le tableur peut maintenant apparaître comme un remplaçant avantageux de certains applicatifs associés à une base de données (e.g., grilles de saisie, états). En effet, une extension du langage DaTabLog telle que nous la proposons semble être un bon candidat pour faire le lien entre les deux applications (e.g., spécification de grilles de saisies, réalisation d'états complexes). Cela peut se faire notamment en étudiant comment le tableur peut constituer une interface "normalisée" pour base de données dénormalisée, via une correspondance entre la propriété de monovaluation des cellules pour le tableur et les dépendances fonctionnelles de la base de données.

Dans le deuxième cas, le cadre peut être utilisé pour sécuriser d'autres fonctionnalités permises par les tableurs. Ainsi, une extension simple et originale du langage est la prise en compte des aspects graphiques tels qu'on les trouve dans Excel [1]. L'intuition donnée au chapitre II est qu'un graphique n'est qu'une représentation particulière des données d'un tableau (ou d'un cube). Supposons que l'on dispose d'un certain nombre de mots-clés du langage permettant de définir comment on construit un graphique, à partir de données sélectionnées dans un tableau. Par exemple, si le mot-clé *histo3D* indique que l'on désire un histogramme tridimensionnel, la règle suivante

spécifierait la construction du graphique de la figure V.2 à partir du tableau de la figure V.1 :
$histo3D \cdot ventes1996(P, R, V) : \langle\rangle \longleftarrow ventes1996(P, R) : V.$

ventes 1996	est	ouest	sud	nord
écrous	50	60	40	
clous	70			40
vis		50	50	60

FIG. V.1 - *Le tableau* ventes 1996

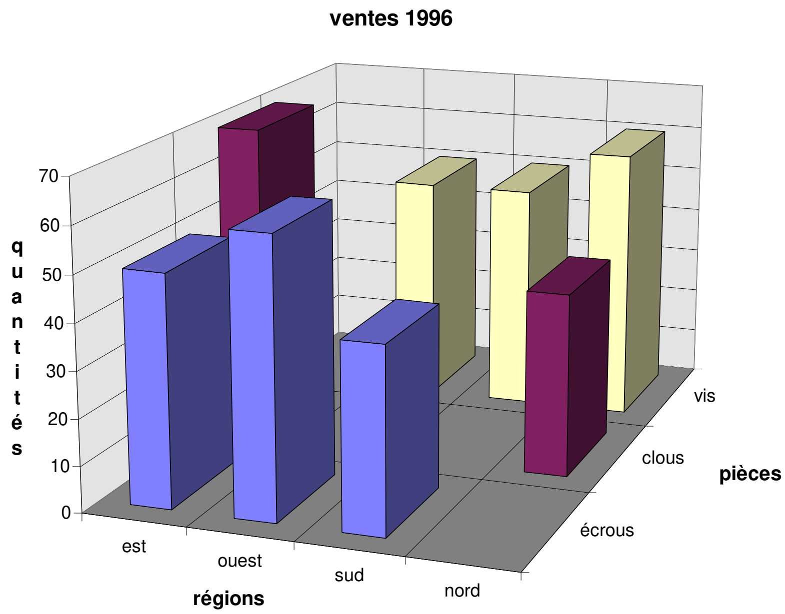

FIG. V.2 - *Le graphique* histo3D ventes 1996

histo3D·ventes 1996 donne son nom au graphique, et les valeurs de P, R, et V constituent les étiquettes de chaque dimension. La partie droite de la règle indique la provenance des valeurs affichées. La différence entre tableau et graphique est gommée, puisque le graphique peut être interrogé ou manipulé comme un tableau. Par exemple la règle suivante fait subir une rotation au graphique de la figure V.2 :
$histo3D/2 \cdot ventes1996(R, P, V) : \langle\rangle \longleftarrow histo3D \cdot ventes1996(P, R, V) : \langle\rangle.$

et cette dernière fournit un graphique pour la région *est* seulement (figure V.3) :

$$histo2D \cdot est(P, V) : \langle\rangle \longleftarrow histo3D \cdot ventes1996(P, est, V) : \langle\rangle.$$

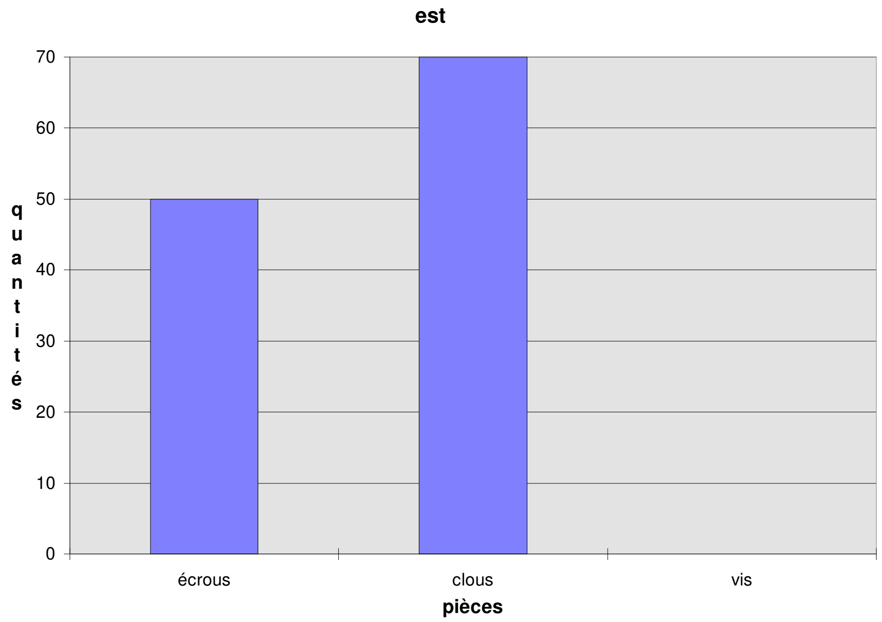

FIG. V.3 - *Le graphique* est

Un travail dans cette direction a été initié dans [9].

Bibliographie

[1] *EXCEL V5.0 user's guide*, 1993, Microsoft Corporation édition. 871 p.

[2] Abiteboul S. and Grumbach S. A rule-based language with functions and sets. *ACM Transactions on Database Systems*, 1991, vol. 16, n° 1, p. 1–30.

[3] Abiteboul S. and Hull R. Data functions, Datalog and negation. *In : Proceedings of the ACM SIGMOD International Conference on Management of Data*, Chicago, Illinois, USA, June 1988, p. 143–153.

[4] Abiteboul S., Hull R. and Vianu V. *Foundations of databases*, Reading, Massachusetts, USA : Addison-Wesley, 1995. 685p.

[5] Abiteboul S. and Kanellakis P. C. Object identity as a query language primitive. *In : Proceedings of the ACM SIGMOD International Conference on Management of Data*, Portland, Oregon, USA, June 1989, p. 159–173.

[6] Agrawal R., Gupta A. and Sarawagi S. *Modeling multidimensional databases*, San Jose, CA : IBM Almaden research center, 1996, 23p., Research report.

[7] Agrawal R., Gupta A. and Sarawagi S. Modeling multidimensional databases. *In : Proceedings of the Thirteenth International Conference on Data Engineering*, Birmingham, United Kingdom, April 1997, p. 232–243.

[8] Boulicaut J.-F., Hacid M.-S., Marcel P. and Rigotti C. *Un langage de manipulation de données pour feuilles de calcul*, INSA de Lyon : LISI, January 1997, 24p., Rapport de recherche n° RR-97-01.

[9] Boulicaut J.-F., Marcel P. and Pinet F. *Les tableurs : un talon d'achille pour les systèmes d'informations intégrés*, INSA de Lyon : LISI, January 1998, 28p., Rapport de recherche n° RR-98-02.

[10] Boulicaut J.-F., Marcel P., Pinet F. and Rigotti C. Spreadsheet generation from rule-based specifications. *In : Proceedings of the Sixth International Workshop on Deductive Database and Logic Programming*

(DDLP'98), edited by Fraternali P., Geske U., Ruiz C. and Seipel D., Manchester, United Kingdom, June 1998, p. 59–70.

[11] Brown P. and Gould P. An experimental study of people creating spreadsheets. *ACM Transactions on Information Systems*, 1987, vol. 5, n° 3, p. 258–272.

[12] Cabibbo L. and Torlone R. Querying multidimensional databases. *In : Proceedings of the Sixth International Workshop on Database Programming Languages*, Estes Park, Colorado, USA, August 1997, p. 17.

[13] Cabibbo L. and Torlone R. A logical approach to multidimensional databases. *In : Proceedings of the Sixth International Conference on Extending Database Technology*, Valencia, Spain, March 1998, p. 183–197.

[14] Chaudhuri S. and Dayal U. An overview of data warehousing and OLAP technology. *ACM SIGMOD RECORDS*, 1997, vol. 26, n° 1, p. 65–74.

[15] Chaudhuri S. and Kyuseok S. Optimization of queries with user-defined predicates. *In : Proceedings of the Twenty-second International Conference on Very Large Databases*, Bombay, India, 1996, p. 87–98.

[16] Chen W., Kifer M. and Warren D. HiLog: a foundation for higher-order logic programming. *Journal of Logic Programming*, 1993, vol. 15, n° 3, p. 187–230.

[17] Codd E. F. A relational model for large shared data banks. *Communications of the ACM*, 1970, vol. 13, n° 6, p. 377–387.

[18] Codd E. F., Codd S. B. and Salley C. T. Providing OLAP (On-Line Analytical Processing) to user-analysts: An IT mandate [on-line]. 1993. 31p. White Paper Available from internet : <URL : http://www.arborsoft.com/essbase/wht_ppr/coddTOC.html>.

[19] Cousot P. and Cousot R. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 1992, vol. 13, n° 2/3, p. 103–179.

[20] Cragg P. B. and King M. Spreadsheet modeling abuse: an opportunity for OR? *Journal of the Operational Research Society*, 1993, vol. 44, n° 8, p. 743–752.

[21] Finkelstein R. Understanding the need for on-line analytical servers [on-line]. 1995. 10p. White Paper Available from internet : <URL : http://www.arborsoft.com/essbase/wht_ppr/finkTOC.html>.

[22] Gray J., Bosworth A., Layman A. and Pirahesh H. Data cube : a relational operator generalizing group-by, cross-tab, and sub-totals. *In : Proceedings of the Twelfth International Conference on Data Engineering*, New Orleans, Louisianne, USA, February 1996, p. 152–159.

[23] Gyssens M. and Lakshmanan L. V. S. A foundation for multidimensional databases. *In : Proceedings of the Twenty-third International Conference on Very Large Databases*, Athens, Greece, August 1997, p. 106–115.

[24] Gyssens M., Lakshmanan L. V. S. and Subramanian I. N. Tables as a paradigm for querying and restructuring. *In : Proceedings of the Fifteenth ACM Symposium on Principles of Database Systems*, Montreal, PQ, Canada, June 1996, p. 93–103.

[25] Hacid M. S., Marcel P. and Rigotti C. A rule-based CQL for 2-dimensional tables. *In : Proceedings of the Second International Workshop on Constraint Database Systems (CDB'97)*, Delphi, Greece, January 1997, p. 92–104.

[26] Hacid M. S., Marcel P. and Rigotti C. A rule-based data manipulation language for OLAP systems (short paper). *In : Proceedings of the Fifth International Conference on Deductive and Object Oriented-Databases*, Montreux, Switzerland, December 1997, p. 417–418.

[27] Hacid M. S., Marcel P. and Rigotti C. A rule-based language for ordered multidimensional databases. *In : Proceedings of the Fifth International Workshop on Deductive Database and Logic Programming (DDLP'97)*, edited by Geske U., Ruiz C. and Seipel D., Leuven, Belgium, July 1997, p. 69–81.

[28] Hacid M. S., Marcel P. and Rigotti C. Rule-based restructuring and summarizing in multidimensional databases. *In : Actes des Journées Francophones de Programmation en Logique et par Contrainte*, Nantes, France, May 1998, p. 115–134.

[29] Hainaut J.-L. *Bases de données et modèles de calcul : outils et méthodes pour l'utilisateur*, Paris : InterEditions, 1994. 364p.

[30] Isakowitz T., Schocken S. and Lucas JR H. C. Towards a logical/physical theory of spreadsheet modeling. *ACM Transactions on Information Systems*, 1995, vol. 13, n° 1, p. 1–37.

[31] Kemp D. B. and Stuckey P. J. Optimizing bottom-up evaluation of constraint queries. *Journal of Logic Programming*, 1996, vol. 26, n° 1, p. 1–30.

[32] Kifer M., Lausen G. and Wu J. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 1995, vol. 42, n° 4, p. 741–843.

[33] Klug A. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the ACM*, 1982, vol. 29, n° 3, p. 699–717.

[34] Lenz H. J. and Shoshani A. Summarizability in olap and statistical databases. *In : Proceedings of the Ninth International Conference on Statistical and Scientific Databases Management*, Olympia, Washington, USA, August 1997.

[35] Li C. and Wang X. S. A data model for supporting on-line analytical processing. *In : Proceedings of the Fifth International Conference on Information and Knowledge Representation*, Rockville, Maryland, USA, November 1996, p. 81–88.

[36] Maier D. and Vance B. A call to order. *In : Proceedings of the Twelfth ACM Symposium on Principles of Database Systems*, Washington, DC, USA, May 1993, p. 1–16.

[37] Manthey R. and Bry F. Satchmo: a theorem prover implemented in prolog. *In : Proceedings of the 9th International Conference on Automated Deduction CADE'88*, edited by Lusk E. L. and Overbeek R. A., Argonne, Illinois, USA, May 1988, p. 456–459.

[38] Marcel P. *Un SQL pour tableurs*, INSA de Lyon : LISI, January 1997, 22 p., Rapport de recherche n° RR-97-01.

[39] Panko R. R. and Halverson R. P. Spreadsheets on trial: a survey of research on spreadsheet risks. *In : Proceedings of the Twenty-Ninth Hawaï International Conference on System Sciences*, Maui, Hawaï, January 1996, p. 326–335.

[40] Papadimitriou C. P. A note on the expressive power of prolog. *Bulletin of the EATCS*, 1985, vol. 26, p. 21–23.

[41] Pendse N. and Creeth R. The olap report [on-line], 1998. Web site Available from internet : <URL : http://www.olapreport.com>.

[42] Pilot Software. An introduction to OLAP [on-line], 1995. 23p. White Paper Available from internet : <URL : http://www.pilotsw.com/r_and_t /whtpaper/olap/olap.htm>.

[43] Pinet F. *Analyse Statique d'un Langage de Requêtes sous Contraintes*. Mémoire de DEA, INSA de Lyon, LISI, July 1997. 31p.

[44] Ronen B., Palley M. A. and Lucas H. C. Spreadsheet analysis and design. *Communications of the ACM*, 1989, vol. 32, n° 1, p. 84–93.

[45] Ross K. A. On negation in hilog. *Journal of Logic Programming*, 1994, vol. 18, n° 1, p. 27–53.

[46] Shoshani A. Olap and statistical databases: Similarities and differences. *In : Proceedings of the Sixteenth ACM Symposium on Principles of Database Systems*, Tucson, Arizona, USA, 1997, p. 185–196.

[47] Shukla A., Deshpande P. M., Naughton J. F. and Ramasamy K. Storage estimation for multidimensional aggregates in the presence of hierarchies. *In : Proceedings of the Twenty-second International Conference on Very Large Databases*, Bombay, India, September 1996, p. 522–531.

[48] Widom J. Research problems in data warehousing. *In : Proceedings of the Fourth International Conference on Information and Knowledge Representation*, edited by Pissinou N., Silberschatz A., Park E. K. and Makki K., Baltimore, Maryland, USA, November 1995, p. 25–30.

[49] Wielemaker J. *SWI-Prolog 2.9.6*, Amsterdam, The Netherlands : Dept. of Social Science Informatics (SWI). University of Amsterdam, 1997. 103p.

Annexe A

Rappels

1 Lexique

Nous rappelons quelques définitions concernant les bases de données relationnelles et les langages de manipulation associés.

Schéma de base de données Un schéma de base de données est un ensemble non vide de noms de relation.

Instance de base de données Une instance d'un schéma de base de données est une fonction associant à chaque nom de relation du schéma de la base de données un ensemble fini de tuples.

Si nous utilisons le terme fonction pour désigner des fonctions dont le domaine est constitué d'instances de schémas de base de données, nous avons les définitions suivantes :

Fonction close Pour un schéma de base de données, une fonction est dite close si, appliquée à une instance du schéma de la base de donnée, son résultat est aussi une instance du schéma la base de données.

Fonction calculable Une fonction f est calculable si il existe une machine de Turing qui, pour toute instance de base de données I donnée en entrée, termine avec la réponse $f(I)$ sur le ruban si $f(I)$ est définie, ou ne termine pas sinon.

Fonction générique Une fonction f est générique si pour toute instance de base de donnée I et toute permutation de constante ρ de I, $\rho(f(I)) = f(\rho(I))$.

Requêtes Une requête est une fonction partielle calculable et générique d'une instance de base de données vers une instance de base de données.

Un langage de requête (ou langage de manipulation de données) peut être vu comme l'ensemble des requêtes qu'il permet d'exprimer. En étudiant la classe de complexité des requêtes ainsi décrites, il est possible d'associer à un langage une puissance d'expression.

2 Langages de requête relationnels

Nous présentons maintenant les langages de requête relationnels, en les différenciant par les trois paradigmes qui leur ont donné naissance. Chaque paradigme engendre une famille de langages, qui ont en commun plusieurs propriétés : ce sont des langages de haut niveau, orientés tuples et *associatifs* (les tuples sont identifiés par leur contenu) [4]. Une propriété remarquable de ces familles est que chacune d'elles peut engendrer des langages dont les équivalents existent dans les autres familles.

L'algèbre relationnelle Le premier paradigme consiste en la définition d'opérateurs algébriques, unaires ou binaires, polymorphes, pour manipuler des instances de relations et construire les réponses aux requêtes. Ces opérateurs sont principalement des opérateurs ensemblistes (comme l'union ou le produit cartésien), auxquels viennent s'ajouter des opérateurs traitant les spécificités du modèle relationnel. Ces opérations spécifiques sont la *sélection*, qui permet d'isoler des tuples dont une composante satisfait certaines propriétés, et la *projection*, qui permet d'isoler certaines composantes d'un ensemble de tuples.

A titre d'exemple, si j est un entier positif, a une constante, I une instance de relation d'arité supérieure à j et $t(j)$ désigne la $j^{ième}$ composante du tuple t, alors l'opérateur de sélection, σ, est formellement défini par :

$$\sigma_{j=a}(I) = \{t \in I \mid t(j) = a\}$$

L'algèbre relationnelle est conceptuellement "procédurale" [4] : la formulation d'une requête consiste en une séquence d'opérations constructives. La forme générique d'une requête en algèbre relationnelle peut donc être décrite par :

$$op_1(op_2(\ldots(op_n(r_1, r_2), r_3)\ldots)r_m)$$

où les $op_i, i \in [1, \ldots, n]$ sont des opérateurs algébriques et les $r_j, j \in [1, \ldots, m]$ des instances de relations.

L'étude des propriétés des opérateurs algébriques a fourni des résultats importants en terme d'optimisation de requêtes et de génération de plan d'évaluation.

Le calcul relationnel La deuxième famille est issue du paradigme logique ; les langages sont des variantes du calcul des prédicats du premier ordre, privés des symboles de fonctions. Un prédicat est associé à une relation de même arité, et les variables ont comme domaine d'interprétation l'ensemble des constantes atomiques de la base. La forme générique d'une requête en calcul relationnel peut donc être décrite par :

$$\{x_1, \ldots, x_n \mid F(x_1, \ldots, x_n)\}$$

où F est une formule du premier ordre impliquant des relations de la base, et dont les seules variables libres sont $x_1, \ldots, x_n$. Intuitivement, F énonce les conditions à satisfaire pour constituer la réponse à la requête. Sa sémantique est celle d'une formule de la logique du premier ordre, et chaque instantiation des $x_1, \ldots, x_n$ satisfaisant F forme un des tuples de la réponse à la requête.

Le calcul relationnel est qualifié de langage *déclaratif*, car les tuples réponses sont spécifiés par des propriétés à satisfaire, sans référence à un plan d'accès ou un algorithme pour construire l'ensemble réponse.

Datalog Le dernier paradigme provient de la programmation logique, et donne naissance aux langages de la famille *Datalog*. Ce sont des langages de règles dont les variables sont quantifiées universellement, et dont le corps (les prémisses) est une conjonction d'atomes et la tête (le conséquent) est constitué d'au plus un atome. Sous sa forme équivalente au calcul relationnel et à l'algèbre relationnelle, Datalog, appelé alors $nr\text{-}Datalog^{\neg}$, admet une forme de négation (pour exprimer la différence ensembliste), mais interdit la définition de requêtes récursives, c'est à dire la définition d'une relation en fonction d'elle même. La forme générique d'une requête en Datalog peut être décrite par un ensemble de règles de la forme :

$$A \longleftarrow \; B_1, \ldots, B_n$$

où A est un atome, et les $B_i, i \in [1, \ldots, n]$ des littéraux éventuellement négatifs. La sémantique intuitive associée à une telle règle est : si les $B_i, i \in [1, \ldots, n]$ sont vrais, alors A est vrai.

Sans rajout syntaxique et sémantique, ni le calcul ni l'algèbre ne permettent d'exprimer la récursion. En revanche, si on admet pour les règles une sémantique ad-hoc, le langage Datalog offre un cadre approprié pour l'étude des requêtes récursives (e.g., techniques d'optimisation, sémantique en présence de négation ou d'agrégation).